A Decision-Theoretic Architecture for Selecting Tutorial Discourse Actions

R. Charles Murray
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260
rmurray@pitt.edu

Kurt VanLehn
LRDC
University of Pittsburgh
Pittsburgh, PA 15260
vanlehn@pitt.edu

Jack Mostow
Project LISTEN
Carnegie Mellon University
Pittsburgh, PA 15213
mostow@cs.cmu.edu

Abstract

We propose a decision-theoretic architecture for selecting tutorial discourse actions. *DT Tutor*, an action selection engine which embodies our approach, uses a dynamic decision network to consider the tutor's objectives and uncertain beliefs in adapting to the changing tutorial state. It predicts the effects of the tutor's discourse actions on the tutorial state, including the student's internal state, and then selects the action with maximum expected utility. We illustrate our approach with prototype applications for diverse target domains: calculus problem-solving and elementary reading. Formative off-line evaluations assess DT Tutor's ability to select optimal actions quickly enough to keep a student engaged.

1 Introduction

A tutoring system achieves many of its objectives through discourse actions intended to influence the student's internal state. For instance, a tutor might tell the student a fact with the intended effect of increasing the student's knowledge and thereby enabling her to perform a problem-solving step. The tutor might also be concerned with the student's goals, focus of attention, and affective or emotional state, among other internal attributes. However, a tutor is inevitably uncertain about the student's internal state, as it is unobservable. Compounding the uncertainty, the student's state changes throughout the course of a tutoring session–after all, that is the purpose of tutoring. To glean uncertain information about the student, a tutor must make inferences based on observable actions and guided by the tutor's beliefs about the situation. The tutor is also likely to be concerned with observable attributes of the tutoring situation, or tutorial state, including the discourse between tutor and student and their progress at completing tutorial tasks (e.g., solving problems).

The tutor's actions depend not only on the tutorial state, but also on the tutor's objectives. Tutorial objectives often include increasing the student's knowledge within a target domain, helping the student solve problems or complete other tasks, and bolstering the student's affective state (Lepper et al., 1993). Tutors also generally want to be cooperative discourse partners by coherently addressing topics that are relevant to the student's focus of attention. Objectives and priorities may vary by tutor and even for an individual tutor over time. Furthermore, tutors must often strike a "delicate balance" among multiple competing objectives (Merrill et al., 1992, p. 280).

To model the tutor's uncertainty about the student's internal state, probabilistic reasoning is becoming increasingly common. However, almost all probabilistic tutoring systems still model the tutor's objectives implicitly at best, and use heuristics to select tutorial actions. *DT Tutor* uses a decision-theoretic approach to select tutorial actions, taking into account both the tutor's uncertain beliefs and multiple objectives regarding the changing tutorial state. This paper describes DT Tutor's approach along with prototype applications for diverse domains, calculus problem-solving and elementary reading.

2 General Approach

2.1 Belief and Decision Networks

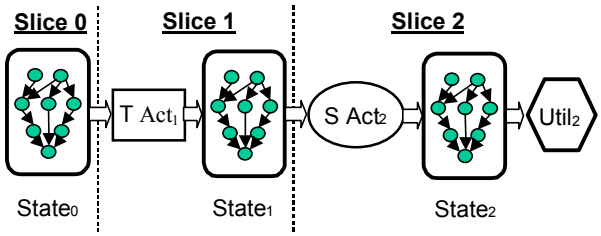

Figure 1. Tutor Action Cycle Network, overview

DT Tutor represents the tutor's uncertain beliefs in terms of probability using Bayesian belief networks. A belief network is a directed acyclic graph with *chance nodes* representing beliefs about attributes and *arcs* between nodes representing conditional dependence relationships among the beliefs. Beliefs are specified in terms of probability distributions. DT Tutor's chance nodes represent the tutor's beliefs about the tutorial state. For each node with incoming arcs, a conditional probability table specifies the probability distribution for that node conditioned on the possible states of its parents. For nodes without incoming arcs, prior probability distributions are specified.

At any particular time, each node within a belief network represents an attribute whose value is fixed. For an attribute whose value may change over time (such as a tutorial state attribute), separate nodes can be used to represent each successive value. Dynamic belief networks do just that. For each time in which the values of attributes may change, a dynamic belief network creates a new *slice*. Each slice is of a set of chance nodes representing attributes at a specific point in time. For tutoring, slices can be chosen to represent the tutorial state after a tutor or student action, when attribute values are likely to change. Nodes may be connected to nodes within the same or earlier slices to represent the fact that an attribute's value may depend on (1) concurrent values of other attributes and (2) earlier values of the same and other attributes.

Decision theory extends probability theory to provide a normative theory of how a rational decision-maker should behave. Quantitative utility values are used to express preferences among possible outcomes of actions. To decide among alternative actions, the expected utility of each alternative is calculated by taking the sum of the utilities of all possible outcomes weighted by the probabilities of those outcomes occurring. Decision theory holds that a rational agent should choose the alternative with maximum expected utility. A belief network can be extended into a decision network (equivalently, an influence diagram) by adding decision and utility nodes along with appropriate arcs. For DT Tutor, decision nodes represent tutorial action alternatives, and utility nodes represent the tutor's preferences among the possible outcomes.

A dynamic decision network (DDN) is like a dynamic belief network except that it has decision and utility nodes in addition to chance nodes. DDNs model decisions for situations in which decisions, attributes or preferences can change over time. The evolution of a DDN can be computed while keeping in memory at most two slices at a time (Huang et al., 1994).

2.2 General Architecture

DT Tutor's action selection engine uses a DDN formed from dynamically created tutor action cycle networks (TACNs). A TACN consists of three slices, as illustrated in Figure 1. The tutorial state ($State_s$) within each slice is actually a sub-network representing the tutor's beliefs about the tutorial state at a particular point in time (slice)[1]. The $T\ Act_1$ decision node represents the tutorial action decision, the $S\ Act_2$ chance node represents the student turn following the tutor's action, and the $Util_2$ utility node represents the utility of the resulting tutorial state.

Each TACN is used for a single cycle of tutorial action, where a cycle consists of deciding a tuto-

[1] For sub-network and node names, a numeric subscript refers to the slice number. A subscript of *s* refers to any appropriate slice.

rial action and carrying it out, observing the subsequent student turn, and updating the tutorial state based on the tutor and student actions. During the first phase (deciding upon a tutorial action), slice 0 represents the tutor's current beliefs about the tutorial state. Slice 1 represents the tutor's possible actions and predictions about their effects on the tutorial state. Slice 2 represents a prediction about the student's next turn and its effect on the tutorial state. The DDN update algorithm calculates which tutorial action has maximum expected utility.

In the next phase of the cycle, the tutor executes that action and waits for the student response. The tutor then updates the network based on the observed student action(s).

At this point, the posterior probabilities in $State_2$ represent the tutor's current beliefs. It is now time to select another tutor action, so another TACN is created and the DDN is rolled forward: Posterior probabilities from $State_2$ of the old TACN are copied as prior probabilities to $State_0$ of the new TACN, where they represent the tutor's current beliefs. The old TACN is discarded. The tutor is now ready to begin the next cycle by deciding which action to take next.

With this architecture, the tutor not only reacts to past student actions, but also anticipates future student actions and their ramifications. Thus, for instance, it can act to prevent errors and impasses before they occur, just as human tutors often do (Lepper et al., 1993).

In principle, the tutor can look ahead any number of slices without waiting to observe student actions. The tutor simply predicts probability distributions for the next student turn and the resulting $State_2$, rolls the DDN forward, predicts the tutor's next action and the following student turn, and so on. Thus, the tutor can select an optimal sequence of tutorial actions for any fixed amount of look ahead. However, a large amount of look ahead is computationally expensive with decreasing predictive accuracy.

3 Application Domains

3.1 Calculus Problem-Solving

CTDT (Calculus Tutor, Decision-Theoretic) is a prototype action selection engine for calculus related rates problems (Murray & VanLehn, 2000). Singley (1990) developed a tutoring system for this domain with an interface designed to make student problem-solving actions observable, including goal-setting actions that are normally invisible. CTDT presumes an extension to Singley's interface to make all problem-solving actions observable. This makes it easier to select tutorial actions for two reasons. First, as each problem-solving action is executed through the interface, CTDT has the opportunity to intervene. (However, CTDT can select a *null* action on its turn and thus allow the student to execute multiple actions without tutorial intervention). This means that CTDT can select a response for only a single student action per turn, rather than deciding which of multiple student actions to respond to. Moreover, it is easier to predict a single student action per turn than to predict a combination of multiple actions.

Second, when CTDT can observe all of the student's prior actions, it knows exactly what portion of the problem solution space the student had already completed and thus what steps the student is likely to attempt next. Calculus related rates problems, like problems in many other domains, have a prerequisite structure that induces a partial order in which problem steps may be completed – for instance, the chain rule (e.g., $dx/dy * dy/dz = dx/dz$) cannot be applied until the component equations are in the required form. The student is unlikely to be able to successfully complete problem steps for which prerequisites have not been completed, and is therefore less likely to attempt them. The student is also unlikely to repeat problem-solving steps that have already been completed successfully. This means that the student is most likely to attempt problem steps that (1) have not already been completed, and (2) have no uncompleted prerequisite steps. We

call these *ready* steps. Thus, by observing which steps the student has already completed, CTDT can easily determine the set of *ready* steps that the student is most likely to attempt next.

Even so, predicting the next student action is still not trivial, since there may be more than one way to solve a calculus related rates problem (i.e., more than one *solution path*), and there may be multiple orders in which the steps of a solution path can be executed.

3.2 Project LISTEN's Reading Tutor

RTDT (Reading Tutor, Decision-Theoretic) is a prototype action selection engine for Project LISTEN's Reading Tutor, which uses mixed-initiative spoken dialogue to provide reading help for children as they read aloud (Mostow & Aist, 1999). The Reading Tutor has helped to improve the reading of real students in real classrooms (Mostow & Aist, in press). It displays one sentence at a time for the student to read, and a simple animated persona that appears to actively watch and patiently listen. As the student reads, the Reading Tutor uses automated speech recognition to detect when the student may need help, which it provides using both speech and graphical display actions. Thus, the Reading Tutor already has an extensively developed interface. This is in contrast to CTDT, for which we assumed an interface built to our specifications. Inter-operability with existing tutoring systems is a key to extending the applicability of DT Tutor's approach.

RTDT models some of the Reading Tutor's key tutorial action decisions in just enough depth to determine the feasibility of applying DT Tutor to this domain. We targeted two types of unsolicited help: (1) *preemptive help* before the student attempts a sentence, and (2) *corrective feedback* after the student has stopped reading (whether or not the student has completed the sentence). The Reading Tutor provides preemptive help when it believes that the student is likely to misread a word, and corrective feedback when it detects words read incorrectly, skipped words and disfluent reading. To avoid disrupting the flow of reading, the Reading Tutor ignores errors on a list of 36 common function words (e.g., *a, the*) that are unlikely to affect comprehension. For the Reading Tutor's corpus of readings, approximately two-thirds of the words in a sentence are non-function words, or *content* words.

Tutoring reading differs enough from coaching calculus problem-solving to pose challenges for adapting DT Tutor's approach. First, student turns may consist of multiple reading actions, where each action is an attempt to read a word. Therefore, in contrast to CTDT, RTDT must predict and respond to multiple student actions per turn. Student turns may indeed include multiple actions in many target domains, so meeting this challenge is important for extending DT Tutor's generality.

Second, beginning readers often make repeated attempts at words or phrases and sometimes omit words, with the effect of jumping around within a sentence. Even when jumping around, a student may be able to read each individual word. Thus, the order in which beginning readers attempt words is not always sequential, and has very little prerequisite structure. This means that the set of actions that the student is likely to attempt next is less constrained than with CTDT, posing a challenge for predicting the student's next turn. A similar challenge must be faced for tutoring in any target domain with weak structure for the order in which actions may be completed.

4 Tutor Action Cycle Networks in More Detail

4.1 TACN Components

Figure 2 provides a closer look at the major TACN components and their interrelationships. The $State_s$ representation in each slice actually consists of several sub-networks. These include the $Knowledge_s$, $Focus_s$, and $Affect_s$ sub-networks which compose the student model, and the *Task* $Progress_s$ and *Discourse* $State_s$ sub-networks. Arcs between corresponding sub-networks in dif-

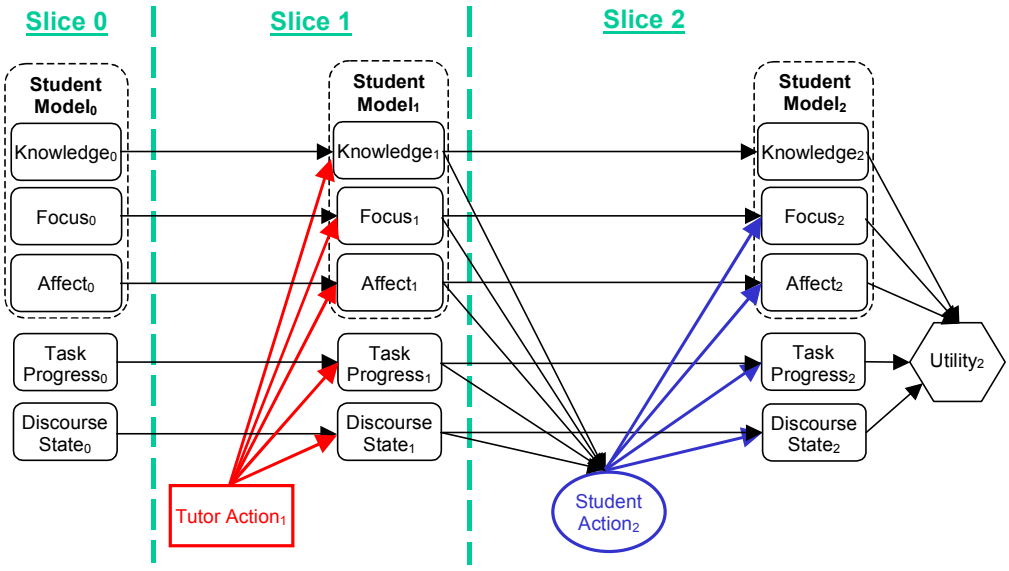

Figure 2. TACN architecture in more detail

ferent time slices represent the stability of attributes over time. For instance, the student's knowledge in slice 1, *Knowledge*$_1$, is likely to be about the same as the student's knowledge in slice 0, *Knowledge*$_0$, except as influenced by the tutor's action, *Tutor Action*$_1$.

The architecture shown in Figure 2 is generic. Depending on the needs of the application, fewer or more components may be required. For instance, the initial implementation of the RTDT prototype lacks a model of the student's affective state because we focused on modeling other tutorial state attributes, such as multiple student actions per turn. Therefore, its TACNs do not include the *Affect*$_s$ sub-networks. However, RTDT also has *Tutor Efficacy*$_s$ sub-networks to model the efficacy of the various tutorial help alternatives. The *Tutor Efficacy*$_s$ sub-networks dynamically tune RTDT's model of the effects of the tutor's actions on the student's knowledge, helping RTDT to avoid repeating ineffective tutorial actions and reducing the need for accurate conditional probabilities regarding the influence of *Tutor Action*$_1$ on *Knowledge*$_1$.

Selected components are described below along with illustrations from CTDT and RTDT.

4.1.1 *Tutor Action*$_1$ Nodes

The purpose of the TACN is to compute the optimal alternative for *Tutor Action*$_1$, which may consist of one or more decision nodes. For CTDT, *Tutor Action*$_1$ consists of two decision nodes, one to specify the *topic* of the tutor action and one to specify the action *type*. The action *topic* is the problem-related focus of the action, such as a problem step or related rule in the target domain. The *type* is the manner in which the topic is addressed, including *prompt, hint*, *teach*, *positive* or *negative feedback*, *do* (tell the student how to do a step) and *null* (no tutor action).

For RTDT, *Tutor Action*$_1$ is currently a single decision node with values *null* (no tutor action), *move_on* (move on to the next sentence), *read_move_on* (read the sentence to the student and then move on), *hint_sentence* (e.g., read the current sentence to the student), and *hint_word_i* for each content word *i* in the current *n*-content-word sentence, *i = {1, 2, …, n}*. The *hint_sentence* and *hint_word_i* alternatives specify the *topic* but not the *type* of the tutorial action – e.g., they don't specify whether the Reading Tutor should hint about a particular word by saying the word itself or by giving a rhyming hint. Deciding among action type alternatives would require information than was not available for the prototype implementation. For instance, information about

the student's knowledge of the letter-sound mappings pertinent to a particular word would help RTDT determine the likelihood that a rhyming hint would supply the required knowledge.

CTDT considers tutoring only on *ready* problem steps and related rules, plus the step that the student has just completed (e.g., to give *positive* or *negative feedback*). RTDT considers every action alternative for preemptive help, including hinting on each content word. However, for fast response time on corrective feedback, RTDT does not consider hinting on words that the student has already read correctly, because such hints are less likely to be pedagogically productive.

4.1.2 Student Model $Knowledge_s$ Sub-Network

The $Knowledge_s$ sub-network represents the tutor's beliefs about the student's knowledge related to the target domain. Each $Knowledge_s$ node has possible values *known* and *unknown*. For CTDT, the student's knowledge related to each problem is represented in a belief (sub-)network whose structure is obtained directly from a *problem solution graph*. See Figure 3 for an example. The top two rows of nodes in the figure represent rules licensing each problem step. The remaining nodes represent problem steps, from the givens (the goal *Find dx/dz for z=c* and the facts $x=ay^b$, $y=ez^f$ and $z=c$) through each goal-setting and fact-finding step in all solution paths (this example has only one solution path) until the answer is found ($dx/dz=bay^{b-1}fec^{f-1}$). Arcs represent dependence between nodes. For instance, knowledge of a step depends on knowledge of both its prerequisite steps and the rule required to derive it.

For RTDT, $Knowledge_s$ includes nodes to represent the student's knowledge of how to read each content word and the sentence. For each content word *i*, a $Know\_Word\_i_s$ node represents the student's knowledge of how to read the word. A $Know\_Sentence_s$ node represents the student's knowledge of how to read the sentence as a whole.

In slice 1, each $Knowledge_1$ node is influenced by the tutor's action. For instance, a tutorial hint about a particular problem step or word increases the probability that the node corresponding to the knowledge element is *known*. After the student turn has been observed, $Knowledge_1$ is updated diagnostically to reflect its causal role in the success of the student's action(s).

$Knowledge_2$ is not directly influenced by the student's turn because student actions generally do not influence student knowledge without feedback (e.g., by the tutor). Instead, $Knowledge_2$ is influenced by $Knowledge_1$, which is diagnostically influenced by the student's turn.

4.1.3 Student Model $Focus_s$ Sub-Network

The $Focus_s$ sub-network represents the student's focus of attention within the current tutorial task. For CTDT, the focus may be any problem step, so $Focus_s$ has the same problem solution graph structure as $Knowledge_s$. *Ready* steps are most likely to be in focus. Nodes representing these steps have some distribution over the values *ready* and *in_focus*, where *in_focus* means that the step is in the student's focus of attention. Consistent with a human depth-first problem-solving bias (Newell & Simon, 1972), any such steps that are in the student's current solution path are most likely to be *in_focus*. *Focus aging* is also modeled: the probability that an uncompleted step is *in_focus* attenuates with each passing time slice as other problem steps come into focus.

For RTDT, $Focus_s$ models the likelihood of each content word being the first word in the student's focus of attention. $Focus\_Word\_i_s$ nodes for each content word *i* in the current sentence have possible values *in_focus* and *out_of_focus*, where *in_focus* means that the word is the first content word in the student's focus of attention.

In slice 1, each $Focus_1$ node is influenced by the tutor's action. For instance, if the tutor hints about a problem step or word, the corresponding node is likely to be *in_focus*. For RTDT, a tutor hint about the sentence as a whole increases the probability that the student will attempt to read

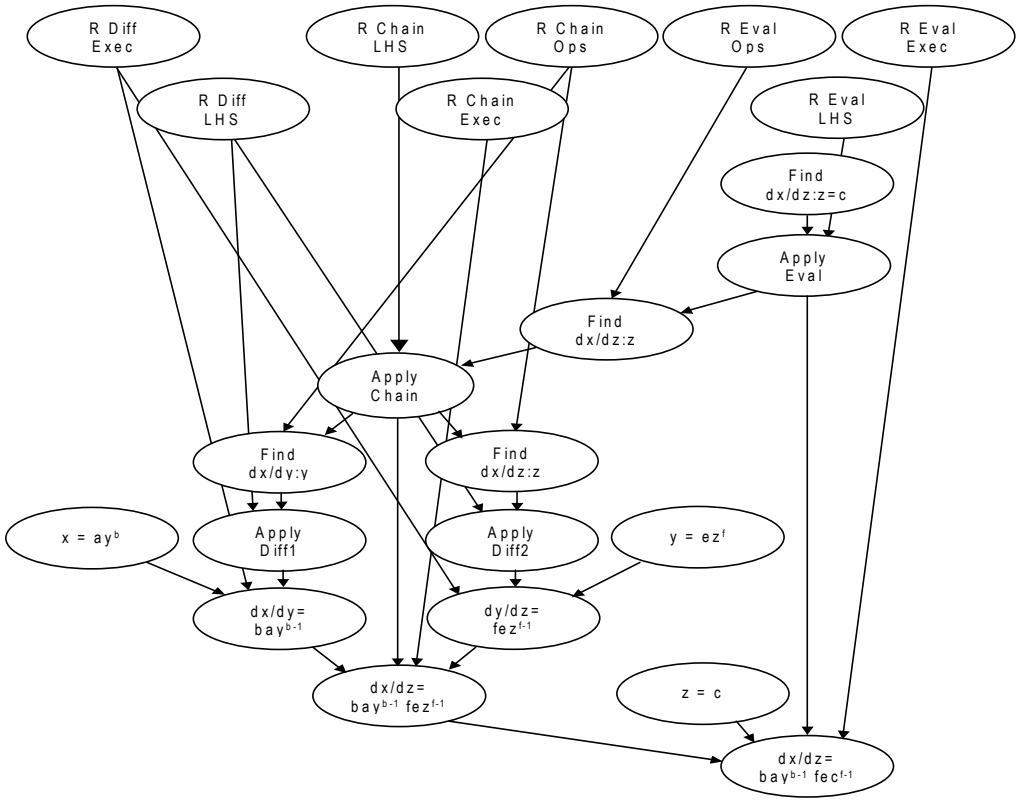

Figure 3. Problem solution graph for CTDT

the entire sentence (starting with the first word), increasing the probability that *Focus_Word_1*$_1$ is *in_focus*. In slice 2, the student action influences the tutor's beliefs about the student's focus of attention (in *Focus*$_2$). For instance, if the student experiences an impasse on a problem step or a word, the corresponding node is more likely to be *in_focus*.

4.1.4 *Student Action*$_2$ Nodes

These nodes represent one or more actions taken on the student's turn. For CTDT, a single student action is assumed. This action is represented by two nodes, one for the action *topic* and another for the action *type*. The action *topic* may be any problem step and the action *type* may be *correct*, *error*, *impasse*, or *null* (no student action).

For RTDT, the student turn may include multiple reading actions, where each action is an attempt to read a word. Student action *Word_i*$_2$ nodes represent the student's reading of each content word *i* as *not_read*, *error*, or *correct*. This representation models student turns ranging from no productive attempt (all words *not_read* – e.g., a silent impasse), to all words read correctly (all words *correct*), to any combination of words *not_read*, read in *error*, and read *correct*ly. In addition, a student action *Sentence*$_2$ node models the student's reading of the sentence as a whole as either *fluent* or *disfluent*.

Both CTDT and RTDT probabilistically predict the next student action. For CTDT, *Focus*$_1$ influences the student action *topic*. Given the action *topic*, whether the action *type* will be *correct*, *error* or *impasse* depends on the student's knowledge. Therefore, both the student action *topic* and *Knowledge*$_1$ influence the student action *type*.

For RTDT, influences on each *Word_i*$_2$ node from the corresponding *Focus_Word_i*$_1$ node probabilistically predict which word the student will attempt first. For any word that the student at-

tempts, an influence from the corresponding $Know\_Word\_i_1$ node predicts whether the reading will be in *error* or *correct*. We assume that if a student reads one word correctly, she is most likely to attempt the next word, and so on, until she gets stuck or makes an error. Therefore, arcs from each node $Word\_i_2$ to node $Word\_i+1_2$, $i = \{1, 2, ..., n-1\}$, model the influence of reading word i correctly on the likelihood that the student will attempt word $i+1$. For a *fluent* reading of the sentence, each word must be *correct* without pauses in between – i.e., the student must be able to read each word and the sentence as a whole. The $Sentence_2$ node is therefore influenced by each $Word\_i_2$ node and by the $Know\_Sentence_1$ node.

4.1.5 *Discourse State*$_s$ Sub-Network

For CTDT, a *Coherence* node represents the coherence of the tutor's action in response to the previous student action as either *coherent* or *incoherent*. For instance, *negative feedback* in response to a *correct* student action is *incoherent*. A *Relevance* node, with values *high* and *low*, models how well the tutor cooperates with the student's focus of attention by assessing the extent to which the same problem steps are *in_focus* before and after the tutor's action: Problem steps that are in the student's focus of attention are likely to be *in_focus* in $Focus_0$. A tutorial action which addresses a problem step or related rule that is in the student's focus of attention will further increase the probability that the problem step is *in_focus* in $Focus_1$. Therefore, if the same problem steps are most likely *in_focus* in $Focus_0$ and $Focus_1$, *Relevance* is most likely *high*.

For RTDT, *Discourse State*$_s$ is simply the number of discourse turns, counted as a measure of success at avoiding spending too much time on a sentence.

4.1.6 $Utility_2$ Nodes

$Utility_2$ consists of several utility nodes in a structured utility model representing tutor preferences regarding tutorial state outcomes. Total utility is a weighted sum of the utilities for each tutorial state component (e.g., student knowledge, focus, and affect; task progress; discourse state). The utility value for each component may in turn be a weighted sum of the utilities for each subcomponent. For instance, $Knowledge_2$ rules that are important to the curriculum may be weighted more heavily than certain problem steps.

The tutor's behavior can easily be modified by changing the utilities or their weights. For instance, it may be that the best way for the tutor to improve the student's domain knowledge is to focus on the student's knowledge at the expense of helping the student make progress on tutorial tasks (e.g., solving problems). The tutor will do this automatically if a high weight is assigned to the utility of student knowledge and a low weight is assigned to the utility of task progress.

4.2 Implementation

With input from a problem solution graph (CTDT) or text (RTDT), DT Tutor creates a TACN with default values for prior and conditional probabilities and utilities. Default values are specified by parameter for easy modification. An optional file specifies any prior probability or utility values that differ from the defaults. After creating the initial TACN, DT Tutor recommends tutorial actions, accepts inputs representing tutor and student actions, updates the network, and adds new TACNs to the DDN as appropriate.

We automated construction of the large number of conditional probability table entries using a much smaller number of rules and parameters. For instance, for RTDT, the rule for the probability that a student will remember in slice 2 a word that she knew in slice 1 is:

$$P(Know\_Word\_i_2 = known \mid Know\_Word\_i_1 = known) = 1.0 - word\text{-}forget\text{-}probability$$

word-forget-probability is a parameter that specifies the probability that the student will forget a

known word between slices.

Both of DT Tutor's applications are prototypes for testing the viability and generality of the approach. CTDT does not yet have an interface, and RTDT has not been integrated with the Reading Tutor. Therefore, we used simulated student input for formative evaluations.

5 Formative Evaluation

Our goal was to determine whether DT Tutor's prototype applications can select optimal actions quickly enough to keep a student engaged.

5.1 Response Time

One of the major challenges facing probabilistic systems for real-world domains is tractability. We performed response time testing on a 667-MHz Pentium III PC with 128-MB of RAM. Using Cooper's (1988) algorithm for decision network inference using belief network algorithms, we tested with three algorithms: an exact clustering algorithm (Huang & Darwiche, 1996) and two approximate, sampling algorithms, likelihood sampling (Shachter & Peot, 1989) and heuristic importance (Shachter & Peot, 1989), with 1,000 samples each. Response times reported are the mean over 10 trials. The times for the approximate algorithms were extremely close, with neither holding an advantage in all cases, so they are reported as one below.

For CTDT, only the approximate algorithms had reasonable response times for both problems tested: 1.5 seconds for a 5-step problem and 2.1 seconds for an 11-step problem.

For the Reading Tutor's corpus of readings, sentence length ranges from approximately 5 to 20 words as reading level progresses from kindergarten through fifth grade, with approximately two-thirds content words, so we tested response times for preemptive help on sentences with 2 to 14 content words. Our response time goal was 0.5 seconds or less. For all three algorithms, response times for sentences with up to 7 content words were less than 0.5 seconds, ranging from 0.04 seconds for 2 content words to .49 seconds for 7 content words. Response times for the exact algorithm blew up starting at 10 content words with a time of 12.48 seconds. Response times for the approximate algorithms remained promising (as explained below) for up to 12 content words, ranging from .59 seconds for 8 content words to 3.14 seconds for 12 content words. However, response times for even the approximate algorithms blew up at 13 content words with times of 23-26 seconds. Therefore, response time for preemptive help was satisfactory for students at lower reading levels, did not meet the goal for longer sentences (starting at 8 content words), and was entirely unsatisfactory even with the approximate algorithms for the longest sentences (13-14 content words). Response time would tend to increase if the number of tutor action types is increased (see section 4.1.1), although the amount of increase would be at most linear in the proportion of additional action alternatives considered.

For decision-making purposes, it is sufficient to correctly rank the optimal alternative. When only the rank of the optimal alternative was considered, the approximate algorithms were correct on every trial. While this result cannot be guaranteed, it may make little practical difference if the alternative selected has an expected utility that is close to the maximum value. Moreover, many sampling algorithms have an *anytime* property that allows an approximate result to be obtained at any point in the computation (Cousins et al., 1993), so accuracy can continue to improve until a response is needed. For RTDT, response times for corrective feedback should generally be faster because RTDT does not consider helping with words that have already been read correctly. In any case, faster response times can be expected as computer hardware and probabilistic reasoning algorithms continue to improve. Therefore, the response times reported above for the approximate algorithms show promise that DT Tutor applications for real-world domains will be able to re-

spond accurately enough within satisfactory response time. To handle the more challenging cases (such as the longest sentences faced by RTDT) in the near-term, application-specific adjustments may be required – e.g., abstraction in the knowledge representation within TACN components.

5.2 Action Selections

DT Tutor's decision-theoretic representation guarantees that its decisions will be optimal given the belief structure and objectives that it embodies. Nevertheless, the first step in evaluating a tutoring system is to see if it behaves in a manner that is consistent with strong intuitions about the pedagogical value of tutorial actions in specific situations. Such a sanity check cannot of course be a complete test. The space of network structures and probability and utility values, in combination with all possible student actions, is infinite, so the most we can do is sample from this space. However, if DT Tutor can handle many situations in which our intuitions are strong, we are more apt to have faith in its advice in situations where intuitions are less clear, and this is a prerequisite for testing with human subjects. Therefore, we tested DT Tutor's behavior in clear-cut situations.

First, we used default parameters to initialize TACNs with intuitively plausible probability and utility values. Next, we simulated student action inputs while perturbing probability and utility values to probe dimensions of the situation space. For instance, to test whether CTDT and RTDT would give preemptive help when warranted, we simply perturbed the prior probabilities for student knowledge of one or more domain elements (e.g., problem steps or words) to be most likely *unknown* and then verified that the application would suggest appropriate preemptive help.

The tests showed that DT Tutor is capable of selecting tutorial actions that correspond in interesting ways to the behavior of human tutors. Notable action selection characteristics include the following:

- Preemptively intervenes to prevent student errors and impasses, as human tutors often do (Lepper et al., 1993).
- Does not provide help when the student does not appear to need it. Human tutors often foster their students' independence by letting them work autonomously (Lepper et al., 1993).
- Adapts tutorial topics as the student moves around the task space and predicts the influence of the tutor's actions on the student's focus of attention.
- With equal utilities for knowledge of rules and steps, CTDT tends to address the student's knowledge of rules rather than problem-specific steps (because rule knowledge helps the student complete steps on her own). Effective human tutoring is correlated with teaching generalizations that go beyond the immediate problem-solving context (VanLehn et al., in press).
- CTDT tempers its actions based on consideration of the student's affective state (e.g., avoiding *negative feedback*). Human tutors consider the student's affect as well (Lepper et al., 1993).
- RTDT avoids repeating ineffective tutorial actions.

6 Related Work

Very few tutoring systems have used decision theory. Reye (1995) proposed a decision-theoretic approach for tutoring systems, mentioning an implementation in progress for tutoring SQL. Reye (1996) also proposed modeling the student's knowledge using a dynamic belief network. CAPIT (Mayo & Mitrovic, 2001, to appear), a decision-theoretic tutor for capitalization and punctuation, bases its decisions on a single objective and ignores the student's internal state in order to focus on observable variables. DT Tutor is a domain-independent architecture which considers multiple objectives, including objectives related to a rich model of the student's internal state.

Tutoring is a type of practical, mixed-initiative interaction. Within this broader domain, systems

by Horvitz and colleagues (e.g., Horvitz et al., 1998; Horvitz & Paek, 1999) also model the state of the interaction, including the user's state, with connected sets of Bayesian models, and employ decision theory for optimal action selection. Some of these systems (e.g., Horvitz & Paek, 1999) use value-of-information to guide user queries and observation selection, which DT Tutor does not (yet) do. To model temporal evolution, a number of probabilistic approaches have been tried, including dynamic and single-stage network representations (e.g., Horvitz et al., 1998). DT Tutor appears to be alone among systems for mixed-initiative interaction in (1) using a dynamic decision network to consider uncertainty, objectives, and the changing state within a unified paradigm, and (2) explicitly predicting the student's next action and its effect on the interaction.

7 Future Work and Discussion

We are currently selecting the domain for the first full-fledged implementation of DT Tutor's action selection engine in a complete tutoring system, either by combining it with an existing tutoring system (such as the Reading Tutor) or by building our own user interface. We are also investigating applications that are more explicitly dialogue-oriented. Whichever domain we select, our next major milestone will be testing the effectiveness of DT Tutor's approach with students.

Efficiently obtaining more accurate probability and utility values is a priority. However, precise numbers may not always be necessary. For instance, diagnosis (say, of the student's knowledge) in Bayesian systems is often surprisingly insensitive to imprecision in specification of probabilities (Henrion et al., 1996). For a decision system, it is sufficient to correctly rank the optimal decision alternative. Moreover, if the actual expected utilities of two or more alternatives are very close, it may make little practical difference which one is selected.

This work has shown that a decision-theoretic approach can be used to select tutorial discourse actions that are optimal, given the tutor's beliefs and objectives. DT Tutor's architecture balances tradeoffs among multiple competing objectives and handles uncertainty about the changing tutorial state in a theoretically rigorous manner. Discourse actions are selected both for their direct effects on the tutorial state, including the student's internal state, and their indirect effects on the subsequent student turn and the resulting tutorial state. The tutorial state representation may include any number of attributes at various levels of detail, including the discourse state, task progress, and the student's knowledge, focus of attention, and affective state. A rich model of the tutorial state helps DT Tutor to select actions that correspond in interesting ways to the behavior of human tutors. Response time remains a challenge, but testing with approximate algorithms shows promise that applications for diverse real-world domains will be able to respond with satisfactory accuracy and speed.

As an action-selection engine, DT Tutor plays at most the role of a high-level discourse planner, leaving the specifics of dialogue understanding and generation (parsing, semantic interpretation, surface realization, etc.) to other components of the tutoring application. It performs near-term discourse planning by anticipating the effects of its actions on the student's internal state, the student's subsequent discourse turn, and the resulting tutorial state. To predict how its actions will influence the tutorial state, including the student's internal state, DT Tutor's architecture includes strong domain reasoning and student modeling.

Acknowledgments

This research was sponsored by the Cognitive Science Division of the Office of Naval Research under grant N00014-98-1-0467. For decision-theoretic inference, we used the SMILE reasoning engine contributed to the community by the Decision Systems Laboratory at University of Pittsburgh (http://www.sis.pitt.edu/~dsl). We thank the reviewers for several helpful suggestions.

References

Cooper, G. F. (1988). A method for using belief networks as influence diagrams. In *Workshop on Uncertainty in Artificial Intelligence*, pp. 55-63.

Cousins, S. B., Chen, W., & Frisse, M. E. (1993). A tutorial introduction to stochastic simulation algorithms for belief networks. *Artificial Intelligence in Medicine 5*, pp. 315-340.

Henrion, M., Pradhan, M., Del Favero, B., Huang, K., Provan, G., & O'Rorke, P. (1996). Why is diagnosis in belief networks insensitive to imprecision in probabilities? In *12th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 307-314.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., & Rommelse, K. (1998). The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *14th Conference on Uncertainty in Artificial Intelligence*, pp. 256-265.

Horvitz, E., & Paek, T. (1999). A computational architecture for conversation. In *Seventh International Conference on User Modeling*, pp. 201-210.

Huang, C., & Darwiche, A. (1996). Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning 15*, 225-263.

Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S., & Weber, J. (1994). Automated symbolic traffic scene analysis using belief networks. In *12th National Conference on Artificial Intelligence*, pp. 966-972.

Lepper, M. R., Woolverton, M., Mumme, D. L., & Gurtner, J.-L. (1993). Motivational techniques of expert human tutors: Lessons for the design of computer-based tutors. In S. P. Lajoie & S. J. Derry (Eds.), *Computers as Cognitive Tools*, pp. 75-105. Lawrence Erlbaum Associates.

Mayo, M., & Mitrovic, A. (2001, to appear). Optimising ITS behaviour with Bayesian networks and decision theory. *International Journal of Artificial Intelligence in Education 12*.

Merrill, D. C., Reiser, B. J., Ranney, M., & Trafton, J. G. (1992). Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *The Journal of the Learning Sciences 2*(*3*), 277-306.

Mostow, J., & Aist, G. (1999). Giving help and praise in a reading tutor with imperfect listening -- because automated speech recognition means never being able to say you're certain. *CALICO Journal 16*(*3*), 407-424.

Mostow, J., & Aist, G. (in press). Evaluating tutors that listen: An overview of Project LISTEN. In K. Forbus & P. Feltovich (Eds.), *Smart Machines in Education: The coming revolution in educational technology*. MIT/AAAI Press.

Murray, R. C., & VanLehn, K. (2000). DT Tutor: A dynamic, decision-theoretic approach for optimal selection of tutorial actions. In *Intelligent Tutoring Systems, 5th International Conference,* pp. 153-162.

Newell, A., & Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall, Inc.

Reye, J. (1995). A goal-centred architecture for intelligent tutoring systems. In J. Greer (Ed.) *World Conference on Artificial Intelligence in Education*, pp. 307-314.

Reye, J. (1996). A belief net backbone for student modeling. In *Intelligent Tutoring Systems, Third International Conference*, pp. 596-604.

Shachter, R., & Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks. In *5th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 221-231.

Singley, M. K. (1990). The reification of goal structures in a calculus tutor: Effects on problem solving performance. *Interactive Learning Environments 1*, 102-123.

VanLehn, K., Siler, S., Murray, C., Yamauchi, T., & Baggett, W. B. (in press). Human tutoring: Why do only some events cause learning? *Cognition and Instruction*.