Looking Ahead to Select Tutorial Actions: A Decision-Theoretic Approach

R. Charles Murray, Kurt Vanlehn, Jack Mostow

Looking Ahead to Select Tutorial Actions: A Decision-Theoretic Approach

R. Charles Murray, *Intelligent Systems Program & LRDC, 3939 O'Hara Street, University of Pittsburgh, Pittsburgh, PA, 15260, USA.*
rmurray@pitt.edu
Kurt VanLehn, *Computer Science Department & LRDC, 3939 O'Hara Street, University of Pittsburgh, Pittsburgh, PA, 15260, USA.*
vanlehn@cs.pitt.edu
Jack Mostow, *Robotics Institute & Project LISTEN, Carnegie Mellon University, RI-NSH 4213, 5000 Forbes Avenue, Pittsburgh, PA,15213, USA.*
mostow@cs.cmu.edu

Abstract. We propose and evaluate a decision-theoretic approach for selecting tutorial actions by looking ahead to anticipate their effects on the student and other aspects of the tutorial state. The approach uses a dynamic decision network to consider the tutor's uncertain beliefs and objectives in adapting to and managing the changing tutorial state. Prototype action selection engines for diverse domains - calculus and elementary reading - illustrate the approach. These applications employ a rich model of the tutorial state, including attributes such as the student's knowledge, focus of attention, affective state, and next action(s), along with task progress and the discourse state. For this study, neither of our action selection engines had been integrated into a complete ITS, so we used simulated students to evaluate their capabilities to select rational tutorial actions that emulate the behaviors of human tutors. We also evaluated their capability to select tutorial actions quickly enough for real-world tutoring applications.

INTRODUCTION

This paper proposes and evaluates a decision-theoretic approach for selecting tutorial actions while helping a student with a task such as solving a problem. The approach, which we call *DT Tutor*, involves explicitly looking ahead to anticipate how the tutorial action alternatives will influence the student and other aspects of the tutorial state. For each tutorial action alternative, the tutor computes (1) the probability of every possible outcome of that tutorial action, (2) the utility of each possible outcome in relation to the tutor's objectives, and then (3) the alternative's expected utility by weighting the utility of each possible outcome by the probability that it will occur. The tutor then selects the action with maximum expected utility. This approach unifies considerations regarding the tutor's objectives, the tutor's uncertain beliefs about the student's changing internal state (e.g., the student's knowledge), and the tutor's uncertain beliefs about the effects of tutorial actions. One advantage of a decision-theoretic approach is the capability to

balance multiple tutorial objectives in computing the utility of each outcome. DT Tutor leverages this capability by considering multiple objectives related to tutorial state outcomes such as the student's knowledge, focus of attention, affective state, and next action, along with task progress and the discourse state.

There are at least three ways that this approach could fail. First is the *knowledge representation issue*: For real-world tutorial contexts, it might not be feasible to decision-theoretically represent the tutorial state with enough fidelity. The real tutorial state is of course hopelessly complex. It includes the student's knowledge (which is changing, we hope), the student's focus of attention, the student's affect, progress on the tutorial task, the tutor's domain knowledge and pedagogical objectives, the discourse history, etc. We can approximate some of these tutorial state attributes. For instance, we could represent student affect as a variable with just two values: *high* or *low*. However, such coarse approximations may make it impossible to predict future tutorial states and rate their utilities accurately enough. Thus, one challenge is to develop a representation of the tutorial state that is accurate enough to make good decisions but not so complex that it is computationally infeasible.

Another way that this approach could fail is in its *tutorial action selection capabilities*. DT Tutor's architecture is novel. Its networks can be complex because they model multiple outcomes as they change over time, and they may include hundreds of nodes and thousands of probabilities and utilities. Yet these networks are just approximations of the tutorial state. DT Tutor's basis in decision-theory gave us good reason to believe that it should be able to select tutorial actions rationally. However, we did not know just what action selection capabilities would emerge from such a complex yet approximate representation, or whether they would be comparable to the capabilities of tutors using heuristics refined through years of experience or human intuition. Furthermore, we did not know how sensitive DT Tutor would be to changes in its probabilities and utilities.

A third potential point of failure is the *real-time inference requirement*: A tutor must select actions quickly enough to keep the student engaged. Our decision-theoretic approach uses an extension of Bayesian networks. DT Tutor's networks have many uninstantiated variables, are multiply-connected, and can be large. Such characteristics, which appear to be necessary for many complex, real-world domains (Cooper, 1990), can make probabilistic inference NP-hard (Cooper, 1990; Dagum & Luby, 1993) and thus can make real-time inference challenging.

We investigated the feasibility of our approach with tutorial action selection engines for diverse domains: calculus related rates problems and reading aloud. At the time of this study, neither of our action selection engines had a modern user interface, so we could not use them to test DT Tutor's effectiveness with students. However, we did use them to address the potential points of failure listed above. We addressed the knowledge representation issue by encoding the tutorial action selection problem for the two domains. We tested DT Tutor's action selection capabilities by presenting both action selection engines with a variety of scenarios and checking to see whether the actions selected were both (1) rational in light of the system's probabilities and utilities, and (2) comparable to the actions of human or other automated tutors. To see whether the real-time inference requirement could be met, we tested the response times of both action selection engines for a variety of problem sizes.

While many ITSs and other user modeling systems have used Bayesian networks for reasoning under uncertainty (see, e.g., Jameson, 1996), decision-theoretic approaches for selecting tutorial actions remain rare. Reye (1995) was perhaps the first to propose a decision-

theoretic approach for ITSs but left many details unspecified. Murray and VanLehn (2000) presented DT Tutor's architecture in the context of the action selection engine for calculus related rates problems. We know of two other decision-theoretic approaches for ITSs, both of which differ significantly from our work: (1) *iTutor* (e.g., Pek, 2003; Pek & Poh, 2000) uses a decision-theoretic approach to reason about curriculum topics such as which problems to present to the student, but uses heuristics to decide how to help the student with individual problems. (2) CAPIT (Mayo & Mitrovic, 2001) models the tutorial state in terms of observable constraints only. This approach eases the burden of learning empirically justifiable probabilities but sacrifices the capability to reason about unobservable tutorial state attributes, including the student's knowledge, which we believe is an important attribute for an ITS to consider.

Decision-theoretic methods have also been employed in the broader user modeling community - for instance, in the work of Horvitz (e.g., Horvitz et al., 1998) and Jameson (e.g., Jameson et al., 2001). Conati's architecture (e.g., 2002) for educational games is probably the closest to DT Tutor in that it too employs a dynamic decision network representation to consider both the user's knowledge and affective state to select actions while helping the user with a task, although so far it focuses only on modeling the user's affective state. We provide a more complete review in the Relation to Prior Work section.

Below, we first describe the decision-theoretic basis of our approach. Next, we present our solution to the knowledge representation issue by describing DT Tutor's general architecture and its realization as action selection engines for calculus related rates problems and reading aloud. Then we present the results of tests of DT Tutor's action selection capabilities and response time. Next, we describe related work in ITSs and other user modeling systems. We conclude with a discussion of our results and plans for future work.

A DECISION-THEORETIC APPROACH

The term *decision-theoretic* has been used in various ways (Jameson et al., 2001). We define our general approach in this section, reviewing prerequisite concepts while working up to a description of the dynamic decision network that is at the heart of DT Tutor.

Probability has long been the standard for modeling uncertainty in diverse scientific fields. In recent years, algorithms for *belief networks* (Pearl, 1988, equivalently, Bayesian networks) have made probabilistic modeling of complex domains more feasible. A belief network is a directed acyclic graph with (1) a chance node for each modeled attribute to represent beliefs about its value, and (2) arcs between nodes to represent conditional dependence relationships among the beliefs. Beliefs are specified in terms of probability distributions for the attribute's possible values. For a node with incoming arcs, a conditional probability table specifies its probability distribution conditioned on the possible values of its parents. For a node without parents, a prior probability table specifies its probability distribution prior to observation of actual node values. In many real-world scenarios, a substantial number of conditional independence relationships exist. When this is the case, a belief network can concisely represent the entire joint probability distribution - the probabilities for every possible combination of attribute values - with exponentially fewer probability entries, making it possible to model more complex domains. Belief networks provide a mathematically sound basis for updating beliefs about any set of nodes in the network given any set of observations. Prior and conditional beliefs

may be determined subjectively, theoretically, or empirically. Using Bayes' rule and a variety of inference algorithms, belief networks can be used to perform diagnostic, causal and intercausal reasoning, as well as any combination of these (Russell & Norvig, 1995).

Each node within a belief network represents possibly changing beliefs about an attribute whose value is fixed even though it may be unknown. *Temporal probabilistic networks* (Dean & Kanazawa, 1989) support reasoning under uncertainty in domains where the values of attributes may change over time (as tutorial state attributes often do). For each attribute whose value may change, a sequence of nodes represents the attribute's value at each point in time. Typically, a new *slice* is created for each time point at which attribute values may change, where a slice is a set of 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 (Reye, 1996; Reye, 2004). In addition to atemporal arcs between nodes within the same slice, temporal arcs extend between nodes across time slices to represent the fact that attribute values may also depend on earlier values of the same and other attributes. The set of temporal arcs represents the network's state evolution model (Russell & Norvig, 1995). Typically (e.g., Albrecht et al., 1998), each slice is constructed so that the Markov property holds true, by adding additional nodes if necessary (Russell & Norvig, 1995): attribute values in one slice depend only on attribute values in the same slice and in the immediately preceding slice.

In static temporal networks, the number of slices is fixed in advance. *Dynamic temporal networks* (e.g., *dynamic belief networks*) avoid this limitation by creating additional slices dynamically and removing old slices when they are no longer required. They rely on the Markov property to *roll up* beliefs from an old slice into the following slice so that beliefs in the following slice summarize all accumulated evidence and the old slice can be removed. However, attributes that are conditionally independent in one slice may eventually be influenced by a common historical cause, making them conditionally dependent in later slices. This can cause nodes in later slices to become fully connected (Boyen & Koller, 1998), eliminating the conciseness advantage of belief network representations. To avoid this situation, rollup schemes that approximate a slice's belief state without full connectivity are typically used (e.g., Boyen & Koller, 1998).

Decision theory extends probability theory to provide a normative account of how a rational decision-maker should behave (Keeney & Raiffa, 1976). The decision-maker's preferences in light of her objectives are quantified in terms of a numeric utility value for each possible outcome of the decision-maker's action. 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, thereby maximizing the utility achieved when averaged over all possible outcomes (Russell & Norvig, 1995). Explicitly quantifying the decision-maker's preferences facilitates comparing and prioritizing outcomes, helps to clarify the rationale underlying decisions (Jameson et al., 2001), and supports modifying the agent's behavior simply by changing utility values. The expected utility mechanism integrates considerations about probability and utility over a continuous range of values. Decision theory thus provides a rational, transparent, flexible and integrated mechanism for comparing decision alternatives in light of probabilities and priorities regarding any number of competing objectives. A belief network can be extended into a *decision network* (equivalently, an *influence diagram*) to model a decision-theoretic approach by adding *decision* and *utility* nodes along with appropriate

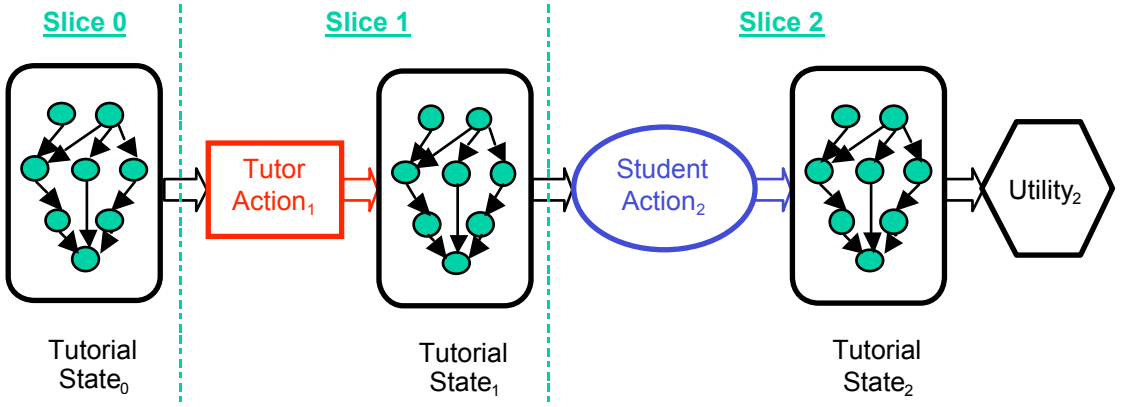


Fig. 1. Tutor Action Cycle Network (TACN), high-level overview

arcs (Howard & Matheson, 1984).

A *dynamic decision network* (DDN) combines the capabilities of a dynamic belief network and a decision network by combining chance, decision and utility nodes in a dynamic temporal representation (Dean & Wellman, 1991). DDNs model scenarios in which decisions, attribute values, or priorities among objectives can vary over time. They provide a unified mechanism for computing the decision with maximum expected utility considering both uncertainty about the changing state and multiple competing objectives. As with dynamic belief networks, DDNs are typically constructed so that they can rely on the Markov property to dynamically add new slices and remove old slices. Rollup methods are similar to those for dynamic belief networks.

DT Tutor uses a DDN to make tutorial action decisions by looking ahead to anticipate their effects on the changing tutorial state in light of the tutor's uncertain beliefs and multiple competing objectives. For DT Tutor, chance nodes represent the tutor's beliefs about tutorial state attributes, decision nodes represent tutorial action alternatives, and utility nodes represent the tutor's preferences among the possible tutorial states.

GENERAL ARCHITECTURE

DT Tutor's DDN is formed from dynamically created decision networks. We call each of these networks a *tutor action cycle network* (TACN) because they each represent a single cycle of tutorial action, where a cycle consists of deciding a tutorial action and carrying it out, observing the next student action, and updating the tutorial state based on these two actions.

Each TACN consists of three slices, as illustrated in Figure 1[1]. The *Tutorial State*$_s$ subnetwork in each slice is a set of chance nodes representing the student's state and all other

[1] In figures in this paper, decision nodes are represented by rectangles, chance nodes are represented by ovals, utility nodes are represented by hexagons, and subnetworks are represented by rounded rectangles. Each arc into or out of a subnetwork actually represents multiple arcs to and from various subnetwork nodes. For subnetwork and node names, a subscript of *0*, *1*, or *2* refers to the slice number. A subscript of *s* refers to any applicable slice.

attributes of the tutorial state, such as the task state and the discourse state. The *Tutor Action*$_1$ and *Student Action*$_2$ nodes represent tutor and student actions, respectively. The *Utility*$_2$ node is a high-level representation of multiple utility nodes that together represent the tutor's preference structure regarding the various possible outcomes of the tutor's action for the current TACN.

TACNs are used both for deciding the tutor's action and for updating the tutorial state. Let us first consider how a TACN is used for deciding the tutor's action. During this phase, slice 0 represents the tutor's current beliefs about the tutorial state, slice 1 represents the tutor's possible actions and predictions about their influence on the tutorial state, and slice 2 represents a prediction about the student's next action, its influence on the tutorial state, and the utility of the resulting tutorial state outcomes. The decision network inference algorithm calculates the action with maximum expected utility and the tutor selects that action. This ends the decision-making phase. The tutor executes the action. After the tutor has observed the student's action or decided that the student is at an impasse, the update phase begins.

The tutor enters the student action as evidence in slice 2 and updates the network. At this point, the posterior probabilities in *Tutorial State*$_2$ represent the tutor's current beliefs. Since it is now time for another tutorial action selection, another TACN is created and the dynamic network is rolled forward: posterior probabilities from *Tutorial State*$_2$ of TACN$_i$ are copied as prior probabilities to *Tutorial State*$_0$ of TACN$_{i+1}$, where they represent the tutor's current beliefs[2]. This initializes the new TACN. The old TACN is discarded. This ends the update phase. The tutor is ready to begin the next phase, deciding what action to take next.

With this architecture, the tutor both reacts to past student actions (e.g., for corrective feedback), whose effects are summarized by the beliefs in *Tutorial State*$_0$, and anticipates future student actions and their ramifications (e.g., to provide proactive help). In principle, the tutor can look ahead any number of slices without waiting to observe student actions in order to consider the long-term effects of its action alternatives. The tutor simply predicts probability distributions for the next student action and the resulting *Tutorial State*$_2$, rolls the DDN forward, predicts the tutor's next action and the following student action, and so on. However, a large amount of lookahead can be computationally prohibitive, so DT Tutor currently looks ahead only as far as the student's next action and the resulting tutorial state.

Tutorial State$_s$ Components

Figure 2 shows a generic TACN in slightly more detail. The *Tutorial State*$_s$ subnetwork in each slice is further divided into subnetworks to model various tutorial state attributes. In Figure 2, *Student Model*$_s$ is composed of the *Student Knowledge*$_s$ subnetworks to model the student's task-related knowledge, the *Student Focus*$_s$ subnetworks to model the student's task-related focus of attention, and the *Student Affect*$_s$ subnetworks to model the student's affective state. Outside the student model are the *Task Progress*$_s$ subnetworks to model the student and tutor's task-related progress, and the *Discourse State*$_s$ subnetworks to model the state of the discourse between student and tutor. The *Tutor Action*$_1$ and *Student Action*$_2$ representations, shown respectively as

[2] This is a naïve network rollup scheme which neglects additional dependencies between nodes in the new slice (slice 0 of TACN$_{i+1}$) that are induced by shared dependence on nodes in previous time slices. Future work includes refining this rollup using an algorithm for approximate summarization of past dependencies (e.g., Boyen & Koller, 1998).

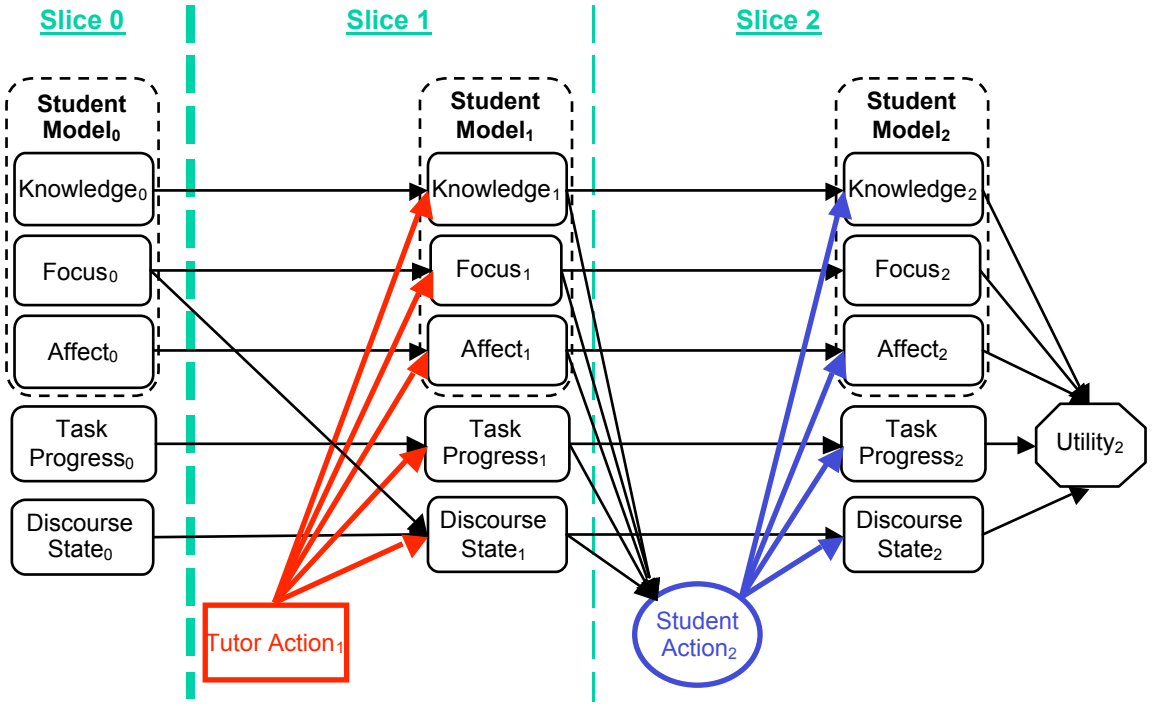


Fig. 2. TACN architecture in more detail

single decision and chance nodes, may actually consist of more than one node, depending on the application. Each of these components will be described in greater detail in the next section. This generic TACN is a framework in which a variety of components can be included or omitted, and each component can be modeled at various levels of richness and detail.

APPLICATIONS

We investigated the feasibility and generality of DT Tutor's knowledge representation schema by creating tutorial action selection engines for two domains: calculus related rates problems (Murray & VanLehn, 2000) and reading aloud (Murray et al., 2001). We present each application below, describing major components of their TACNs. We also describe the implementation process for both applications, including generating conditional probability table entries (which make up the vast majority of a TACN's numeric entries) using a much smaller number of rules with numeric parameters.

This section is crowded with details, but what makes these details significant is that they explicitly represent much of the common sense psychology and pedagogy that is usually hidden away within an ITS's heuristics. For instance, DT Tutor models relationships between the target domain's structure and the student's knowledge, focus of attention, and next action(s). Such relationships are hypotheses at this point, with the research literature still undecided. But it is interesting that DT Tutor can explicitly represent hypotheses of this kind. Indeed, it may provide

a vehicle for testing them. However, our claim at present is only that DT Tutor's architecture supports explicit representation of such hypothesized relationships.

Calculus Application

CTDT (Calculus Tutor, Decision-Theoretic) is an action selection engine for calculus related rates problems. A sample word problem for this domain (Singley, 1990, p.104) follows:

> *The economy of the newly-founded republic of San Pedro is growing such that, in any year* y*, the level* m *of the money supply in billion dollars is 2 times the square of the number of years elapsed. The gross national product* g *of the economy is 4 times the money supply. How fast is the gross national product growing when* y *equals 2 years?*

In equation form, the givens are $m = 2y^2$, $g = 4m$, and $y = 2$, and the goal is to find dg/dy when y equals 2. Singley (1990) developed a tutoring system for 32 problem types in this domain with an interface designed to make student problem-solving actions observable, including goal-setting actions that are normally unobservable.

Problem Solution Graph

A key knowledge representation scheme for CTDT is the *problem solution graph* that we use as the basis for modeling the relationship between the structure of problem solutions and the student's knowledge, focus of attention, next action, and task progress. A problem solution graph is a hierarchical dependency network (Huber et al., 1994) that includes all steps and related rules along any solution path for a problem. Like Andes1 (Conati et al., 2002), CTDT uses a rule-based problem solver to create a problem solution graph for each problem. Figure 3 presents an example for a simple problem of the same type as the example above except with generic variable names (x, y, z) and variables in place of constants. For ease of explication, only one solution path is shown. The goal is to find dx/dz when $z=c$ (node *Find dx/dz:z=c*) and the given equations are $x=ay^b$, $y=ez^f$, and $z=c$. *Rule* nodes are shown with diagonal shading in the two rows along the top. *Step* nodes are *facts* or *goals*: fact nodes are equations; the remaining nodes are goal nodes. Solidly shaded nodes are steps that have already been completed. The first subgoal is to apply evaluation (node *Apply Eval*) by substituting the equation $z=c$ into an equation for dx/dz in terms of z ($dx/dz:z$). Since no equation for $dx/dz:z$ exists, a subgoal is established to *Find dx/dz:z*. One way to find $dx/dz:z$ is to apply the chain rule (goal node *Apply Chain*) to equations for $dx/dy:?$ ("$:?$" means in terms of any variable) and $dy/dz:z$. These equations do not exist either, so subgoals are established to find them (*Find dx/dy:?* and *Find dy/dz:z*). One way to find $dy/dz:z$ is to apply differentiation (goal node *Apply Diff2*) to an equation for y in terms of z. One of the givens is $y=ez^f$, so differentiation is applied to derive the fact $dy/dz=fez^{f-1}$. A parallel process is used to derive the fact $dx/dy=bay^{b-1}$. The parallel steps to derive the latter two facts may be executed in any order. Now the chain rule can be applied to the equations $dx/dy=bay^{b-1}$ and $dy/dz=fez^{f-1}$ to derive fact $dx/dz=bay^{b-1}fez^{f-1}$, to which evaluation can be applied with fact $z=c$ to derive the answer, $dx/dz=bay^{b-1}fec^{f-1}$. Rule nodes license each problem step. For example, given the goal *Apply Eval* to $dx/dz:z=c$, a rule for finding evaluation operands (node *Eval Ops*) is used to establish the subgoal *Find dx/dz:z*. Once the fact $dx/dz=bay^{b-1}fez^{f-1}$ has been derived, a rule for

executing evaluation (*Eval Exec*) is combined with the goal *Apply Eval* and the operands $dx/dz=bay^{b-1}fez^{f-1}$ and $z=c$ to derive the answer.

Other solution paths are possible for this problem, many of which are inefficient. For instance, instead of differentiating $x=ay^b$ to get dx/dy, one could restate $x=ay^b$ in terms of y ($y = (x/a)^{1/b}$), differentiate the restated equation to get dy/dx, and then flip the derivative to get dx/dy. Furthermore, if the student is allowed to go down parts of multiple solution paths, the student could repetitively derive identical intermediate or final results in multiple ways. We are currently experimenting with our automated problem solver and student interface to decide which types of solution paths to allow.

Tutor Action$_1$ Nodes

CTDT addresses the tutor action *topic* in the manner specified by the tutor action *type*. These action components are represented by the decision nodes *Tutor Action Topic$_1$* and *Tutor Action Type$_1$*. The *Tutor Action Type$_1$* alternatives are currently *prompt*, *hint*, *teach*, *positive feedback*, *negative feedback*, *do* (tell the student exactly how to do a step), and *null* (no tutor action).

The *Tutor Action Topic$_1$* alternatives may consist of any problem step (fact or goal) or related rule in the problem solution graph. However, students rarely repeat steps that they have already completed successfully, and they are unlikely to be able to complete steps for which prerequisites have not been completed. Accordingly, tutors are less likely to address such steps. Therefore, for faster response time, CTDT normally considers as the tutor action topic only uncompleted steps for which prerequisites have been completed, any step that has just been completed (e.g., to give *positive feedback*), and related rules. A tutor action topic of *null* is also supported to model (1) a tutor action with a type but no specific topic, such as general *positive feedback*, or (2) no tutor action.

Student Action$_2$ Nodes

Like *Tutor Action$_1$*, CTDT's student action representation consists of nodes to model the student's action *topic* and *type*: *Student Action Topic$_2$* and *Student Action Type$_2$* respectively. *Student Action Type$_2$* may be *correct*, *error*, *impasse*, or *null*. A *correct* action matches an action in the problem solution graph. An action type of *impasse* models either a help request on a specific topic (specified by the *Student Action Topic$_2$* value) or a general help request such as "*What should I do next?*" *Null* means no student action. All other student actions are of type *error*. *Student Action Topic$_2$* may be any step in the problem solution graph or *null* to model either no action at all or an action with no specific topic (such as a general help request).

Student Focus$_s$ Subnetworks

For CTDT, the *Student Focus$_s$* subnetworks represent the tutor's beliefs about two components of the tutorial state: (1) the student's focus of attention within the current problem, and (2) the student's task progress (CTDT has no separate *Task Progress$_s$* subnetworks). Figure 4 illustrates the *Student Focus$_s$* subnetworks for a simple problem with just five steps (facts or goals), the first of which (*Step 1$_s$*) is given, and three related rules. We model the student's focus of attention relative to the problem steps, so the *Student Focus$_s$* subnetworks consist of just the step nodes in

the problem solution graph, except for *Student Focus*$_1$, which also includes the rule nodes, as explained below.

Student Focus$_s$ step nodes have four possible values: *not_ready, ready, in_focus,* and *complete*. 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. Such steps have the value *not_ready*. The student is also unlikely to repeat problem-solving steps that have already been completed successfully. These steps have the value *complete*. The remaining steps are uncompleted steps that the student could productively attempt next since all prerequisite steps have been completed, and thus are more likely to be in the student's focus of attention. They have some distribution over the values *ready* and *in_focus*, with *ready* meaning that the student is ready to attempt the step next, and *in_focus* meaning that the step is also in the student's focus of attention. In Figure 4, the probability distribution between *ready* and *in_focus* is depicted by the density of the dots shading the nodes, with denser dots meaning that the node is more likely to be *in_focus*.

Nodes in slice 0 represent the tutor's prior beliefs about the tutorial state and are disconnected except for arcs to slice 1. In *Student Focus*$_0$ of the first TACN for a tutorial session, prior probabilities for the given steps (the problem goal and facts) are set to *complete* with probability *1.0*. Steps with uncompleted precedent steps are set to *not_ready* with probability *1.0*. Prior probabilities for the remaining steps are set to a distribution over the values *ready* and *in_focus*, with a probability mass of *1.0* for *in_focus* divided equally among these steps. For subsequent TACNs, prior probabilities for slice 0 are copied from posterior priorities in slice 2 of the previous TACN. Slice 0 in Figure 4 depicts a situation in which *Step 1*$_0$ was given, *Step 2*$_0$ and *Step 3*$_0$ have equal probabilities of being *in_focus*, and *Step 4*$_0$ and *Step 5*$_0$ are *not_ready*.

Student Focus$_1$ represents the influence of the tutor's action on the student's focus of attention. The tutor's action can influence the student's focus among problem steps directly - e.g., by a *hint* on a specific step -so the tutor action nodes influence *Student Focus*$_1$ step nodes. The tutor normally considers addressing only steps that are *ready* or *in_focus*, plus any step that was just completed (not given steps), so in Figure 4 there are arcs from the tutor action nodes to *Step 2*$_1$ and *Step 3*$_1$. The tutor can also indirectly influence the student's focus among problem steps by addressing a related rule. In Figure 4, *Rule A*$_1$ is the rule parent of both *Step 2*$_1$ and *Step 3*$_1$, so there are arcs from the tutor action nodes to *Rule A*$_1$. *Student Focus*$_1$ rule nodes have a distribution over the values *in_focus* and *out_of_focus* with the obvious meanings. *Student Focus*$_1$ rule nodes in turn influence the probability that their related step nodes are *in_focus*.

Student Focus$_1$ influences the *topic* of the student's next action, *Student Action Topic*$_2$, which may be any problem step. However, if every *Student Focus*$_1$ step node influenced *Student Action Topic*$_2$, the number of conditional probability table entries required could be prohibitive: Each *Student Focus*$_1$ step node has 4 possible values, and *Student Action Topic*$_2$ has *s+1* possible values, where *s* is the number of problem steps. If there are just 15 steps and if each step node in *Student Focus*$_1$ influenced *Student Action Topic*$_2$, the conditional probability table for *Student Action Topic*$_2$ would require $4^{15} * (15+1)$ entries, or over 17 billion. In order to limit the size of the conditional probability table, only the step nodes that are more likely to be in the student's focus of attention - those that are *ready* or *in_focus* - influence *Student Action Topic*$_2$. The remaining steps - those that are *not_ready* or *complete* - may still be the topic of the next student action; it is just that they are assumed to have a uniformly low probability of being the action topic. This requires dynamic specification of arcs to *Student Action Topic*$_2$ and dynamic creation

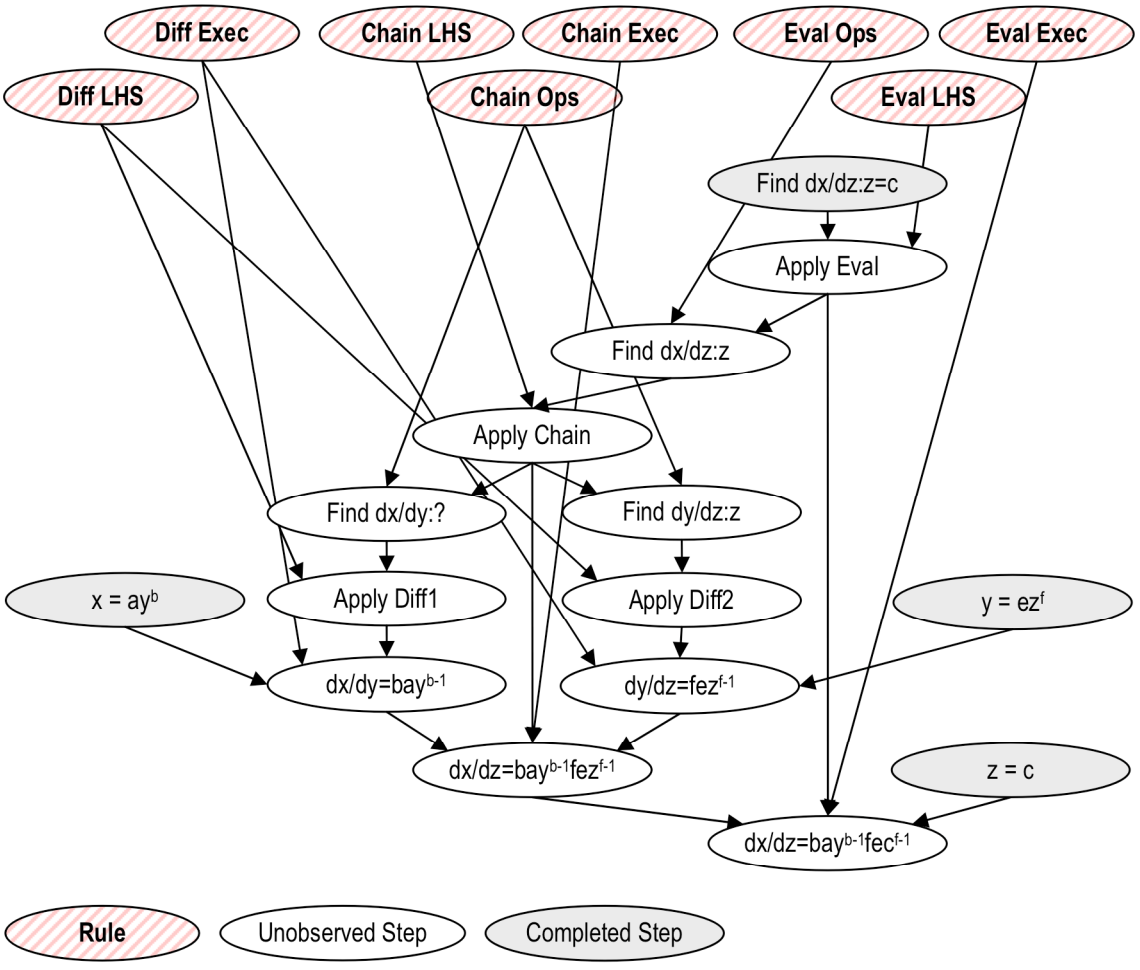


Fig. 3. Problem solution graph for CTDT

of *Student Action Topic*$_2$'s conditional probability table for each new TACN as student actions are observed. In the situation depicted in Figure 4, only *Step* 2_1 and *Step* 3_1 are *ready* or *in_focus* in slice 1, so only they influence *Student Action Topic*$_2$. This restriction results in considerable savings. For instance, if only 2 of 15 steps are *ready* or *in_focus*, only $4^2*(15+1) = 256$ conditional probabilities must be specified instead of over 17 billion.

The student action nodes can in turn influence the *Student Focus*$_2$ step nodes. In slice 2 of Figure 4, the student has just completed *Step 2*, so it is *complete*. *Student Focus*$_2$ step nodes are also influenced by their prerequisite steps. In Figure 4, when *Step* 2_2 becomes *complete*, its child, *Step* 4_2, has a distribution over the values *ready* and *in_focus* since all of its prerequisite steps (just *Step* 2_2) are now *complete*.

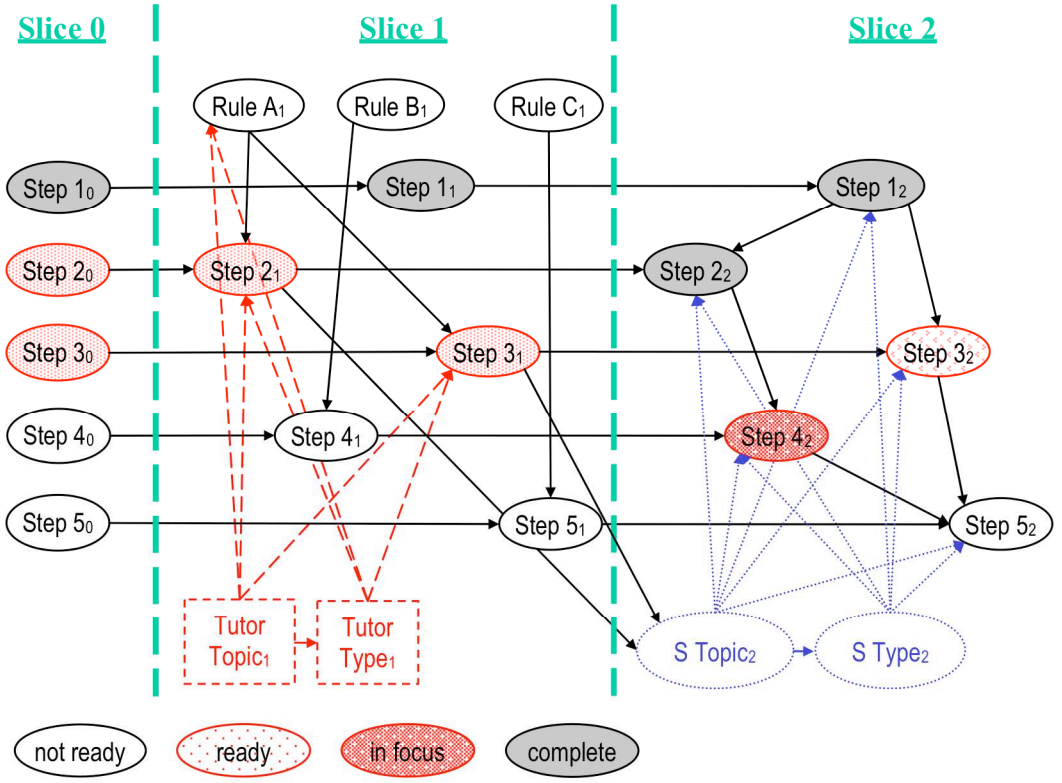


Fig. 4. *Student Focus*$_s$ subnetworks in CTDT's TACN

Focus Evolution and Aging

Temporal arcs between *Student Focus*$_s$ step nodes model the persistence of the student's focus of attention and task progress over time. For instance, steps that are *not_ready* remain so until either all of their parent steps are *complete* or the student completes the step (e.g., by guessing). In contrast, steps that are *in_focus* at some point in time become a little less likely to be *in_focus* with each passing slice. This is to model focus aging: steps that were *in_focus* slowly become less *in_focus* over time as the student moves on to other topics. In Figure 4, *Step 3*'s probability of being *in_focus* decreases from slice 1 to slice 2 as the student completes *Step 2*$_2$ instead.

When there are multiple steps that could be *in_focus* because they are the next *ready* step along some portion of a solution path, DT Tutor needs some way to decide how likely the various steps are to be *in_focus*. To do this, DT Tutor, like Andes1 (Gertner et al., 1998), assumes a depth-first bias: Students usually prefer to complete work on one portion of a solution path before starting to work on another. A depth-first bias in problem solving corresponds to a depth-first traversal of the problem solution graph. Such a bias is consistent with activation-based theories of human working memory (e.g., Anderson, 1993) and observations of human problem

solvers (e.g., Newell & Simon, 1972). However, depth-first bias is not absolute (VanLehn et al., 1989): at any given step, there is some probability that a student will not continue depth-first.

To model depth-first bias, when a step first becomes *ready* or *in_focus* because all of its parent steps have become *complete*, that step has a high probability of being *in_focus*. This is because the student, having just completed the last of the step's parents, is likely to continue working with the step itself. In Figure 4, *Step* 4_2 is highly likely to be *in_focus* since *Step* 2_2 has just been completed. Focus aging helps to model another aspect of depth-first bias: preferring to backtrack to more recently *in_focus* steps. When the student completes or abandons a portion of the solution path, steps that were recently *in_focus* but that are still not *complete* have had less focus aging than steps that were *in_focus* in the more distant past, so the more recently raised steps remain more likely to be *in_focus*.

Student Knowledge$_s$ Subnetworks

The *Student Knowledge$_s$* subnetworks represent the tutor's beliefs about the student's problem-related knowledge. Figure 5 provides an illustration for the same problem that was described in the previous subsection. To create these subnetworks, the problem solution graph is converted into a belief network, associating each node with a probability distribution for the values *known* and *unknown*. Rule nodes represent the tutor's belief about the student's knowledge of the corresponding rule. Step nodes represent the tutor's beliefs about the student's capability to derive the corresponding fact or goal given the student's rule knowledge. In Figure 5, the step nodes are shaded according to whether their *Student Focus*$_0$ subnetwork values are *not_ready*, *ready* or *in_focus* ("*ready/i-f*"), or *complete*. This shading is intended to illustrate why the tutor action nodes influence some nodes (the *ready* or *in_focus* nodes and their rule parents) and not others, as explained below.

In the first TACN for a tutorial session, prior probabilities for the *Student Knowledge*$_0$ rule nodes are based on the best information available, such as pretest data for a particular student or statistical data for a student population. Prior probabilities for the given steps (the problem goal and the given facts) are set to *known* with value *1.0*. Prior probabilities for the remaining steps are set to *unknown* with probability *1.0*. For subsequent TACNs, prior probabilities for slice 0 are copied from posterior priorities in slice 2 of the previous TACN.

Within slices 1 and 2, the *Student Knowledge$_s$* subnetworks have the same basic structure as the problem solution graph: atemporal arcs from rule nodes model the influence of rule knowledge on the student's ability to derive related steps, and atemporal arcs between step (fact or goal) nodes model prerequisite relations. Temporal arcs between corresponding nodes in adjacent slices model the persistence of the student's knowledge over time.

Student Knowledge$_1$ represents the influence of the tutor's action on the student's knowledge. The tutor normally considers addressing only steps that are *ready* or *in_focus* - this is the reason for the shading in Figure 5 - plus any step that was just completed. The tutor also considers tutoring on rules related to steps that are *ready* or *in_focus*, since (1) these rules are more likely to be in the student's focus of attention, and (2) tutoring on them may provide the knowledge necessary for the student to complete the corresponding step. Therefore, in Figure 5 there are arcs from the tutor action nodes to *Step* 2_1 and *Step* 3_1 (there is not an arc to *Step* 1_1 since it was given) and to *Rule* A_1 since it is the parent of both *Step* 2_1 and *Step* 3_1.

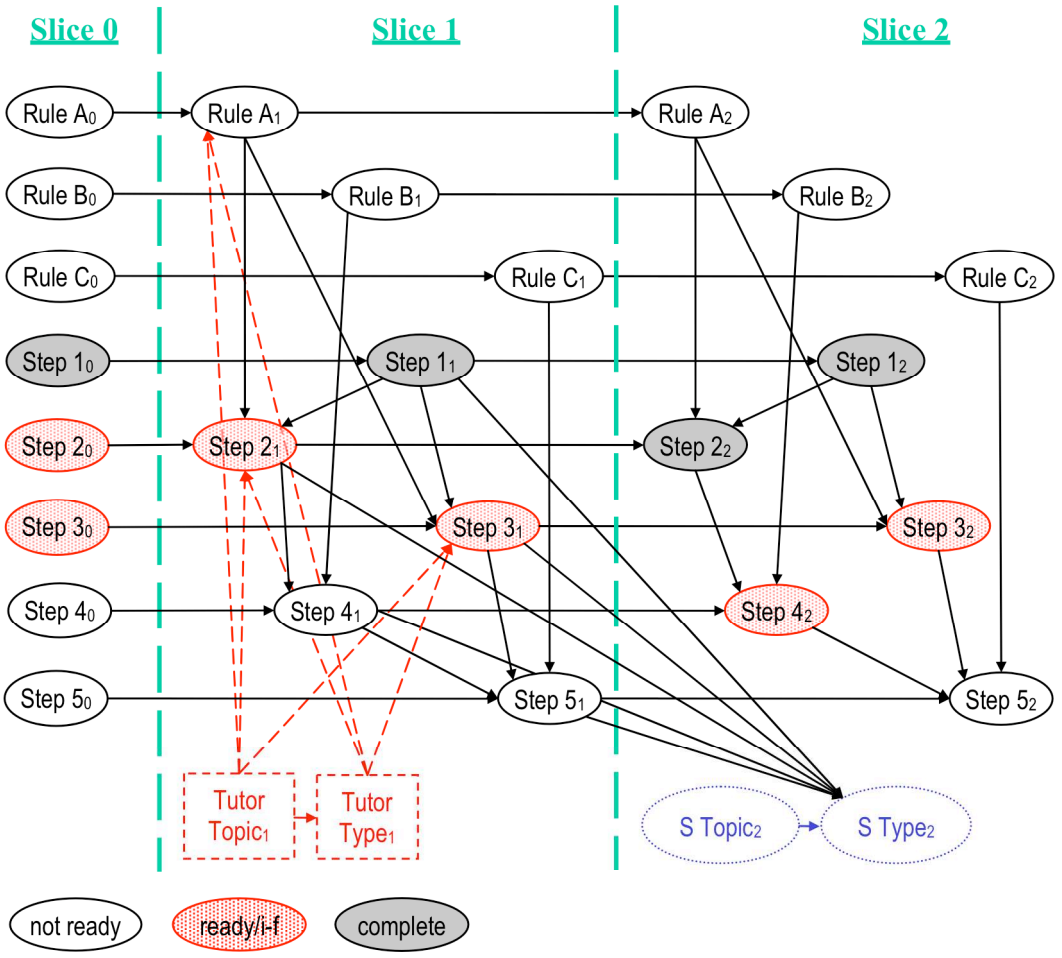


Fig. 5. Student Knowledge$_s$ subnetworks in CTDT's TACN

Given the topic of a student action (*Student Action Topic*$_2$), which may be any problem step (or *null*), the student's knowledge of the topic influences the student action type (e.g., *correct*, *error*, or *impasse*), *Student Action Type*$_2$. Therefore, *Student Knowledge*$_1$ step nodes influence *Student Action Type*$_2$. In Figure 5, all five step nodes in *Student Knowledge*$_1$ influence *Student Action Type*$_2$. Just as with the influence of *Student Focus*$_1$ step nodes on *Student Action Topic*$_2$, this could result in an untenably large number of influences on *Student Action Type*$_2$. However, in this case, since *Student Action Topic*$_2$ may be any step and we want to be able to learn diagnostically about the student's knowledge of the action topic based on her action type, we cannot restrict which *Student Knowledge*$_1$ step nodes influence *Student Action Type*$_2$. Instead, we employ what we call a *funnel subnetwork* (Murray, 1999, not shown in Figure 5) between the *Student Knowledge*$_1$ step nodes and *Student Action Type*$_2$. This funnel subnetwork does not change the semantics of the TACN in any way (which is why it is not shown in the figure); it is simply a factoring technique to minimize the number of conditional probability table entries.

In slice 2, the *Student Action*$_2$ nodes do not directly influence *Student Knowledge*$_2$ nodes. This is because a student action does not influence the student's knowledge without feedback (e.g., from the tutor), which is not modeled until the next TACN. Rather, once *Student Action*$_2$ has been observed, it influences *Student Knowledge*$_1$ nodes diagnostically, which in turn influence the corresponding *Student Knowledge*$_2$ nodes.

Student Affect$_s$ *Subnetworks*

The *Student Affect*$_s$ subnetworks represent the tutor's beliefs about the student's affective state. CTDT incorporates a simple model with just two attributes: (1) *Independence*$_s$, the student's feelings of independence or self-efficacy within the domain (e.g., whether she feels like she can solve problems without the tutor's help), and (2) *Morale*$_s$, the student's satisfaction with engaging in the current task. Independence is one of the attributes modeled in del Soldato and du Boulay's (1995) seminal work for affective modeling within ITSs, and is related to the attributes of *challenge* and *confidence* suggested by Lepper and colleagues (1993). Morale in CTDT is a coarse approximation of Lepper and colleagues' (1993) suggested attributes of *curiosity* and *control* and del Soldato and du Boulay's (1995) attribute of *effort*.

The *Independence*$_s$ and *Morale*$_s$ nodes each have five possible values, *level 0* through *level 4*, with higher levels representing greater independence or morale. Both the tutor and student actions influence the *Independence*$_s$ and *Morale*$_s$ nodes. For instance, a *Tutor Action Type*$_1$ value of *null* (no action) increases the *Independence*$_1$ value but leaves the *Morale*$_1$ value about the same. A *Student Action Type*$_2$ value of *correct* is likely to boost both *Independence*$_2$ and *Morale*$_2$, while a *Student Action Type*$_2$ value of *error* or *impasse* is likely to have the opposite effect. Arcs between corresponding *Independence*$_s$ and *Morale*$_s$ nodes in adjacent slices model the persistence of the student's affective state over time.

Utility$_2$ *Subnetwork*

Utility$_2$ is actually a number of utility nodes in a structured utility model representing tutor preferences regarding the following outcomes:

1. Student rule knowledge in slice 2 (rule nodes in *Student Knowledge*$_2$)
2. Student problem solving progress in slice 2 (step nodes in *Student Focus*$_2$)
3. Student independence in slice 2 (*Independence*$_2$)
4. Student morale in slice 2 (*Morale*$_2$)
5. Tutor action type in slice 1 (*Tutor Action Type*$_1$)
6. Discourse state coherence in slice 1 (*Coherence*$_1$)
7. Discourse state relevance in slice 1 (*Relevance*$_1$)

We use linearly-additive multi-attribute utility functions to combine subutilities for the outcomes above: Subutilities are combined by assigning a weight to each subutility, multiplying each subutility value by its weight, and summing the weighted subutility values. These functions make it easy to change DT Tutor's behavior by simply changing the weights. For instance, DT

Tutor will focus on student rule knowledge at the expense of problem-solving progress if a high weight is assigned to the former and a low weight is assigned to the latter.

Reading Application

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, 2001). 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, in contrast to CTDT, the Reading Tutor already has an extensively developed interface to which RTDT must adapt.

Currently, when the Reading Tutor gives help on an individual word, it selects randomly from a set of tutorial actions deemed to be felicitous for that word. We investigated the feasibility of applying DT Tutor to replace this random action selection mechanism for two types of unsolicited help: *proactive help* and *corrective feedback*. The Reading Tutor provides proactive help before the student attempts a sentence when it believes that she is likely to misread a word, and corrective feedback when it detects words read incorrectly, skipped words and disfluent reading. DT Tutor considers both proactive help and corrective feedback on every turn, so RTDT may provide proactive help even after the student's first attempt.

Tutoring reading differs enough from coaching calculus problem solving to pose challenges for adapting DT Tutor. First, student turns may consist of multiple reading actions, where a turn is defined to end when the student pauses for some threshold amount of time, and 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 it is important to meet this challenge.

Second, beginning readers often make repeated attempts at words or phrases and sometimes omit words, with the effect of jumping around within a sentence. Thus, the order in which beginning readers attempt words is not always sequential and has 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 constraints on the order in which actions may be completed.

Below, we describe some of the major components of RTDT's knowledge representation, highlighting differences with CTDT and the mechanisms by which we model the probabilistic relationship between the student's knowledge, focus of attention, and next actions. We also describe a new component, the *Tutor Efficacy*$_s$ subnetworks.

Network Structure

Figure 6 illustrates RTDT's general TACN structure. In contrast to the generic TACN structure in Figure 2: (1) *Student Model*$_s$ does not currently include *Student Affect*$_s$ subnetworks (although they could be added), (2) *Tutor Efficacy*$_s$ subnetworks are included, and (3) the *Student Action*$_2$

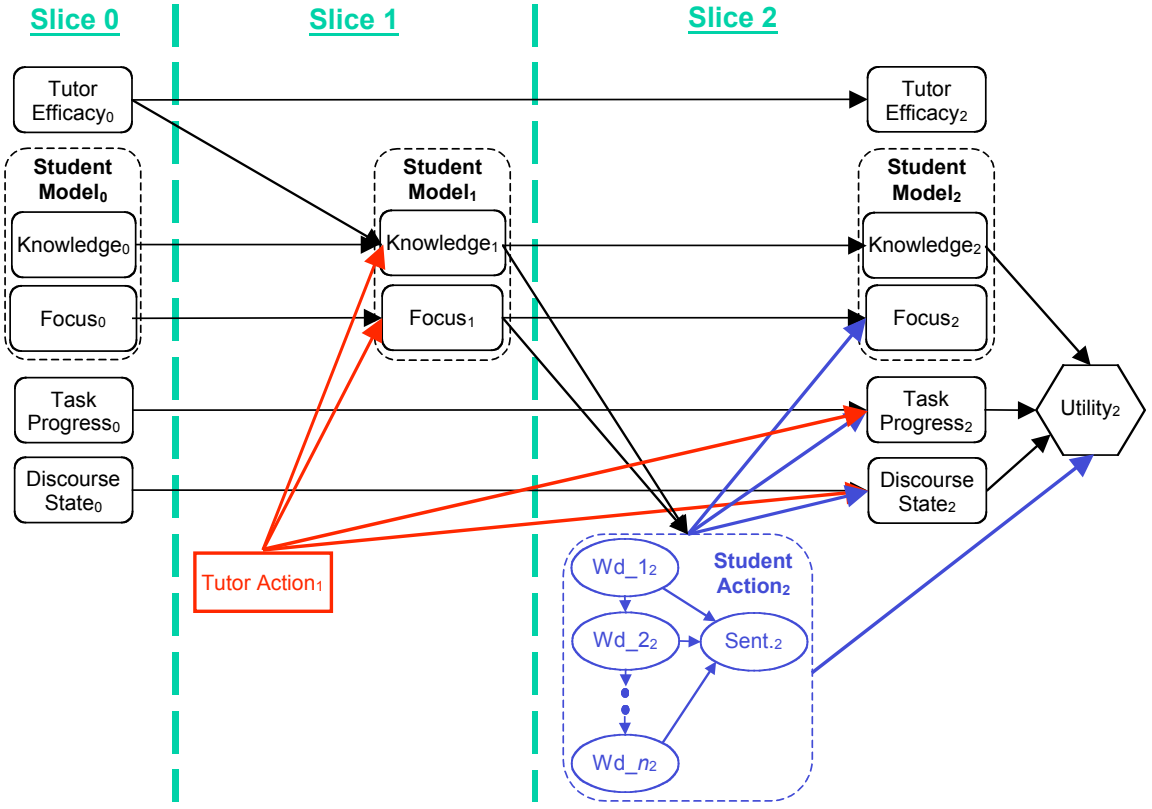


Fig. 6. RTDT's Tutor Action Cycle Network

representation has many more nodes. Also, not all TACN components are included in each slice - for instance, *Tutor Efficacy*$_s$ is not included in slice 1. For efficiency, we omit components where they are not necessary in order to reduce the size of the network. In contrast to CTDT, RTDT's *Task Progress*$_s$ subnetworks are not combined with the *Student Focus*$_s$ subnetworks.

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. Approximately two-thirds of the words in sentences within the Reading Tutor's corpus of readings are non-function words, or *content* words. RTDT ignores errors on function words as well and so does not include nodes to represent them in its subnetworks.

Tutor Action$_1$ *Node*

RTDT's tutor action representation consists of only one decision node, *Tutor Action*$_1$. The decision alternatives are *null* (do nothing), *move_on* (move on to the next sentence - e.g., after the student has completed a 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* (give a hint for word *i*) for each content word *i* in the current *n*-content-word sentence, *i = {1, 2, ..., n}*.

Specifying the type of hint as well - for example, whether to hint about a particular word by saying the word itself or by giving a rhyming hint - would require information that 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 probability that a rhyming hint would supply knowledge that the student needs.

Before the student's first attempt at a sentence, RTDT considers every action alternative, including hinting on each content word. For faster response time on subsequent attempts, RTDT does not consider hinting on words that the student has already read correctly, since hints on words that the student already knows are less likely to have pedagogical benefit.

Student Focus$_s$ Subnetworks

For RTDT, *Student Focus$_s$* includes a *Focus_Word_i$_s$* node for each content word *i* in the current sentence. Each node has possible values *in_focus* and *out_of_focus*, where *in_focus* means that the student intends to attempt to read this word next. In slice 1, each *Focus_Word_i$_1$* node is influenced by the tutor's action. For instance, if the tutor hints about word *j*, *Focus_Word_j$_1$* is more likely to be *in_focus*. A tutor hint about the sentence as a whole increases the probability that the student will attempt to read the entire sentence (starting with the first word), increasing the probability that *Focus_Word_1$_1$* is *in_focus*.

Student actions also influence the tutor's beliefs about the student's focus of attention. For instance, if the student misreads a word, she is more likely to focus on it (since she may know she misread it), so it is more likely to be *in_focus*. Similarly, if the student stops at a word (perhaps because she is having difficulty), it is more likely to be *in_focus*.

Student Knowledge$_s$ Subnetworks

For RTDT, the *Student Knowledge$_s$* subnetworks represent the student's knowledge of how to read the current sentence. For each content word *i*, a *Know_Word_i$_s$* node represents the student's knowledge of how to read the word. In addition, a *Know_Sentence$_s$* node represents the student's knowledge of how to read the sentence as a whole. Each of these nodes has possible values *known* and *unknown*.

The tutor's action influences the *Student Knowledge$_1$* nodes. For instance, *hint_word_j* increases the probability that *Know_Word_j$_1$* is *known*, and *hint_sentence* increases the probabilities that each *Know_Word_i$_1$* node and the *Know_Sentence$_1$* node are *known*. Knowing the sentence requires knowing each word, so *Know_Sentence$_1$* is also influenced by the student's knowledge of each content word (*Know_Word_i$_1$*). *Student Knowledge$_1$* in turn influences the success of the student's turn, as described in the next subsection. After the student's turn has been observed, *Student Knowledge$_1$* is updated diagnostically to reflect its causal role in the student's success at reading any words attempted.

Student Action$_2$ Nodes

The *Student Action$_2$* nodes model student turns which may consist of multiple reading actions, where each action is an attempt to read a word. Figure 7 illustrates the *Student Action$_2$* nodes in the context of predicting the next student turn. The student action *Read_Word_i$_2$* nodes represent

the student's reading of each content word *i* as *not_read*, *error*, or *correct*, based on results provided by the automated speech recognizer. 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, the student action $Read\_Sentence_2$ node models the student's reading of the sentence as a whole as either *fluent* or *disfluent*.

To predict the student's next turn, influences on each $Read\_Word\_i_2$ node from the corresponding $Focus\_Word\_i_1$ node probabilistically predict which word the student will attempt first. For any word that the student attempts, 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 $Read\_Word\_i_2$ to node $Read\_Word\_i+1_2$, $i = \{1, 2, ..., n-1\}$, model the influence of reading word *i* correctly on the probability that the student will attempt word *i+1*.

For a *fluent* reading of the sentence, each $Read\_Word\_i_2$ node must be *correct*, plus the sentence must be read without extraneous utterances or long pauses. The $Read\_Sentence_2$ node is therefore influenced by each $Read\_Word\_i_2$ node and by the $Know\_Sentence_1$ node.

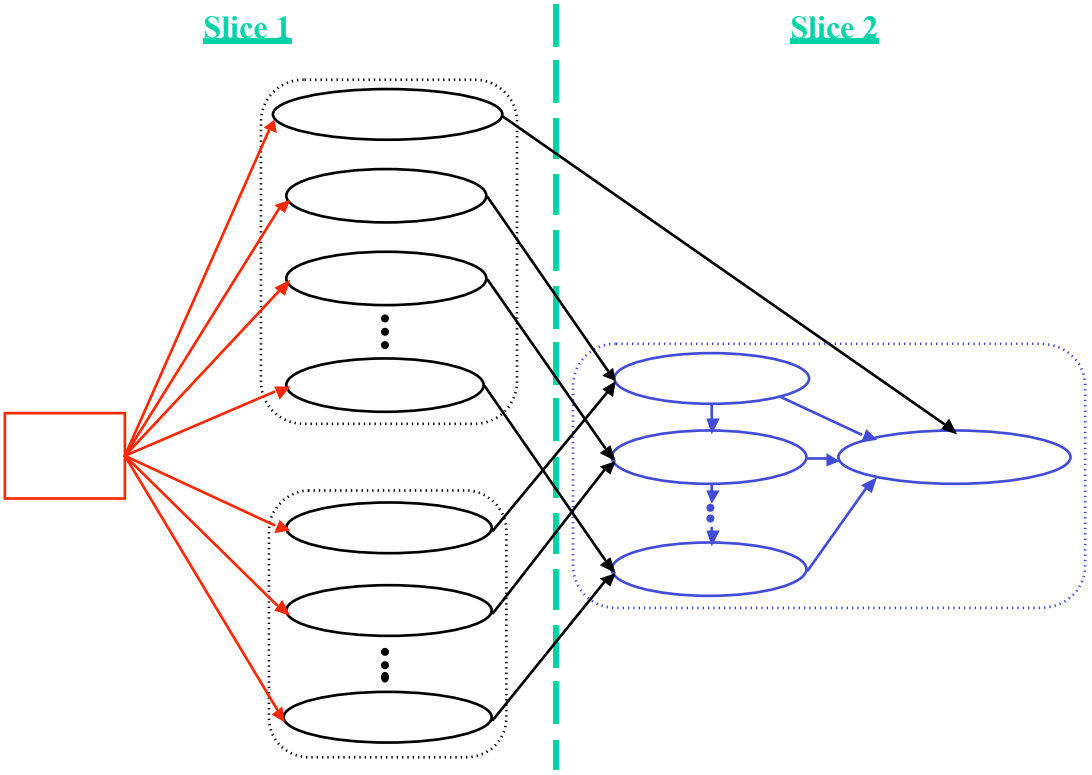
Tutor Efficacy$_s$ Subnetworks

The *Tutor Efficacy$_s$* subnetworks represent the tutor's beliefs about the efficacy of its help at increasing the student's knowledge. Each node represents the efficacy of a particular help alternative. RTDT represents the effectiveness of the *hint_sentence* and *hint_word_i* (for each word *i* in the sentence) alternatives with corresponding $Hint\_Sentence_s$ and $Hint\_Word\_i_s$ nodes. Each node has possible values *effective* and *ineffective*, where *effective* means that the tutorial help is immediately effective at causing the student to know the help topic (i.e., for *Hint_Word_i*, to know how to read word *i*; for *Hint_Sentence*, to know how to read the sentence as a whole). *Tutor Efficacy$_0$* nodes combine with the *Tutor Action$_1$* node to influence the *Student Knowledge$_1$* node or nodes corresponding to the help topic: $Hint\_word\_i_1$ influences $Know\_Word\_i_1$ while $Hint\_Sentence_1$ influences every *Student Knowledge$_1$* node. *Student Knowledge$_1$* in turn influences *Student Action$_2$*. The success of *Student Action$_2$* provides evidence about tutor efficacy that is propagated diagnostically through *Student Knowledge$_1$* to the appropriate *Tutor Efficacy$_0$* node.

The *Tutor Efficacy$_s$* nodes tune the network to the particular student, reducing the need for developers to provide accurate conditional probabilities regarding the effects of *Tutor Action$_1$* on *Student Knowledge$_1$*, and helping RTDT to avoid repeating ineffective tutorial actions.

Implementation

With input from a problem solution graph (CTDT) or sentence text (RTDT), the action selection engine creates the initial TACN. Thereafter, it recommends tutorial actions, accepts inputs representing student actions, updates the network, and rolls the DDN forward to decide each new tutorial action. Below, we describe some methods we used to cope with the size of the networks: factoring techniques and automatically creating conditional probability table entries.



Factoring Techniques

As part of creating TACNs, the action selection engines use factoring techniques to reduce the sizes of conditional probability tables. For example, the *Tutor Action Topic*$_1$ (CTDT) and *Tutor Action*$_1$ (RTDT) decision nodes can have many values: in CTDT, *null* plus every problem step and related rule; in RTDT, four values plus every word in the sentence. These decision nodes have arcs to many nodes in the *Student Knowledge*$_1$ and *Student Focus*$_1$ subnetworks. Without factoring, each arc would cause the number of entries in the target node's conditional probability table to be multiplied by the number of tutor action alternatives even though there is a direct influence only if the tutor action topic corresponds to the target node. Instead, we insert a *filter node* between these tutor action nodes and each target node. Each filter node reduces the influence of the tutor action node to a binary distinction: either the tutor action node directly influences the target node or it does not. Since filter nodes have only one input (a tutor action node) and a binary output, their conditional probability tables remain relatively small while they limit the increase in the size of their target nodes' conditional probability tables to a factor of two. We employ filter nodes wherever warranted throughout TACNs but we do not clutter network diagrams with them because they have no effect on network semantics. For similar reasons (as described previously), we insert a *funnel subnetwork* between the step nodes in CTDT's *Student Knowledge*$_1$ subnetwork and *Student Action Type*$_2$ (Murray, 1999).

Conditional probability table entries

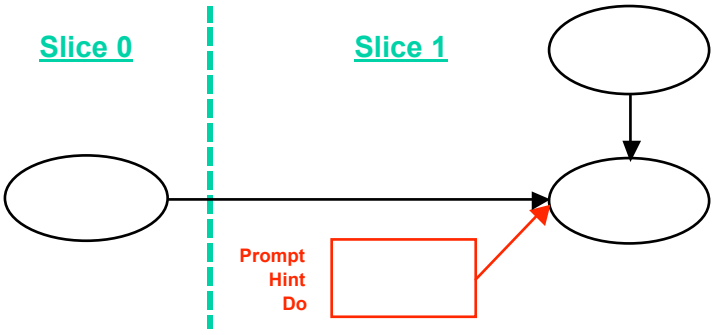
Conditional probability table (CPT) entries make up the vast majority of a TACN's numeric entries. They often follow patterns so that the CPTs of different nodes may be similar in many respects even if the nodes have different numbers of influences from different parent nodes. For instance, consider the CPTs of two different CTDT *Student Knowledge*$_1$ subnetwork step nodes, one of which has one parent (antecedent) step, and the other of which has two parent steps (both steps also have rule node parents). For both nodes, if any of the parent nodes is *unknown*, then the student does not have the information to complete the step successfully, so the step is likely to be *unknown* unless the student guesses correctly. Conversely, for both nodes, if all of the parent nodes are *known*, then the step is likely to be *known* unless the student somehow slips in making the inference required for the step. The probability of a correct guess or a slip is likely to be about the same for both steps, at least in the absence of knowledge about special circumstances related to either node - probabilities summarize uncertainty due to such ignorance anyway (Russell & Norvig, 1995). Similar levels of tutor help may also have similar probabilities of helping a student know different steps, again in the absence of more specific knowledge, and especially in extreme cases, such as when the tutor simply prompts the student (providing little or no information) or when the tutor tells the student exactly how to do a step.

We use rules to specify such patterns along with numeric parameters representing the probability of a correct guess, a slip, success after a specific level of tutor help, etc. For our prototype implementations, we used our best judgment to set default values for CPT parameters, prior probabilities and utilities, leaving obtaining more accurate values as an important goal for future research. The action selection engines also accept an optional file to specify any probability or utility values that differ from the defaults.

Figure 8, along with the rules listed below, provides an example of automated CPT creation for CTDT *Student Knowledge*$_1$ subnetwork step nodes. The table in Figure 8 is the CPT for the slice 1 node $dx/dy = bay^{b-1}$ (shown in Figure 3) except that it has been simplified as follows: (1) The step's slice 1 parents - its antecedent steps and related rule - are simplified to a single node, *Antecedent Steps & Rule*, which has value *unknown* if any of the parents are *unknown* and value *known* otherwise. (2) The *Tutor Action Type*$_1$ decision node has just 3 alternatives - *prompt*, *hint*, and *do* - which can be extended as described below. (3) The influence of *Tutor Action Topic*$_1$ (through a filter node) is not shown. If the value of *Tutor Action Topic*$_1$ does not correspond to this node (i.e., if the tutor addresses some other knowledge element), then *Tutor Action Type*$_1$ does not directly influence the student's knowledge of this node. For these cases, the table entries are the same as the table entries for the content-free *Tutor Action Type*$_1$ value of *prompt*, which also does not influence the student's knowledge of this node.

The rules and parameters follow. Each rule specifies only the conditional probability that the node is *known*, *p(known)*, since *p(unknown)* is simply *1 - p(known)*.

1. If a step is *known* in slice 0, then the step is *known* with probability *1 - f*, where *f* is a parameter representing the probability of forgetting a step known in the previous slice.



Conditional Probability Table for $dx/dy = bay^{D-1}$ in Slice 1

Step, Slice 0	Unknown						Known					
Antecedents	Unknown			Known			Unknown			Known		
T Action Type	Prompt	Hint	Do	Prompt	Hint	Do	Prompt	Hint	Do	Prompt	Hint	Do
Unknown	1 – g	1 – h	s	s	s	s	f	f	f	f	f	f
Known	g	h	1 –s	1 – s	1 - s	1- s	1 - f	1 - f	1 - f	1 - f	1 - f	1 - f

g= probability of a correct **guess**
h= probability that a **hint** about an unknown step will be successful
s= probability of a **slip** on a known step
f= probability of **forgetting** a step known in the previous slice

2. Otherwise, if all of the step's parents are *known*, then the step is *known* with probability *1 - s*, where *s* is a parameter representing the probability of a slip on a known step.
3. Otherwise, the probability that the step is *known* depends on *Tutor Action Type*$_1$:
   - If *Tutor Action Type*$_1$ is content-free, such as *prompt*, then the step is *known* with probability *g*, where *g* is a parameter representing the probability of guessing an unknown step correctly.
   - If *Tutor Action Type*$_1$ is *do* (tell the student exactly how to do a step), then the step is *known* with probability *1 - s*.
   - If *Tutor Action Type*$_1$ is neither content-free nor as explicit as *do*, the step is *known* with a probability corresponding to the efficacy of *Tutor Action Type*$_1$ at conveying the information. In Figure 8, this probability is *h*, a parameter representing the probability that a *hint* about an unknown step will be successful. This schema is easily extended to various levels of *hint* efficacy and to other *Tutor Action Type*$_1$ values such as *teach*.

TUTORIAL ACTION SELECTION EVALUATION

We had two goals for evaluating DT Tutor's action selection capabilities. First was to determine whether DT Tutor's action selections are rational in light of its probabilistic beliefs and utilities. This included testing whether DT Tutor is sensitive to changes in these values. Second was to

determine whether DT Tutor can select actions that are comparable to those of other tutors in similar situations. At the time of this evaluation, we had not yet developed a modern graphical user interface for CTDT, and RTDT has not been integrated with the Reading Tutor, so for testing we created a temporary text interface to simulate student inputs and tutorial actions (e.g., for CTDT, the student and tutor action *type* and *topic*).

As discussed above, the tutor must balance multiple competing objectives. This balance is affected by the tutor's beliefs and utilities, so variations in any of the probability or utility values may be enough to sway the balance to a different tutorial action selection. A TACN incorporates thousands of real-numbered probability and utility values that may be varied in any combination, and the number of possible variations for each value is unbounded, so the most we can do is sample from this unlimited space. To sample from this space in such a way that the results would be comprehensible, we first used our best judgment to initialize the system with reasonable default values using parameters as described in the previous section. From a reasonable starting state, it is easier to attribute the results of changes to one or more probability or utility values to those changes rather than to odd interactions among the values.

DT Tutor's modular construction makes it easy to isolate its components, and we used this capability for testing. Each of the major outcomes - e.g., the student's knowledge, focus of attention, and affective state; task progress; and the discourse state - corresponds both to a subnetwork of the TACN and to a subutility node. Total utility is a weighted sum of the subutilities for the major outcomes. To isolate network components related to any subset of TACN outcomes, we simply assign a weight of zero to the subutilities for all other outcomes. Of course, this capability makes for a flexible tutor as well. For instance, one can easily configure the tutor to focus on increasing the student's knowledge at the expense of task progress, or vice versa, simply by changing the weights assigned to student knowledge and task progress.

We used both action selection engines to test DT Tutor's major components individually and in various combinations while we varied probability and utility values. In addition, we tested the tutor's action selections with all components active through all steps of problems while simulating a range of student action types. Due to space limitations, we present only some highlights below.

Considering Only the Student's Knowledge

With CTDT, we tested the effects of varying prior probabilities for student rule and problem step knowledge while considering only knowledge-related outcomes. First, the prior probability that each rule was *known* was fixed at *0.5* while we varied the prior probability that each step was *known* from *0.1* to *0.5* to *0.9*. Then, we reversed the manipulation, holding the prior probabilities for the step nodes fixed at *0.5* while similarly varying prior probabilities for the rule nodes. With equal prior probabilities that rules and steps are *known*, tutorial actions that increase rule knowledge were preferred because increasing rule knowledge also increases the probability that the student knows task steps that depend upon that knowledge (accomplishing both objectives at once). Effective human tutoring is correlated with teaching generalizations that go beyond the immediate problem-solving context (VanLehn et al., 2003). Otherwise, CTDT tended to prefer to address the topic with lower probability of being *known*.

There is at least one situation in which a human tutor might choose not to devote attention to a rule or task step that has a low probability of being *known*: when the tutor cares little about

whether the student learns the particular rule or task step. By helping the student get past the need for low utility knowledge, the tutor and student can proceed with higher priority goals. We tested CTDT's capability to emulate such behavior by setting a low prior probability for knowledge of the rule required to do a particular problem step and setting a low subutility for that rule. The tutor recommended a tutor action topic corresponding to the problem step and a tutor action type of *do*, which helped the student get past the step without learning the rule. These tests demonstrate DT Tutor's capability to emulate human tutors' proclivity to prioritize their actions based on the student's needs and to avoid wasting time addressing topics that the student does not need to know (Merrill et al., 1995).

Trade-offs between the Student's Knowledge and Task Progress

With CTDT, we tested the effects of assigning various weights to the subutilities for student knowledge (*Student Knowledge*$_2$) and task progress (*Student Focus*$_2$). When the weights were apportioned only to student knowledge, tutorial actions that increase rule knowledge were preferred. When the weights were apportioned equally, tutorial actions that increase rule knowledge were again preferred, all other things being equal. This is because increasing rule knowledge also increases the probability that the student knows how to do related steps and thus to make progress on the tutorial task. When the weights were apportioned only to task progress, tutorial actions that directly increase step knowledge with high probability (e.g., *do*, *teach*) were preferred. However, tutorial actions that directly increase step knowledge with only low probability (e.g., a *prompt* or an ineffective *hint*) were sometimes less preferred than tutorial actions that increase knowledge of a related rule. Again, this is because increasing the student's rule knowledge reaps rewards for task progress as well.

Considering Only the Student's Affective State

With CTDT, we tested the effects of considering only individual components of the student's affective state (*independence* or *morale*). Only the *Tutor Action Type*$_1$ component of the tutor's action influences the student's affective state, so we expected that the tutor action alternatives would be ranked solely by tutor action type, and this proved generally to be the case.

However, there was a secondary influence that we should have anticipated. *Student Action Type*$_2$ also influences the student's affective state, with a value of *correct* improving both *Morale*$_2$ and *Independence*$_2$. The influence of the student's future action on the student's affective state is mitigated by uncertainty about what that action will be, so it is a secondary influence. Among tutor action types with just a small positive or a negative direct influence on student affect, the influence of the predicted student action type sometimes outweighed the direct influence of the tutor action type. For instance, when considering student independence, action type *do* was preferred over *negative feedback*, even though *do* was more likely to decrease *Independence*$_1$, because *do* was also most likely to lead to a *correct* student action. Human tutors are likewise sensitive to the student's affective state but sometimes willing to take risks with it in order to teach something, in part because they realize that success will be beneficial for the student's affective state in the long run (Lepper et al., 1993).

We also varied prior probabilities for the affective state variables. We tested the effects of three different prior probability distributions for each variable: a medium distribution centered

around the middle of the five values, a low distribution and a high distribution. The direction and magnitude of the influence of the *Tutor Action Type$_1$* alternatives does not depend on prior probabilities (except for ceiling and floor effects), so we hypothesized that varying prior probabilties for student affect variables would change the expected utility *values* associated with the various *Tutor Action Type$_1$* alternatives, but not their preference *order*. This turned out to be true for many cases, but a surprise effect was that for student morale with a high prior probability distribution, tutor action types *do* and *teach* were most preferred even though *positive feedback* was most likely to directly increase student independence and morale. In retrospect, this is not so surprising: With *Morale$_0$* already likely to be high, not much expected utility could be gained by further increasing the probability of *Morale$_1$* being high. Instead, maximizing the probability that *Morale$_2$* would be high could best be ensured by increasing the probability that *Student Action Type$_2$* would be *correct*. Thus, DT Tutor is willing to take more risks with a student's morale when it is high in order to increase the student's knowledge or task progress. Lepper and colleagues (1993) conjecture that expert human tutors are also likely to provide less emotional reassurance and support to more able and self-assured learners.

Providing Proactive Help by Considering both Informational and Affective Outcomes

We tested DT Tutor's ability to provide proactive help when the tutor believes the student needs it, and conversely to refrain from providing help when the student does not appear to need it. Most ITSs do not provide proactive help, reacting instead to student errors and impasses. Human tutors sometimes provide proactive help (Lepper et al., 1993; Merrill et al., 1995), but most do not provide it on every turn unless it is necessary. This may be in part because they also consider other factors, such as the student's affective state, in addition to the student's cognitive state (Lepper et al., 1993). CTDT's explicit consideration of the student's affective state counterbalances objectives involving task progress and the student's cognitive state. For instance, providing help only when necessary promotes the student's feeling of independence, and providing proactive help serves to maintain student morale by avoiding failures. RTDT does not explicitly reason about the student's affective state, but it does reason about objectives with affective impact. These objectives include maximizing correct reading without the tutor's help, which inhibits the tutor from giving proactive help unless it is needed, and minimizing incorrect reading, which spurs RTDT to provide proactive help when necessary.

With both CTDT and RTDT, we varied prior probabilities for student knowledge elements - steps and rules for CTDT; words for RTDT - to test whether the tutors would intervene with proactive help when appropriate. When the probability was low that the student had the knowledge necessary to complete the next problem step (CTDT) or read the sentence (RTDT), DT Tutor suggested help before the student could experience failure. Conversely, with both CTDT and RTDT, the tutor did not suggest help (i.e., it selected a *null* action) when prior probabilities indicated that the student was likely to be able to complete the next problem step (CTDT) or read the sentence (RTDT) successfully.

Considering the Student's Focus of Attention and Discourse State Relevance

For CTDT, the *Discourse State Relevance$_1$* outcome models the extent to which the tutor cooperates with the student's focus of attention, assuming a depth-first topic bias. We tested

CTDT's ability to cooperate by configuring it to consider only the utility of the $Relevance_1$ outcome when selecting tutorial actions for the problem whose problem solution graph is shown in Figure 3. Of course, CTDT normally has other objectives as well, such as facilitating task progress, in which case CTDT might choose not to cooperate with the student's focus of attention - for instance, in order to get the student back onto a productive problem-solving track.

First, we simulated a student completing the *Apply Chain* step. Next, the student has a choice among two subgoal setting steps, *Find dx/dy:?* and *Find dy/dz:z*, which happen to have the same rule parent, *Chain Ops*. Since these subgoals share the problem step parent that was just completed, there should be no depth-first preference between them, and testing verified that there was not. Next, the simulated student completed setting one of the subgoals, *Find dx/dy:?*. Now depth-first topic preference is applicable. Depth-first, the next step is *Apply Diff1*, so topics related to completing step *Apply Diff1* (the step itself plus its rule parent, *Diff LHS*) should be preferred, and indeed they were. To test the flexibility of DT Tutor's depth-first topic preference, we next simulated the student completing step *Find dy/dz:z*. Now, the next step depth-first is *Apply Diff2*, so *Apply Diff2* should be preferred over *Apply Diff1* (both steps have the same rule parent, *Diff LHS*), and again it was. Thus, CTDT emulated a common human tutorial tendency to support the student in productive lines of reasoning rather than dictating which step to attempt next.

Considering the Efficacy of Tutorial Actions

With RTDT, we tested DT Tutor's ability to model the efficacy of the various tutorial action alternatives and choose its actions accordingly. First, we varied prior probabilities for the $Hint\_Sentence_0$ and $Hint\_Word\_i_0$ nodes in the $Tutor\ Efficacy_0$ subnetwork. RTDT preferred tutorial actions that it believed to be more effective, all other things being equal. Next, we simulated tutorial hinting actions (*hint_sentence* and *hint_word__i* for each word *i* in the sentence) and the subsequent student reading action (*not_read*, *error*, or *correct* for each word in the sentence, along with either a *fluent* or *disfluent* reading of the sentence as a whole). After the student's action, we updated the network and checked the value of the $Hint\_Sentence_2$ or $Hint\_Word\_i_2$ efficacy node corresponding to the tutor's action. For tutorial *hint_word_i* actions, a subsequent student reading action of *correct* for word *i* increased the corresponding *Hint_Word_i* node's probability of being *effective*, while student-reading actions of *not_read* or *error* decreased it. For the tutorial *hint_sentence* action, the probability that the corresponding *Hint_Sentence* node was *effective* depended upon the student's reading of each word *i* and the sentence as a whole. Finally, we rolled the TACN forward and verified that RTDT prefers not to repeat ineffective tutorial actions.

Tutorial Action Selections for a Complete Problem

We tested CTDT's action selections while simulating student actions through all problem steps. Here, we recount a simulation for the problem whose solution graph is shown in Figure 3. At the beginning of the problem, the only step whose prerequisite steps are all *complete* but that is not yet complete itself - i.e., that is ready to be completed - is the step *Apply Eval*, so in $Student\ Focus_0$ it is *in_focus* with probability *1.0*. The option has been set for CTDT to consider only *ready* or *in_focus* steps (plus any step that was just completed) and their related rules as tutorial

action topics, so the *Tutor Action Topic*$_1$ alternatives are *null* (no topic), the step *Apply Eval*, and the rule *Eval LHS* (the parent rule of *Apply Eval*). For a calculus student, there is a high probability that step *Apply Eval* (setting the goal of applying evaluation to an equation for *dx/dz* using the value *z=c*) and its related rule are *known*. CTDT selects action *null / null*[3], allowing the student to attempt the step on her own. The simulated student completes the step correctly. The next step is to set the goal *Find dx/dz:z*, which again is most likely *known*. Instead of proactive help, the tutor gives positive feedback on the previous student action (*Apply Eval / positive feedback*). The student completes step *Find dx/dz:z* correctly.

The next step is *Apply Chain* (setting the goal of applying the chain rule). The prior probabilities that this step and its related rule, *Chain LHS*, are *known* are low. CTDT selects the action *Apply Chain / teach*. The student's subsequent action is *Apply Chain / error*, decreasing probabilities that the step and its related rule are *known* and decreasing probabilities for high morale and independence values. On its next turn, CTDT selects the action *Chain LHS / teach*, increasing probabilities that the rule and its related step are *known* but decreasing the probability that the student's independence is high. The student completes the step correctly, further increasing probabilities that the step and the rule are *known* and increasing probabilities that morale and independence are high.

Two steps are now *ready* or *in_focus*: *Find dx/dy:?* and *Find dy/dz:z*, both with rule parent *Chain Ops*, which has a middling value of being known. CTDT selects action *Chain Ops / hint*, which not only increases the probability that the rule is *known*, but also increases the probability that both of the steps are *known*. The student correctly sets the goal *Find dx/dy:?*.

There is a relatively high prior probability that the student knows how to set the next goal (depth-first), *Apply Diff1*, so CTDT just provides positive feedback on the previous student action. Instead, the student sets the goal *Find dy/dz:z*. CTDT follows the student in switching to the part of the solution path below *Find dy/dz:z* for depth-first topic preference, now preferring topic *Apply Diff2* to *Apply Diff1*. However, there is still a high probability that the student knows how to set the goal of applying differentiation, so CTDT just provides positive feedback on the previous student action. The next few steps along both branches of the solution path (which are identical in form) - *Apply Diff1*, $dx/dy=bay^{b-1}$, *Apply Diff2*, and $dy/dz=fez^{f-1}$ - all have a high prior probability of being *known*, so CTDT continues to provide positive feedback on previous student actions as the student completes them correctly.

The next step is $dx/dz=bay^{b-1}fez^{e-1}$, the application of the chain rule to the two differentiated equations, with parent rule *Chain Exec* (a rule about how to execute the chain rule). There is only a middling probability that *Chain Exec* is *known*, so CTDT selects tutorial action *Chain Exec / hint*, which is less likely than a *teach*ing action to increase the student's knowledge, but also less likely to decrease the student's independence. Unfortunately, the subsequent student action is an *impasse*. The tutor teaches the step directly ($dx/dz=bay^{b-1}fez^{e-1}$ / *teach*). The student completes the step successfully.

The last step is $dx/dz= bay^{b-1}fec^{e-1}$ with parent rule *Eval Exec* - how to execute evaluation. There is a high probability that the student knows the rule and therefore how to arrive at the answer, so CTDT just provides positive feedback on the previous student action. The student executes the step correctly, completing the problem.

---

[3] In this section, tutor and student actions are specified in the format <*topic*> / <*type*>, where the first position indicates the action *topic* and the second position indicates the action *type*.

Conclusions Regarding Tutorial Action Selections

On the tests described above, DT Tutor's action selection engines behaved rationally in a variety of tutorial situations, both when analyzed in terms of functional components and as a whole. The tests demonstrated that DT Tutor is sensitive to changes in its beliefs and utilities, adapting its actions accordingly. We presented testing scenarios which suggest that, by weighing some of the same considerations that may influence human tutors' decisions, DT Tutor's action selection engines may be able to emulate some of their behaviors. Since there is good evidence that people often reach different conclusions than decision-theoretic systems (see, e.g., Kahneman et al., 1982), we anticipate that in at least some cases the actions of human and decision-theoretic tutors will diverge, not necessarily always to the discredit of decision-theoretic tutors.

We also showed that, by considering combinations of tutorial state attributes over a continuous range of probability and utility values, DT Tutor's applications can select rational actions not just in selected situations to which heuristic rules might apply, but also:

- In environments where it may not be practical to encode a heuristic for each situation - for example, when the number of combinations of tutorial state attributes is unbounded.
- In unanticipated situations - for example, when a tutorial action that has a less positive impact on the student's affective state in the short-term has a more positive impact on the student's affective state in the long-term (due to learning and successful problem solving).
- In situations where heuristics involving different factors suggest conflicting actions - for example, when proactive help would decrease the student's feeling of independence but refraining from proactive help would leave the student highly likely to fail.

TRACTABILITY EVALUATION

We conducted a tractability evaluation to determine whether DT Tutor can be used to select tutorial actions quickly enough to keep students engaged for real-world tutoring applications. Probabilistic inference is NP-hard in the worst case for both exact (Cooper, 1990) and approximate (Dagum & Luby, 1993) algorithms. DT Tutor's networks possess several characteristics that can make inference challenging:

1. Multiply-connected, with some network nodes having three or more parents, for which exact inference can be NP-hard (Cooper, 1990). Multiply-connected networks seem to be necessary to represent many complex, real-world domains (Cooper, 1990).
2. Large (e.g., Cheng & Druzdzel, 2000; Russell & Norvig, 1995), also as seems to be necessary for many complex, real-world domains (e.g., Cooper, 1990)
3. Temporal (Cooper et al., 1989), increasing both the number of nodes (for multiple slices) and connectivity, with temporal as well as atemporal arcs (Ngo et al., 1996)
4. Large conditional probability tables (e.g., Cheng & Druzdzel, 2000)
5. Decision networks, requiring an update for each alternative (Russell & Norvig, 1995)

While some network characteristics can be addressed by specialized inference algorithms (see,

e.g., Lin & Druzdzel, 1999), DT Tutor's combination of characteristics and requirement for real-time inference pose a stiff test. Therefore, it is critical to determine whether our approach can be tractable for real-world tutoring applications and to see how the computation will scale.

For our purposes, it suffices to select the decision alternative that has maximum expected utility. We do not need to know expected utility values or the posterior probabilities of chance nodes at the time the decision is made. Hence, a decision algorithm that does not take the time to compute these values (e.g., Shachter & Peot, 1992) would be most efficient for our application. However, we did not have an implementation available that supports our large structured utility models. Instead, we used Cooper's (1988) algorithm which converts the decision network into an equivalent belief network and then updates the belief network for each decision alternative, computing all expected utilities and posterior probabilities.

Cooper's algorithm supports the use of any belief network algorithm, so we tested with both an exact clustering algorithm (Huang & Darwiche, 1996) and an approximate algorithm, *likelihood weighting* (Shachter & Peot, 1989), with 1000 samples. Stochastic simulation algorithms like likelihood weighting usually work well when any evidence is at the root nodes (e.g., Russell & Norvig, 1995), as it is in our TACNs at the time the best decision is computed. The exact algorithm was too slow with the larger problems. Since we do not require exact probabilities and expected utilities, and state-of-the art rollup schemes for dynamic temporal networks typically approximate the current belief state anyway (see, e.g., Boyen & Koller, 1998), we present in Figure 9 results obtained using the approximate algorithm.

We tested response times for both CTDT and RTDT with a range of problem sizes on two computer systems: a 667-MHz Pentium III with 512-Mb of RAM running Windows '98, and a 1.8-GHz Pentium 4 with 2-Gb of RAM running Windows 2000. We present only response times for the tutorial action selection phase of the TACN life cycle, which are worst case since network updates after the tutor's action has been decided are much faster. For RTDT, we present response times before the student's first attempt at a sentence, which are worst case even for the tutorial action selection phase because for subsequent attempts RTDT does not consider hinting on words that the student has already read correctly.

For CTDT, we tested response times for calculus problems with problem solution graphs representing from 5 to 23 problem-solving steps, with corresponding TACNs ranging from 123 to 318 nodes. The 5-step problem had only one solution path, while the 23-step problem had four solution paths. The number of solution paths per problem depends on the types of solution paths (many of which may be inefficient) allowed by the automated problem solver. The topologies of the problem solution graphs vary depending on (1) the number and types of solution paths, and (2) the particular steps and related rules required to solve the problems. The test results are therefore sample points for problem sizes ranging from the smallest to approximately the largest that we expect to encounter. Mean response times on the Pentium III system ranged from 1.07 seconds to 3.13 seconds, growing approximately linearly. Mean response times on the Pentium 4 system were faster on problems with up to 15 steps, ranging from 0.48 seconds for the 5-step problem to 1.49 seconds for the 15-step problem. For the 19- and 23-step problems, response time grew more than linearly and was actually slower than on the Pentium III system for reasons related to the machines' configurations.

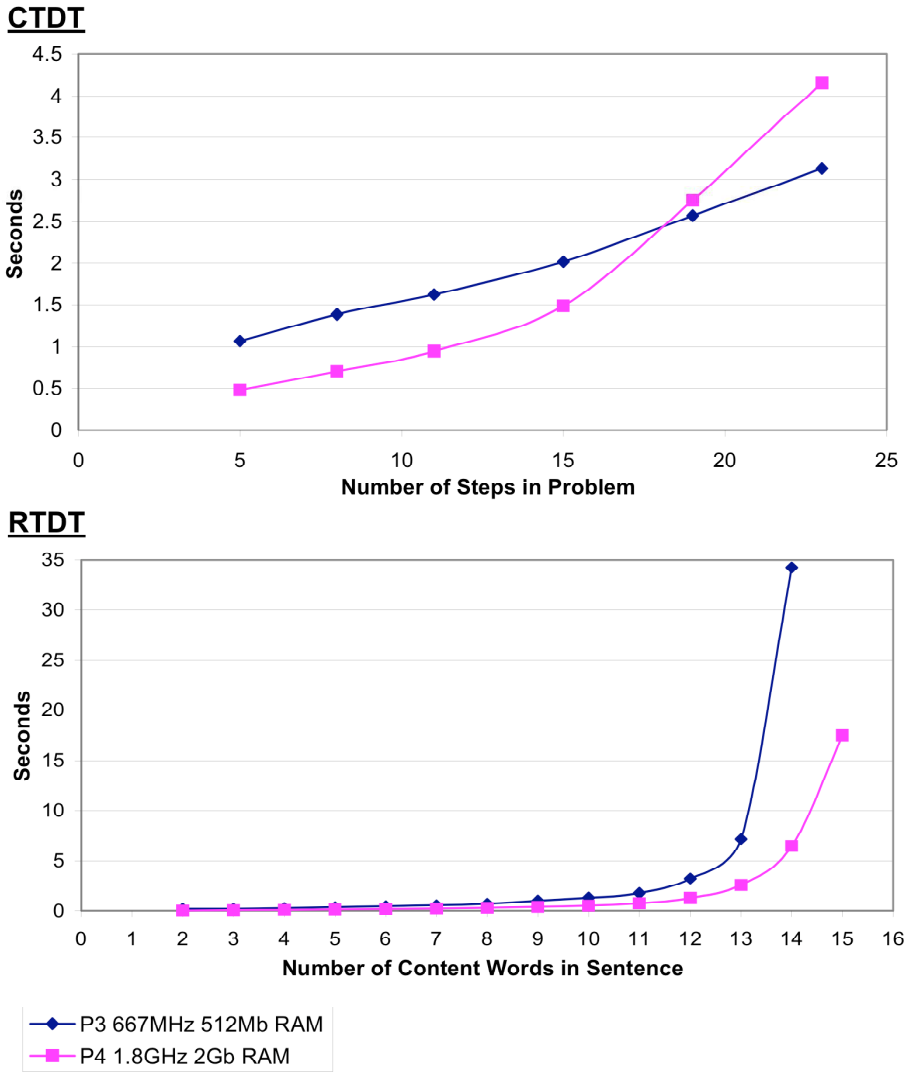


Fig. 9. Tutorial action selection response time for CTDT and RTDT on Pentium III (P3) and Pentium 4 (P4) systems: mean over 10 trials using likelihood weighting with 1000 samples.

For the Reading Tutor's corpus of readings, sentence length ranges from 5 to 20 words as reading level progresses from kindergarten through fifth grade, with about two-thirds content words. We tested response times for sentences with 2 to 15 content words. The number of nodes in the corresponding TACNs grew linearly with the number of content words from 52 to 273. Network topologies for sentences with the same number of content words are identical, so our results represent all such sentences. Our response time goal was 0.5 seconds in order to keep readers engaged in a natural interaction. On the Pentium III system, mean response time met the goal for up to 6 content words. Response time grew approximately linearly up to about 8 content

words, and was within about a second up to 9 content words. Thereafter, response time increases accelerated for sentences with up to 14 content words, when the system ran out of RAM. The system was unable to load the TACN for a sentence with 15 content words. On the Pentium 4 system, response time was close to the goal for up to 10 content words, ranging from 0.03 to 0.52 seconds. Response time grew approximately linearly up to about 10 content words, and was within a second up to 11 content words. Response time increases then accelerated for sentences with up to 15 content words, when RAM usage reached nearly 1 gigabyte.

Besides taking advantage of faster hardware, other speedups are practicable. A decision network algorithm that does not compute all expected utilities and posterior probabilities (e.g., Shachter & Peot, 1992) should improve response time. Stochastic simulation algorithms are amenable to parallel computing algorithms (Binder et al., 1997) which could practically divide the response time by the number of machines that can be used effectively. Furthermore, many approximate algorithms have an *anytime* property that allows an approximate result to be obtained at any point in the calculation (while further calculation yields more precise results), so that an approximate result can be obtained at any time that it is needed.

In summary, if reasonable response time is defined as 1.0 seconds for CTDT and 0.5 seconds for RTDT, DT Tutor on the faster system can already select tutorial actions within reasonable response time for half of CTDT's problems and for 8 out of 14 of RTDT's sentences. Response time was less than 1 second for 10 out of 14 of RTDT's sentences. Response time grew approximately linearly with the size of the problem or sentence except on the most challenging scenarios. A faster system with more RAM bought faster response times except on the most challenging CTDT problems, and for RTDT increased the number of sentences for which response time grew approximately linearly. These results were obtained using prototype action selection engines for diverse, real-world domains. Moreover, significant speedups are practicable using currently available hardware and software solutions. With such speedups applied, and as hardware and software capabilities continue to improve, it seems realistic to anticipate that DT Tutor can be used to select tutorial actions within reasonable response time for real-world tutoring applications.

RELATION TO PRIOR WORK

Related work extends beyond ITSs to include other types of user modeling research because many systems that are not explicitly educational model the user (for an ITS, the student) to inform decisions about what actions to take in order to facilitate the interaction. Below, we describe related work in terms of important elements of the design space for a user modeling system: deciding what actions to take, modeling change in the user and the situation over time, modeling only observable or also unobservable attributes, modeling the user's focus of attention, modeling the user's affective state, and predicting the user's next action.

Making Decisions

Applications that use a belief network representation often resort to heuristics to decide which action to take. For instance, Andes1, the first version of a physics ITS from which DT Tutor is descended, used heuristics to decide the topic of *what-next?* help (Gertner et al., 1998). Like

many other belief network applications, Andes1 incorporated no explicit notion of the utilities of the possible outcomes of its actions. Such applications cannot integrate considerations regarding the probabilities and utilities of the possible action outcomes. Instead, Andes1 selected actions using probability thresholds and rules which reflect implicit priorities.

Some applications take outcome probabilities computed by a belief network and multiply them by their associated utilities outside the network to compute expected utilities for decision-theoretic action selection. These include an ITS for English capitalization and punctuation (CAPIT, Mayo & Mitrovic, 2001) and various other user modeling applications (e.g., Horvitz et al., 1999). An advantage of computing expected utility outside the network is potentially faster inference due to a smaller network (no decision or utility nodes with associated arcs) and the flexibility to consider only subsets of actions or outcomes in the expected utility calculations. However, the potential speedup is mitigated by (1) forgoing the option to use specialized decision network algorithms such as those that find the decision with maximum expected utility without computing exact expected utility values for all alternatives (e.g., Shachter & Peot, 1992), and (2) the potential to miss less obvious decision alternatives or outcome combinations with higher expected utility, resulting in decisions with less than maximum expected utility.

A few user modeling applications use decision network or equivalent representations to directly compute the decision with maximum expected utility. DT Tutor and Conati's representation (2002) for an educational game use DDN architectures to select actions for helping a user with a task. *iTutor* (Pek, 2003) uses a DDN for deciding actions at a different grain size: pre-computing a policy for selecting curriculum topics such as which problems to present to a student. Jameson and colleagues (2001) use a decision network to decide whether to present instructions individually or several at a time.

One benefit of decision-theoretic representations is support for value of information computations to guide user queries and other information-seeking behaviors. Applications that utilize value of information include those of Horvitz and colleagues (Horvitz et al., 1998; Paek & Horvitz, 2000) and *iTutor* (Pek, 2003). DT Tutor does not currently query the user or make decisions about other information-seeking behaviors and so it does not utilize value of information at this time.

Modeling Change over Time

Systems that have used probabilistic networks to model change over time include POLA and Andes1, ancestors of DT Tutor, which employ a static atemporal belief network for each problem. POLA avoided temporal representation by dynamically adding nodes to represent problem-solving actions as the student completed them, along with nodes to represent the student's related physics knowledge (Conati & VanLehn, 1996). In effect, the semantics of each version of the incrementally-built networks changed with each time step to represent the tutorial state at the current point in time (Schäfer & Weyrath, 1997). Because POLA built its networks incrementally, it could not use them to model student knowledge related to uncompleted steps or to predict which action the student was most likely to attempt next (Conati et al., 2002). Andes1's networks do include nodes to represent uncompleted problem-solving actions and related knowledge, but the semantics of these nodes does not distinguish between steps that have already been completed and steps that Andes1 believes the student can complete (Conati et al., 2002). Thus, Andes1's networks cannot track the student's most recent action or current focus of

attention (Conati et al., 2002). Andes1 models the evolution of a student's knowledge at a high level by copying updated beliefs about the student's knowledge between the atemporal networks for each successive problem, but this modeling is at too coarse a grain size to influence tutorial actions while the student is working on any particular problem.

Horvitz and colleagues have modeled change over time with a set of single-slice network models by embedding the notion of time within variable definitions (e.g., "attribute *a* at time *t*") (Horvitz et al., 1998) or by encoding time-dependent conditional probabilities (Horvitz et al., 1998) or utilities (e.g., Horvitz & Barry, 1995). Usually, each successive network represents the current point in time. The state evolution model is specified externally to the networks and is implicit in the changing variable definitions, conditional probabilities and utilities. Without arcs across slices or an equivalent mechanism, many temporal dependencies may be neglected, such as the conditional dependence of attributes on their previous values Without nodes to represent beliefs in more than one slice, a network cannot model changes in beliefs about the present through evidence-based revision of beliefs about the past.

CAPIT (Mayo & Mitrovic, 2001) uses a two-slice static temporal belief network to predict student problem-solving actions in terms of constraints. It adapts conditional probabilities to the current student while she works, using an algorithm for atemporal models heuristically modified to give greater weight to more recent events. Thus, CAPIT adapts its static temporal belief network to reflect changes in the tutorial state beyond its two-slice limit. However, the network does not track the order in which constraints have been attempted or feedback has been given, so it cannot track the student's focus of attention or make a more specific prediction about the student's next action. CAPIT's student model is limited to observable constraints, so it cannot model the evolution of the student's knowledge or other unobservable tutorial state attributes.

Dynamic belief network representations can model the temporal evolution of the model's state over any number of slices, including projections about future slices (Russell & Norvig, 1995), by dynamically creating new slices and removing old slices as they are no longer needed. Reye (1996) proposed dynamic belief network representations for ITSs to model the evolution of the student's knowledge over time and showed (Reye, 1998; 2004) how two probabilistic ITSs (Corbett & Anderson, 1992; Shute, 1995) can be characterized as special cases of a dynamic belief network approach. Other user modeling applications include a game (Albrecht et al., 1998) and office productivity tools (e.g., Horvitz et al., 1999), among others. Dynamic belief networks share with static belief networks the lack of an integrated provision for decision-making.

A DDN extends a dynamic belief network representation to include decision-making capability. Both DT Tutor and Conati (2002) employ DDN architectures for both user modeling and decision-making. *iTutor* (Pek, 2003) uses a DDN to pre-compute which curriculum topics to present to the student but then uses a dynamic belief network to track the student's knowledge as she progresses through the curriculum.

Which Attributes to Model

The set of attributes that an application considers should naturally influence the actions that it selects. For instance, if a help or tutoring application does not consider the user's focus of attention, its help is liable to be directed towards a topic that the user is not concerned about, which may confuse the user (e.g., Gertner et al., 1998). Many ITSs consider only one or two sets of attributes, such as the student's knowledge and task progress. A strength of decision-theoretic

approaches is the ability to smoothly integrate considerations involving multiple sets of attributes. Below, we describe research related to modeling some of the more important attributes that DT Tutor can model.

Modeling Observable and Unobservable Attributes

Some applications have used statistical methods to probabilistically model only observable user attributes. These include a machine learning system for predicting the details of subtraction errors (Chiu & Webb, 1998), CAPIT (Mayo & Mitrovic, 2001), and ADVISOR (Beck & Woolf, 2000), an ITS for grade school arithmetic. Limiting modeling to observable user attributes affords the considerable advantage of simplifying machine learning efforts (e.g., Jameson et al., 2001). All required data can be gathered from log files and other readily observable sources that record values for the attributes of interest (e.g., Horvitz et al., 1998). Data that the system can observe (e.g., keystrokes, mouse actions and timing data in context) can even be used to adjust prior and conditional probabilities while the system is in use in order to further adapt to specific users or populations (e.g., Horvitz et al., 1998; Mayo & Mitrovic, 2001).

However, there are also important advantages to modeling unobservable attributes (Jameson et al., 2001). Perhaps foremost among these for ITSs is that they are usually concerned with the student's knowledge - often to influence and sometimes to assess - which is unobservable. An application must model attributes if it is to reason about them (Großmann-Hutter et al., 1999). Second, unobservable attributes often influence observable attributes. For instance, a student's knowledge influences the correctness of her problem-solving actions. So even if an application is concerned only with observable outcomes, it may be advantageous to consider its influence on unobservable attributes as well. In particular, ITSs often influence their students' observable behaviors through discourse and other actions intended to influence the student's mental state. Modeling conditional dependencies between observable and mental attributes allows one to leverage and even to test research from such fields as education and psychology (Großmann-Hutter et al., 1999). Finally, networks with hidden variables representing unobservable attributes can be more concise (e.g., Heckerman, 1995), making them faster to learn (Binder et al., 1997) and to update (Martin & VanLehn, 1995), with a structure that is easier to elicit from experts (Binder et al., 1997) and more amenable to interpretation in terms of theoretical and empirical knowledge (e.g., Binder et al., 1997; Großmann-Hutter et al., 1999). DT Tutor, like many other ITSs and other user modeling systems, models both observable and unobservable attributes.

Modeling the User's Focus of Attention

Identifying the user's focus of attention can be critical to providing assistance that is timely and relevant to the user's needs (e.g., Horvitz et al., 1999). According to Grosz and Sidner (e.g., 1986), knowledge of focus of attention as well as task structure is necessary for understanding and generating task-oriented discourse. CTDT follows Grosz in modeling focus of attention relative to a hierarchical task structure. However, instead of modeling focus with a stack as in the work of Grosz and colleagues (e.g., Grosz & Sidner, 1986), CTDT's probabilistic approach has more in common with Walker's (1996) cache model of attentional state. The cache model accounts for phenomena such as the influence of the recency of discourse content as well as the influence of the hierarchy of intentions related to the task. The cache model is also consistent

with Albrecht and colleagues' (1998) observation that users may interleave actions to achieve multiple goals. Reye (1995) criticizes the stack model's inflexibility regarding the order in which goals may be pursued within an ITS. DT Tutor also models *focus aging*, or decreasing probability of focus on task elements that were in focus at earlier times, which is consistent with both the cache model and Horvitz and colleagues' (1998) approach of associating observations seen at earlier times with decreased relevance to the user's current goals.

Andes1 uses a hierarchically-structured atemporal belief network to narrow in on a set of task steps that may be in the student's task-related focus of attention when she requests *what-next?* help. However, Andes1's network does not distinguish completed steps and cannot track the student's most recent action, so Andes1 uses a heuristic procedure to guess the student's specific focus of attention (Conati et al., 2002; Gertner et al., 1998). The Adele ITS for medical diagnosis (Ganeshan et al., 2000) likewise models focus of attention relative to a hierarchically-structured atemporal belief network. However, Adele does not model uncertainty about the student's focus of attention probabilistically, instead directing the discourse and asking disambiguating questions to limit the possibilities. The Lumière Project's help systems for office productivity programs probabilistically model focus of attention for non-ITS applications, but at least initially avoided detailed modeling of domain-specific content (Horvitz et al., 1998). Some other applications by Horvitz and colleagues (e.g., Horvitz et al., 1999; Paek & Horvitz, 2000) model focus of attention at mostly a coarser level, such as which agent or application program the user is attending to.

Modeling the User's Affective State

Considering the student's affective or motivational state can be vital for effective tutoring. Lepper and colleagues (1993) observed that their expert human tutors appeared to give as much weight to affective and motivational outcomes as to informational and cognitive outcomes, knowing that a negative affective state can interfere with learning (Goleman, 1995). Many ITSs consider the student's affective state at most implicitly, with corresponding effects on the affective sensitivity of the tutoring that they provide. Most ITSs and other user modeling applications that do consider the student's affective state pay relatively scant attention to other considerations.

For ITSs, detailed models of the student's affective state have been implemented by, for example, del Soldato and du Boulay (1995) and de Vicente and Pain (e.g., 2002). However, these models have at least two shortcomings. First, they do not model the ITS's uncertainty about the student's affective state. Arroyo and Woolf (2001) address this issue with a statistical approach for predicting the student's behavior and affective state. Second, they do not satisfactorily resolve what the tutor should do when there is a conflict between the best tutorial action based on affective outcomes and the best tutorial action based on cognitive or other outcomes.

Decision-theoretic approaches provide a way to take into account the tutor's uncertainty about the student's affective state while balancing considerations regarding affective and other outcomes. CTDT uses a DDN to weigh uncertain beliefs and multiple objectives regarding the student's changing affective state along with other tutorial outcomes. Conati (2002) likewise proposes a DDN representation to consider both the user's affective state and "learning state" for an educational game, employing a detailed model of the user's affective state but leaving the model of the user's learning state unspecified.

CTDT sports a relatively impoverished model of the student's affective state, and RTDT does not currently model affective state at all. DT Tutor's main contribution in this area is providing a framework for weighing uncertain, changing beliefs and priorities regarding any number of outcomes, including the user's affective state, at various levels of detail, depending on the needs and capabilities of the application.

Predicting and Learning from the User's Actions

ITSs and other user modeling applications often choose actions, at least implicitly, on the basis of beliefs about how they will influence the user's performance. Conversely, the user's performance can be used as evidence to update the application's user model. Therefore, it can be important for a user modeling application to predict the user's performance and to learn from the user's actual performance.

An application's prediction capabilities depend in part on the factors that it considers. For instance, Chiu and Webb (1998) consider the student's past subtraction performance in detail to arrive at detailed predictions about future subtraction performance, but they do not consider the influence of help. ADVISOR (Beck & Woolf, 2000), on the other hand, models many other factors, including the help provided, to predict the time required for a student to solve an arithmetic problem and whether she will be correct, but does not model or predict the student's performance on problem subskills. Jameson and colleagues (2001) likewise predict a user's execution time and errors based in part on the system's delivery of instructions. CAPIT (Mayo & Mitrovic, 2001) models and makes predictions about student performance in terms of 25 constraints.

All of the systems above model the user and make predictions strictly in terms of observable attributes, which facilitates empirical learning both prior to and during interaction with the user. However, modeling relationships between unobservable attributes, such as the user's knowledge and focus of attention, and observable user actions can help in predicting observable user actions. Furthermore, such models can be used for diagnostic learning about unobservable attributes based on observed user actions.

Albrecht and colleagues (1998) model an unobservable attribute, the user's quest in a game, as part of predicting the user's next action and location within the game space. Horvitz and colleagues (1999) and DT Tutor both model the user's focus of attention as part of predicting the user's next action. DT Tutor models focus of attention along with student knowledge at a finer grain size - particular task steps and rules within the tutorial domain - to predict the topic and the correctness of, but not the time required for, the student's next action.

ADVISOR (Beck & Woolf, 2000) and the systems that use probabilistic networks (e.g., Albrecht et al., 1998; Horvitz et al., 1999; Jameson et al., 2001; Mayo & Mitrovic, 2001; Murray & VanLehn, 2000) model the system's inherent uncertainty by predicting the user's next action probabilistically. The systems that use probabilistic networks also have the capability to learn diagnostically about unobserved attributes (e.g., the user's goal, knowledge, focus of attention, and even potentially observable attributes) based on observed user actions.

DISCUSSION AND FUTURE WORK

DT Tutor's main contribution is a decision-theoretic framework for a user modeling application to select actions rationally by weighing uncertain, changing beliefs and priorities regarding any number of outcomes. To create practical instantiations of our framework, our first challenge was to build sufficiently accurate yet computationally feasible models of the action selection problem in decision-theoretic form. Our second challenge was automatically creating effective TACNs from simple problem representations. We presented our solution for two domains, evaluating their response times and action selection capabilities. Next, we plan to extend CTDT into a full-fledged ITS and evaluate it with students.

Decision-Theoretic Framework

For CTDT, we devised a DDN representation that includes the student's changing knowledge, focus of attention, morale, feeling of independence, and next action, along with task progress, discourse relevance and discourse coherence. RTDT does not explicitly model the student's affective state or most aspects of the discourse state, but it adds models of the efficacy of tutorial actions and multiple student reading actions per turn. Few previous systems have modeled any of these tutorial state attributes decision-theoretically, let alone in combination.

Although not a requirement of the approach, modeling a spectrum of tutorial state attributes gives DT Tutor's action selection engines a flexible basis for making decisions. For instance, CTDT explicitly models tradeoffs between action types such as *hint*, *teach*, and *do* in terms of their effects on task progress, student knowledge and student affect, among other attributes, with none of the action types dominant along all dimensions. As a result, CTDT might progress from *hint*ing about a rule related to a task step to *teach*ing the step directly, or possibly even tell the student exactly how to *do* the step, as we demonstrated above. A heuristic tutor can achieve the same behavior by fiat. For instance, Andes1 (Conati et al., 2002) and the Cognitive Tutors (Anderson et al., 1995) always work through a sequence of hints starting with the least specific until they terminate at a bottom-out hint that is equivalent to CTDT's action type *do*. Because they use a fixed tutorial strategy, they do not need to explicitly represent tutorial state attributes such as student affect, so their representational requirements may be less complex. However, they pay for their simplicity by being less flexible. As our tests illustrated, DT Tutor's applications can adapt to a variety of circumstances, including selecting action type *do* first when appropriate. Moreover, CTDT and RTDT use the same sets of considerations to provide proactive as well as reactive help, which Andes1 and the Cognitive Tutors do not do. Despite increased representational requirements, it proved possible to implement action selection engines for both domains that require reasonable amounts of computational resources.

Another benefit of using the normative foundation of decision theory is clarifying a rationale for tutorial decisions (Jameson et al., 2001). Since the earliest ITSs, developers have used fixed policies such as "Do not tutor on two consecutive moves, no matter what" (Burton & Brown, 1982, p.91). Some were probably invented when developers observed that their tutor was ineffective or even "oppressive" (Burton & Brown, 1982, p.91). With a fixed decision-theoretic foundation, the only way to debug a decision-theoretic tutor is to understand why its behavior is wrong, and in particular, what objective is adversely impacted by the behavior and how the tutor could have predicted and thus avoided the situation. This might spur a developer to, for instance,

add a new kind of utility, provide more accurate probabilities or utilities, or add overlooked probabilistic influences. Thus, implementing decision-theoretic tutoring engines moves us one step closer to exposing a rational basis for tutorial actions, and thus perhaps to a deeper understanding of tutoring itself.

An important element of DT Tutor's design is looking ahead to explicitly predict the effects of the decision-maker's actions. While this is natural for a decision-theoretic application, it is rare for an intelligent tutoring system. Modeling the tutor's influence on the tutorial state enables the tutor to select the actions that it believes will be most beneficial to the student and to the resulting tutorial state. This requires probabilistically predicting how the tutor's actions will influence, for example, the student's knowledge, affective state, and focus of attention.

A novel component of DT Tutor's representation is its model of the user's focus of attention. Separate representations for the user's focus of attention and knowledge allow the system to probabilistically predict both the *topic(s)* of the user's next turn, based on the user's focus of attention, and the *type(s)* of action(s) in the user's next turn (e.g., whether the action(s) will be *correct*), based on the user's knowledge. Modeling the user's focus of attention also enables the system to be a cooperative discourse partner and to address topics at times when the user is likely to be interested.

DT Tutor's models of some tutorial state attributes, such as the student's affective state, are overly simple thus far. While we believe that considering the user's affective state is a key to improving effectiveness and usability for ITSs and other user modeling applications - and this is important future work - this has not been the main thrust of our research to date. Even with a simple model of the student's affective state as just one of several outcomes considered, DT Tutor is able to move beyond simply presenting a kinder, gentler or more entertaining interface to adapting its tutoring based on the perceived affective and cognitive needs of the user, just as expert human tutors appear to do (Lepper et al., 1993). DT Tutor adapts not only actions with apparent affective impact, such as *positive feedback*, but also actions with subtler affective impact, such as proactive help. Within DT Tutor's framework, models related to various attributes can vary in richness and detail depending on the needs and capabilities of the application.

We would like to extend DT Tutor's knowledge representation to include a more detailed model of the discourse state, to support user queries about task-related domain rules, to generate multiple system (e.g., tutor) actions on a single turn, and to use value of information to decide when to query the user about what.

Creating Effective *Tutor Action Cycle Networks*

Because TACNs can include hundreds of nodes, we needed to find a way to create them automatically from simpler problem representations (a problem solution graph for CTDT; sentence text for RTDT). Part of the challenge involved populating the model with many thousands of probabilities and utilities. While this remains an important area for future work, our solution for conditional probabilities was to develop a rule-based system with a relatively small number of easily modified parameters. Alternatively, DT Tutor's conditional probability table entries could be simplified using standard parametric representations such as Noisy-OR and Noisy-AND nodes. An advantage of a decision-theoretic representation is that it supports obtaining probabilities and utilities from any combination of the best sources available. For

instance, they can be based on (1) pedagogical, cognitive, or psychological theory, and (2) empirical data such as results from pretests, logged student interactions with the system, and post tests. Even the subjective beliefs and objectives of an ITS designer or administrator can be used in the absence of better information. In addition, RTDT's *Tutor Efficacy*$_s$ subnetworks reduce the need for accurate conditional probabilities regarding the influences of the tutor's actions on the student's knowledge. It is encouraging to note that Bayesian systems are often surprisingly insensitive to imprecision in probabilities (e.g., Henrion et al., 1996; VanLehn & Niu, 2001) and that decision quality is generally even less sensitive (Henrion et al., 1996) because only the rank of the decision alternative with maximum expected utility matters.

Even with an automated method for populating TACNs with probability and utility values, we still needed to control the number of arcs and conditional probability table entries. First, we employ factoring methods such as *filter nodes* to limit the number of conditional probability table entries without detracting from the normative status of TACNs. Second, we selectively specify arcs and associated conditional probability table entries as each new TACN is created: For CTDT, we included an option to limit the *Tutor Action Topic*$_1$ alternatives to *ready* and *in_focus* steps and related rules, along with any step that was just completed. Similarly, for RTDT's corrective feedback, we do not consider helping the student on words that she has already read correctly. Also, for CTDT, only *ready* and *in_focus* steps influence *Student Action Topic*$_2$ (although *Student Action Topic*$_2$ may still be any step). Only the last of these restrictions is required (to limit the size of *Student Action Topic*$_2$'s conditional probability table); the other restriction for CTDT and the only restriction for RTDT are optional for improving response time. These restrictions, when applied, detract from DT Tutor's normative status because (1) the tutor does not always consider tutoring on all topics, and (2) for CTDT, we assume that some steps have a uniformly low probability of being the topic of the student's next action. However, as we discussed, such steps or words are unlikely to be the topic of tutor or student actions, so the restrictions may have little practical effect on applications similar to ours. The restrictions also require DT Tutor to wait to specify some arcs and conditional probability table entries for the next TACN until after the current TACN's student action has been observed, meaning that DT Tutor cannot plan several tutorial actions in advance. However, response time requirements and decreasing marginal benefits of computation currently limit lookahead anyway, so this is not a significant additional limitation for our applications at this time. We intend to improve upon our current naïve network rollup scheme by employing an algorithm for approximate summarization of past dependencies (e.g., Boyen & Koller, 1998).

As a result of these efforts, DT Tutor is space-efficient enough to load on common personal computers (our original research with CTDT used a 200-MHz Pentium PC with 64-Mb of RAM). Response time on faster PCs is already reasonable, and sometimes quite fast, for many of the problems faced by our two action selection engines for real-world tutoring domains. Response time for CTDT has already dropped by an order of magnitude (compare to Murray & VanLehn, 2000), and speedups using both hardware- and software-based solutions are currently practicable. Furthermore, we can realistically expect faster hardware to be commonplace in the near future, and perhaps more efficient software as well.

Evaluating Action Selection Capabilities

By sampling from the space of tutoring scenarios using action selection engines for two domains, we determined that DT Tutor can rationally select tutorial actions that emulate some of the interesting behaviors of human and other tutors.

More recently, we have created a Java-based graphical user interface for CTDT that is accessible over the web using popular browsers. To empirically determine prior and conditional probabilities, we have collected log data from 60 students solving problems using a version of the interface that randomly selects from among appropriate tutorial actions. Random action selection was employed so that we can calculate the effects of *individual* tutorial actions while controlling for the effects of *sequences* of tutorial actions by randomizing over the sequences of tutorial actions that were presented to students. We also administered pretests and post tests of key skills required by CTDT. This data will enable us to begin to empirically determine many of CTDT's prior and conditional probabilities. Next, we plan to have human judges rate the tutorial actions selected for identical situations by DT Tutor, random tutorial action selection, and a simulation of the actions that would be selected by a model tracing tutor (e.g., Anderson et al., 1995; Koedinger et al., 1997)

Conclusions

This research has shown that a decision-theoretic approach can be used to select rational tutorial actions, given the tutor's beliefs and objectives, for real-world-sized problems in reasonable response time. The dynamic, decision-theoretic representation handles uncertainty about the student in a theoretically rigorous manner, balances tradeoffs among any number of objectives, models the evolution of the tutorial state, and automatically adapts to changes in beliefs or objectives. Rich models of the tutorial state enable decision-theoretic action selection engines to select correspondingly interesting tutorial actions.

ACKNOWLEDGEMENTS

This research was supported by the Cognitive Science Division of the Office of Naval Research under grant number N00014-98-1-0467, by an Andrew Mellon Predoctoral Fellowship at the University of Pittsburgh, and by the National Science Foundation under IERI Grant Number REC-9979894. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or the official policies, either expressed or implied, of the sponsors or of the United States Government. Our decision-theoretic inference is based on the SMILE reasoning engine for graphical probabilistic models contributed to the community by the Decision Systems Laboratory of the University of Pittsburgh (http://www.sis.pitt.edu/~dsl). We thank former members of the Andes, Olae and Cascade groups at the University of Pittsburgh for ideas upon which this project is based, and the Project LISTEN group at Carnegie Mellon University for the opportunity to work with the Reading Tutor.

REFERENCES

Albrecht, D.W., Zukerman, I., & Nicholson, A.E. (1998). Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8, 5-47.

Anderson, J.R. (1993). *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J.R., Corbett, A.T., Koedinger, K.R., & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *The Journal of the Learning Sciences*, 4(2), 167-207.

Arroyo, I., & Woolf, B.P. (2001). Improving student models by reasoning about cognitive ability, emotions and gender. In M. Bauer, P.J. Gmytrasiewicz & J. Vassileva (Eds.), *User Modeling 2001: 8th International Conference*. Springer-Verlag: Berlin Heidelberg.

Beck, J.E., & Woolf, B.P. (2000). High-level student modeling with machine learning. In G. Gauthier, C. Frasson, & K. VanLehn (eds), *Intelligent Tutoring Systems, 5th International Conference, ITS 2000*, 584-593.

Binder, J., Koller, D., Russell, S.J., & Kanazawa, K. (1997). Adaptive probabilistic networks with hidden variables. *Machine Learning Journal*, 29(2-3), 213-244.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. *Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 33-42.

Burton, R.R., & Brown, J.S. (1982). An investigation of computer coaching for informal learning activities. In D. Sleeman & J. S. Brown (eds.), *Intelligent Tutoring Systems* (pp. 79-98). New York: Academic Press.

Cheng, J., & Druzdzel, M.J. (2000). AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13, 155-188.

Chiu, B.C., & Webb, G.I. (1998). Using decision trees for agent modeling: Improving prediction performance. *User Modeling and User-Adapted Interaction*, 8, 131-152.

Conati, C. (2002). Probabilistic Assessment of User's Emotions in Educational Games. *Journal of Applied Artificial Intelligence*, Special Issue on Merging Cognition and Affect in HCI, 16(7-8), 555-575.

Conati, C., Gertner, A., & VanLehn, K. (2002). Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction*, 12(4), 371-417.

Conati, C., & VanLehn, K. (1996). POLA: A student modeling framework for probabilistic on-line assessment of problem solving performance. *Fifth International Conference on User Modeling (UM96)*, 75-82.

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

Cooper, G. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42, 393-405.

Cooper, G., Horvitz, E., & Heckerman, D. (1989). A method for temporal probabilistic reasoning (Medical Computer Science KSL-88-30). Stanford, CA: Knowledge Systems Laboratory, Stanford University.

Corbett, A.T., & Anderson, J.R. (1992). Student modeling and mastery learning in a computer-based programming tutor. In C. Frasson, G. Gauthier, & G.I. McCalla (eds.), *Intelligent Tutoring Systems, Proceedings of the Second International Conference, ITS '92*, 413-420.

Dagum, P., & Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1), 141-153.

de Vicente, A., & Pain, H. (2002). Informing the detection of the students' motivational state: an empirical study. In S.A. Cerri, G. Gouarderes & F. Paraguacu (eds.), *Sixth International Conference on Intelligent Tutoring Systems, ITS 2002*, 933-943.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3), 142-150.

Dean, T., & Wellman, M.P. (1991). *Planning and Control*. San Mateo, California: Morgan Kaufmann.

del Soldato, T., & du Boulay, B. (1995). Implementation of motivational tactics in tutoring systems. *Journal of Artificial Intelligence in Education*, 6(4), 337-378.

Ganeshan, R., Johnson, W.L., Shaw, E., & Wood, B.P. (2000). Tutoring diagnostic problem solving. In G. Gauthier, C. Frasson & K. VanLehn (eds.), *Intelligent Tutoring Systems, Fifth International Conference, ITS 2000*, 33-42.

Gertner, A., Conati, C., & VanLehn, K. (1998). Procedural help in Andes: Generating hints using a Bayesian network student model. *Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 106-111.

Goleman, D. (1995). *Emotional intelligence: Why it can matter more than IQ*: New York: Bantam.

Großmann-Hutter, B., Jameson, A., & Witttig, F. (1999). Learning Bayesian networks with hidden variables for user modeling. *IJCAI-99 Workshop "Learning About Users"*.

Grosz, B.J., & Sidner, C.L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3), 175-204.

Heckerman, D. (1995). A tutorial on learning Bayesian networks (MSR-TR-95-06): Microsoft Research.

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? *Twelfth Annual Conference on Uncertainty in Artificial Intelligence*, 307-314.

Horvitz, E., & Barry, M. (1995). Display of information for time-critical decision making. *Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, 296-305.

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. *Fourteenth Conference on Uncertainty in Artificial Intelligence*, 256-265.

Horvitz, E., Jacobs, A., & Hovel, D. (1999). Attention-sensitive alerting. *Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, 305-313.

Howard, R.A., & Matheson, J.E. (1984). Influence diagrams. In R. A. Howard & J. E. Matheson (Eds.), *Readings on the Principles and Applications of Decision Analysis* (pp. 721-762). Menlo Park: Strategic Decisions Group.

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

Huber, M.J., Durfee, E.H., & Wellman, M.P. (1994). The automated mapping of plans for plan recognition. *Tenth Annual Conference on Uncertainty in Artificial Intelligence*, 344-351.

Jameson, A. (1996). Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interaction*, 5(3-4), 193-251.

Jameson, A., Grossman-Hutter, B., March, L., Rummer, R., Bohnenberger, T., & Wittig, F. (2001). When actions have consequences: Empirically based decision making for intelligent user interfaces. *Knowledge-Based Systems*, 14, 75-92.

Kahneman, D., Slovic, P., & Tversky, A. (1982). *Judgment under Uncertainty: Heuristics and Biases*. Cambridge: Cambridge University Press.

Keeney, R., & Raiffa, H. (1976). *Decisions with Multiple Objectives*. New York: Wiley.

Koedinger, K.R., Anderson, J.R., Hadley, W.H., & Mark, M.A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8, 30-43.

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). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lin, Y., & Druzdzel, M.J. (1999). Stochastic sampling and search in belief updating algorithms for very large Bayesian networks. *AAAI-1999 Spring Symposium on Search Techniques for Problem Solving Under Uncertainty and Incomplete Information*, 77-82.

Martin, J., & VanLehn, K. (1995). Discrete factor analysis: Learning hidden variables in Bayesian networks. Pittsburgh, PA: University of Pittsburgh.

Mayo, M., & Mitrovic, A. (2001). Optimising ITS behaviour with Bayesian networks and decision theory. *International Journal of Artificial Intelligence in Education*, 12, 124-153.

Merrill, D.C., Reiser, B.J., Merrill, S.K., & Landes, S. (1995). Tutoring: Guided learning by doing. *Cognition and Instruction*, 13(3), 315-372.

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. (2001). 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. (1999). A dynamic, decision-theoretic model of tutorial action selection. MS Thesis, University of Pittsburgh, Pittsburgh, PA. URL: www.isp.pitt.edu/~chas/MS Thesis.htm

Murray, R.C., & VanLehn, K. (2000). DT Tutor: A dynamic, decision-theoretic approach for optimal selection of tutorial actions. In G. Gauthier, C. Frasson & K. VanLehn (eds.), *Intelligent Tutoring Systems, Fifth International Conference, ITS 2000*, 153-162.

Murray, R.C., VanLehn, K., & Mostow, J. (2001). A decision-theoretic architecture for selecting tutorial discourse actions. *AI-ED 2001 Workshop on Tutorial Dialogue Systems*, 35-46.

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

Ngo, L., Haddawy, P., Krieger, R.A., & Helwig, J. (1996). Efficient temporal probabilistic reasoning via context-sensitive model construction. Milwaukee, WI: University of Wisconsin-Milwaukee.

Paek, T., & Horvitz, E. (2000). Conversation as action under uncertainty. *Sixteenth Annual Conference on Uncertainty in Artificial Intelligence*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan-Kaufmann.

Pek, P.-K. (2003). Decision-Theoretic Intelligent Tutoring System. PhD dissertation, National University of Singapore.

Pek, P.-K., & Poh, K.-L. (2000). Using Decision Networks for Adaptive Tutoring. *International Conference on Computers in Education / International Conference on Computer-Assisted Instruction*, 1076-1084.

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

Reye, J. (1996). A belief net backbone for student modeling. In C. Frasson, G. Gauthier & A. Lesgold (eds.), *Intelligent Tutoring Systems, Third International Conference*, 596-604.

Reye, J. (1998). Two-phase updating of student models based on dynamic belief networks. In B. Goettl, H. Halff, C. Redfield & V. Shute (eds.), *Intelligent Tutoring Systems, Fourth International Conference*, 274-283.

Reye, J. (2004). Student Modelling based on Belief Networks. *International Journal of Artificial Intelligence in Education*, 14, 63-96.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, New Jersey: Prentice Hall.

Schäfer, R., & Weyrath, T. (1997). Assessing temporally variable user properties with dynamic Bayesian networks. *User Modeling*: *Proceedings of the Sixth International Conference, UM97*, 377-388.

Shachter, R., & Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks, *Fifth Annual Conference on Uncertainty in Artificial Intelligence* (Vol. 5, pp. 221-231). Mountain View, CA: Association for Uncertainty in AI.

Shachter, R., & Peot, M. (1992). Decision making using probabilistic inference methods. *Eighth Annual Conference on Uncertainty in Artificial Intelligence*, 276-283.

Shute, V.J. (1995). SMART evaluation: cognitive diagnosis, mastery learning & remediation. *Artificial Intelligence in Education, 1995 (AI-ED 95)*, 123-130.

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., Ball, W., & Kowalski, B. (1989). Non-LIFO execution of cognitive procedures. *Cognitive Science*, 13, 415-465.

VanLehn, K., & Niu, Z. (2001). Bayesian student modeling, user interfaces and feedback: A sensitivity analysis. *International Journal of Artificial Intelligence in Education*, 12(2), 154-184.

VanLehn, K., Siler, S., Murray, C., Yamauchi, T., & Baggett, W.B. (2003). Why do only some events cause learning during human tutoring? *Cognition and Instruction*, 21(3), 209-249.

Walker, M.A. (1996). Limited attention and discourse structure. *Computational Linguistics*, 22(2), 255-264.