CHAPTER 5 –A Review of Student Models Used in Intelligent Tutoring Systems

Philip I. Pavlik Jr.[1], Keith Brawner[2], Andrew Olney[1], and Antonija Mitrovic[3]
[1]University of Memphis, [2]U.S. Army Research Laboratory, [3]University of Canterbury - New Zealand

Introduction

Generally, an ITS is considered to have four major components (Elsom-Cook, 1993; Graesser, Conley & Olney, 2012; Nkambou, Mizoguchi & Bourdeau, 2010; Psotka, Massey & Mutter, 1988; Sleeman & Brown, 1982; VanLehn, 2006; Woolf, 2008): the domain model, the student model, the tutoring model, and the tutor-student interface model.

- The *domain model* contains the set of skills, knowledge, and strategies of the topic being tutored. It normally contains the ideal expert knowledge and may also contain the bugs, mal-rules, and misconceptions that students periodically exhibit. It is a representation of all the *possible* student states in the domain. While these states are typically tied to content, general psychological states (e.g., boredom, persistence) may also be included, since such states are relevant for a full understanding of possible pedagogy within the domain.

- The *student model* consists of the cognitive, affective, motivational, and other psychological states that are inferred from performance data during the course of learning. Typically, these states are summary information about the student that will subsequently be used for pedagogical decision making. The student model is often viewed as a subset of the domain model, which changes over the course of tutoring. For example, "knowledge tracing" tracks the student's progress from problem to problem and builds a profile of strengths and weaknesses relative to the domain model (Anderson, Corbett, Koedinger & Pelletier, 1995). Since ITS domain models may track general psychological states, student models may also represent these general states of the student.

- The *pedagogical model* takes the domain and student models as input and selects tutoring strategies, steps, and actions on what the tutor should do next in the exchange with the student to move the student state to more optimal states in the domain. In mixed-initiative systems, the students may also initiate actions, ask questions, or request help (Aleven, McLaren, Roll & Koedinger, 2006; Rus & Graesser, 2009), but the ITS always needs to be ready to decide "what to do next" at any point and this is determined by a tutoring model that captures the researchers' pedagogical theories. Sometimes what to do next implies waiting for the student to respond.

- The *tutor-student interface model* interprets the student's contributions through various input media (speech, typing, clicking) and produces output in different media (text, diagrams, animations, agents). In addition to the conventional human-computer interface features, some recent systems have had natural language interaction (Graesser, D'Mello, et al., 2012; Johnson & Valente, 2008), speech recognition (D'Mello, Graesser & King, 2010; Litman, 2013), and the sensing of student emotions (Baker, D'Mello, Rodrigo & Graesser, 2010; D'Mello & Graesser, 2010; Goldberg, Sottilare, Brawner & Holden, 2011).

This review focuses on the approaches to student modeling, with particular attention to the implications of these models for design of GIFT. GIFT similarly adopts this four-part distinction, but refers to the models as modules, since GIFT is an actual software framework that reifies each of these models with modules in

the software. GIFT also includes an optional sensor module, as a way to measure user states during interactions with the tutoring system.

This chapter specifically focuses on the student model aspect of the ITS. In order to give a clear perspective on the student model, it is essential to also address the domain, pedagogical, and interface models because the models do not function in isolation. The connections between domain and student models are tightly coupled. Less tightly coupled, but still substantial, is the dependence of the pedagogical model on the student model. In contrast, the tutor-student interface model, while it clearly relates to the domain and pedagogical models, is beyond the scope of this article. The structure of the domain model may be necessary to consider in the context of student modeling, and to some small degree, the pedagogical model. However, this chapter does not describe the complications of mapping domains to interfaces or describing how different interfaces reify similar pedagogy. We leave interface issues to subsequent volumes of this book series.

Assessing a Student Model - Criteria

When assessing models, it is appropriate to emphasize the limitations of complexity and fidelity of representation. Consequently, all models of learning are wrong in certain ways because they cannot hope to capture the full complexity of the real student's mind. In the GIFT system, we want the designer to implement the most useful student models, not necessarily the most correct student models. For example, we might suppose that a multilevel neural network, such as the Leabra architecture (O'Reilly, 1998), would be a more correct representation of a student. However, it would not be very useful because it would be monumentally complex and impractical to use it for clear pedagogical inference and tracking of the domain model. We therefore evaluate the student model options in regard to several criteria of usefulness. These criteria include the following:

- *Student model fit to data*. This is a simple validity criterion that refers to how well the student model can be used to simulate the quantitative and/or qualitative patterns of learning in real students. Issues of model fit have been reviewed by Desmarais and Baker (2012) and more generally by Schunn and Wallach (2005).

- *Ease of understanding*. The reality behind educational system design is that student modeling methods need to be comprehensible to system designers and builders. Therefore, student modeling techniques that require optimizing a Bayesian network using simulated annealing may have limited applicability unless such complexity is easily approached by system designers. This limitation of complex models parallels the tendency for complexity of student models in running systems to lag behind the complexity of student modeling techniques presented in research. This explains why existing systems are often less complex from systems invented 60 years ago (Smallwood, 1962).

- *Generality/flexibility*. Many of the student models we review have only one or two primary successful contexts, so some student models have limited generality. Generality may have a downside in that more general models tend to be simpler, and therefore, lose power in specific domains as they gain generality. To what extent can student models be reused in new contexts to improve the adaptation to students? To what extent is a student model essentially chained to specific domain content, so that after this domain content is learned, the student model is no longer relevant?

- *Cost of creation*. Research often cites the high costs of creating content (Aleven, McLaren, Sewall & Koedinger, 2006), and such costs also apply to student models. If a student model

requires 2 years of data collection to determine the parameters, the cost may be prohibitive. The issue of cost refers to person hours of production, while the issue of ease of construction involves how skilled (professional degree) those workers need to be.

- *Granularity*. Grain-size may refer to steps, single problems, single curriculum units, multiple curriculum units, whole domains, or multiple domains. Student models are typically applied to pedagogical decisions at one or more levels of granularity. For example, a student model might trace the state of the student within the steps of a problem, or how that state changes across problems, units, or even different domains. One hypothesis we make in this chapter is that different models may be more or less useful depending on the granularity of the pedagogical decisions.

- *Time scale*. Time scale refers to the overall longevity of the student model.. A model of student state at the step level might be created from only the current problem interaction, or it may take into account prior actions in prior problems or earlier units. The time scale helps determine whether an ITS system is actually a collection of independent units, or whether the student model is cumulative across the domain as the student progresses.

- *Learning gains in practice*. Evidence of learning gains that are directly caused by pedagogical inferences from the student model states is obviously an excellent feature for a student model. However, it is very easy to argue that learning gains depend on far more than the student model, since the pedagogical model and tutor-student interface feedback need to be appropriate for learning gains. Furthermore, comparing learning gains carefully across different control conditions, domains, research groups, and populations is difficult if not impossible to do in a valid way.

To make the assessment according to these criteria requires us to group the student models into a manageable number of categories whose members share deep similarities. One can make finer distinctions in the categories we review, but these finer distinctions may not move us toward our goal of making recommendations for GIFT. Finer distinctions do not highlight the overarching software architectural issues involved in providing support for broad range of very different student models.

Programmed Student Models

Our first category has a long history that traces back first to Pressey in 1926, who is credited with the first "teaching machine," which was little more than a question and answer device with multiple choice scoring and immediate feedback (Lumsdaine & Glaser, 1960). This initial start in the ‘‘teaching machine" movement gained little attention until the 1950 when an explosion of automated teaching theories began to emerge from various sources. Most notable was work by Skinner looking at linear programming of instruction (not to be confused with the unrelated mathematical optimization procedure) and Crowder's work with branching programs (Thomas, Davies, Openshaw & Bird, 1963).

Skinner's contribution came from his theory of behaviorism and the concept of error-free learning. While a teaching machine with a linear program does provide feedback, the key to linear programming was to provide a sequence of tasks where the sequence faultlessly provides the prerequisites in a serial order such that a prepared student might be expected to proceed through the content without errors. To accomplish this goal, detailed sequences of rule and exemplar practices are composed by the linear program designer. These sequences can be derived from a task analysis that uses a rule by rule matrix to determine associated rules as well as rules that need to be discriminated. From this rule matrix, the linear programmer then lays out the faultless sequence, which is composed of several types of cloze (fill in the

blank items) including examples, rules, generalizations, and discriminations in the domain (Lumsdaine & Glaser, 1960; Thomas et al., 1963). Frequent reinforcement of prior learning was built into such sequences.

One clear issue with linear programming is that it mandates that a single sequence is optimal for all students. Crowder challenged this assumption with similar systems that branched extensively based on the student responses, thus providing a basic means of individualizing instruction. Based upon student actions, a student may be "branched" to a review, remediation, or additional content (Crowder, 1959; Lumsdaine & Glaser, 1960; Thomas et al., 1963).

This category of student models as part of the discussion of ITSs raises the question of what we should be counting as a student model. In other words, how smart does an "intelligent" tutoring system need to be to qualify as a true ITS. While this question may have philosophical interest in that it deals with the general question of what is intelligence, it does not seem to be a pragmatically helpful criterion. More sensibly, we might ask how adaptive a system is by characterizing how much freedom the domain model allows in measuring student states to infer one of many pedagogical choices. Typically, branched programming student models offer less adaptivity because student states are hardcoded as triggers for specific pedagogies since the development process for branched models handcrafts each state branch in the student model space. Perhaps these systems do not qualify as ITSs, considering the student state space is small and that pedagogical options are few.

Prior efforts such as this are particularly relevant since they help us see that new methods, such as knowledge space theory (discussed further in the chapter), can be traced to antecedent methods of much older pedigree. Indeed, sometimes these methods come back in similar form to their prior versions. For example, work with "example tracing tutors" at the Pittsburgh Science of Learning Center (PSLC) seems to involve a very advanced method for creating branching tutors by mapping out the possible "example" paths a student may take in solving a problem. They argue that this work should be considered an ITS because "example-tracing tutors are capable of sophisticated tutoring behaviors; they provide step-by-step guidance on complex problems while recognizing multiple student strategies and (where needed) maintaining multiple interpretations of student behavior." (Aleven, McLaren, Sewall & Koedinger, 2009)

The main strength of programming methods is their simplicity of execution. Linear programming is the perhaps the most simple, but arguably so simple that it misses the benefit of the many important pedagogical moves that depend on attending to individual differences. Branched programming address this shortcoming and is made easier when supported by technologies that allow branched program authoring (Aleven et al., 2009). Where branched programming begins to break down is when branching rules for addressing different student model states with different pedagogies become very sophisticated. In this case, the fit of the branched programming model to student states becomes difficult to measure, since it is idiosyncratic. Subsequent generality will then usually be very low. So, if a general system is desired it makes sense to implement adaptation (pedagogy) using methods other than hard coding the student states and their resulting pedagogy as branches in the domain model. We discuss many of these alternative means for adaptation in the next sections.

Student models that consist of branching states now have a more than 50-year history. Despite their shortcomings, they should be included in GIFT. Branched programming provides considerable power and efficiency to a developer with a simple project, perhaps explaining why SCORM includes some simple criterion based branching capability, with new proposals in the SCORM domain advocating deeper personalization through more complex rule based branching (Rey-López, Fernández-Vilas, Díaz-Redondo, Pazos-Arias & Bermejo-Muõz, 2006). However, the main advantage for robust support of branching in GIFT may be in the potential ease of construction aspect. If GIFT provides a powerful and easy mechanism to create adaptive branching tutors (perhaps by allowing the designer to graphically

author possible student moves in the state space), there is the potential to attract a wider user base, with consequent wider impact on the needs of the education and training community.

Overlay Student Models

The origin of overlay models is shrouded in history that reaches back to the heyday of behavioristic thought in the late 50s and earlier. During these early years of educational software, designers were still heavily influenced by the idea that speculating on "skills" and other unseen constructs was bad science, as had been taught by Skinner. Fortunately however, some were beginning to speculate on deeper adaptivity in "teaching machines." In particular, Smallwood (1962) should probably receive credit for his work in setting the stage for concept overlay models, Bayesian knowledge tracing (BKT), and economic optimization of practice. To illustrate the groundbreaking nature of Smallwood (1962), it is useful to quote a few points from the monograph where he proposes this structure (parenthetical comments added):

1. *The decomposition of the subject matter into a set of concepts that the educator would like to teach to the student*. (Part of the domain model.)

2. *A set of test questions, for each concept, that adequately tests the student's understanding of the concept*. (Part of the domain model.)

3. *An array of information blocks, for each concept that can be presented to the student in some order (to be decided by the teaching machine)- and thus provide a course of instruction to the student on the concept*. (Part of the domain model.)

4. *A model that can be used to estimate the probability that a given student with a particular past history will respond to a given block or test question with a particular answer*. (The student model.)

5. *A decision criterion upon which to base the decisions mentioned in 3*. (The pedagogical model.)

While Smallwood continued his explorations of optimal instructional policies (1971), Richard Atkinson, a Stanford professor of psychology, who would later be nominated to head the NSF from 1975–1980 and later head the University of California System, was busy over the 1960s testing a variety of Markov models that captured learning in probabilistic state transition networks (Atkinson & Crothers, 1964; Calfee & Atkinson, 1965; Groen & Atkinson, 1966). Atkinson endorsed many of Smallwood's ideas, but perhaps described them in a more accessible form in his "theory of instruction," which he hoped to formalize in computerized instructional systems (Atkinson, 1972a). Furthering these goals, he produced an excellent demonstration of the utility of a single skill per item Bayesian overlay model, which he used to optimize the learning of German vocabulary items. The strong results supported a student model, which individualized instruction by using different parameters for each vocabulary item to control practice sequencing, obtaining 79% performance at a 1-week test, compared to 38% performance for a random sequence of practice (Atkinson, 1972b).

Work such as Atkinson's, where there is a relatively complex mathematical model that tracks a collection of independent item's correctness (the simplest overlay), has been further exemplified by work with the optimization of Japanese-English language pairs (Pavlik Jr. & Anderson, 2008) by taking account of time costs in the model so as to make pedagogical decisions not just in reference to learning, but also in reference to the time costs of this learning. Pavlik's technology has been implemented in the X-Germs series of arithmetic games from K-12, Inc. This work is arguably unique because of the complexity of the mathematical model that captures several principles of memory and treats all the learned units as

independent declarative chunks. The mathematical formalism is based on ACT-R's declarative memory system and involves using the full history of practice to compute when each item is at the temporal sweet spot of optimal learning. Optimality is achieved by maximizing the expected long-term learning gains against the current cost of additional practice.

Even the early work on overlay models makes clear that an overlay student model tracks some pedagogically relevant quantity as a function of the overlay type (in this case, skills, facts, or concepts). Modern work continues to follow this pattern despite at least three different kinds of overlay models that went beyond this early work with learning sets of independent items. These methods include rule space methods, model tracing, and constraint-based models. To add further complexity, however, each of these methods has a variety of ways in which knowledge strength or probability is traced. To simplify the issue, we associate each of the three methods with its most common long-term model tracing formalisms (e.g., BKT), but keep in mind that the overlay model is loosely coupled to the equation that tracks the strength of overlaid model states. In other words, BKT could be used in a rule space context, just as logistic regression could be used to trace the strength of model traced productions.

Rule Space Student Models

An important early student model was the rule space method, which was introduced as a modification of Item Response Theory (IRT) to provide a way to categorize misconceptions in signed number arithmetic problem solving data (Tatsuoka, 1983). This original approach has stimulated successively more advanced efforts to develop the IRT approach as a student modeling method. Perhaps most important of these efforts was the development of the idea that the count of prior learning attempts can be used to predict later performance. This model has been developed by various authors and is often referred to as the additive factors model (AFM; Draney, Pirolli & Wilson, 1995; Spada & McGraw, 1985).

More recently, researchers have come to refer to the rule space mapping as a Q-matrix, which stands for "question matrix" to capture how it maps each of the questions to a number of proficiencies or misconceptions (Barnes, 2005; Barnes, Stamper & Madhyastha, 2006; Pavlik Jr., Yudelson & Koedinger, 2011). Typically, a Q-matrix is a binary matrix that maps specific skills/ rules to particular questions or types of questions. A main point of this section on overlay models is that the Q-matrix representation could be considered a standard way to represent the domain model semantics. Once a Q-matrix has been determined for some content domain, the main question becomes what mathematical formalism (typically, Bayesian-Markov process models or logistic regression) should be used to represent the student states relative to the possibilities the Q-matrix admits. The point of determining the Q-matrix with parameterized equations is that this allows the ITS to mathematically infer an order of problems that allows learning to be meticulously adapted based on prior knowledge with the goal of diagnosing and providing instruction on the "rules" of the domain, as represented in the matrix.

Q-matrix models with logistic regression quantitative tracking have become part of the modeling apparatus the PSLC (DataShop) though they have not been employed in any running systems we are aware of because the model is not adaptive (AFM, Draney et al., 1995; Spada & McGraw, 1985). However, recent work to improve the fit of the additive logistic regression models has shown the importance of capturing success and failure in this type of model, demonstrating that logistic regression can be at least as accurate as the standard BKT (Corbett & Anderson, 1992; Gong, Beck & Heffernan, 2010; Pavlik Jr., Cen & Koedinger, 2009). In this performance factors analysis (PFA) model (an AFM logistic regression variant), there are two fundamental categories of prior practice, namely, success and failure (Pavlik Jr. et al., 2009), which contribute differently to future performance, unlike in the AFM model where merely the count of prior experience is tracked. The better fit of the PFA model is sensible, since the psychological literature shows successes (in contrast to review after failing) may lead to more

production-based learning and/or less forgetting (Carrier & Pashler, 1992; Karpicke & Roediger, 2008; Pavlik Jr., 2007; Thompson, Wenger & Bartling, 1978) and may lead to automaticity for shallow learning tasks (Peterson, 1965; Rickard, 1997, 1999; Segalowitz & Segalowitz, 1993). Moreover, the logic of representing additional factors of learning easily in the logistic regression model is resulting in further model variants that capture a variety of instructional differences between different student interactions (M. Chi, Koedinger, Gordon, Jordan & VanLehn, 2011) and transfer between related items (Pavlik Jr. et al., 2011). These models may have the advantage of incorporating multi-factor complexity more easily than BKT models. In general, this analytic approach attempts to decompose practice contexts to understand what specific features of the practice can be useful for predicting future performance (Beck & Mostow, 2008). Typically, such work is theory driven.

Model Tracing Student Models

Model tracing comes from Anderson's work on the LISP programming tutor at Carnegie Mellon University (CMU), which used what he called a variant of Atkinson's (1972b) work (Anderson, Conrad & Corbett, 1989). This approach created a student model that both quantitatively traced the learning of production rules in a LISP programming language tutoring system at the same time as it traced the students' responses. Production rules are low-level cognitive steps in a problem solution encoded as IF-THEN rules. For example:

> IF the goal is to prove ΔABC I congruent to ΔDBE and AE and CD are collinear
>
> THEN infer ∠ABC is congruent to ∠DBE because they are vertical angles

Reactivity to user responses was engineered by a mechanism called model tracing. Model tracing has developed from several implications of the ACT theory. These principles include (1) encoding the student model as a set of production rules, (2) communicating these rules through problem-solving exercises, (3) maximizing the learning rate by responding to the quantitative measurement of learning, and (4) providing help for failure on the good answer rather than explaining bad answers (Anderson et al., 1995). When a tutor is in model tracing mode, there are three possible outcomes as the tutor tries to interpret each student action or sequence of actions:

1. The student's action matches one of the production rules from the domain model. (This will later result in an increment to strength in the quantitative model.)

2. Multiple sequences of production match, triggering a disambiguation question. (The disambiguation question would then control what productions are learned.)

3. There is no interpretation of the error. This may result in matching of a buggy production that leads to context specific messages for the student. If no buggy rule is matched, the student must continue to respond until a production is matched. Since this can cause an impasse, the system typically involves backup hints of greater and greater complexity to scaffold the student who is producing unmatchable responses.

There is a distinction between this model tracing process of interpreting behavior and the knowledge tracing process of inferring the growth of knowledge across the sequence of practice. Corbett later presented a canonical paper on what has become known as BKT, which included detailed modeling of individual student differences as well as the core Markov model knowledge tracing formalism (Corbett & Anderson, 1995). This augmented overlay model was used to track the student's progress from problem to problem by building a profile of strengths and weaknesses relative to the production rules (Anderson et al., 1995), rather than in relation to concepts or facts which had been tracked in previous domain overlay

student tracing models. Step-by-step BKT is incorporated in a number of tutors in the PSLC (Aleven, McLaren, Roll, et al., 2006; Anderson et al., 1995; Heffernan, Koedinger & Razzaq, 2008; Ritter, Anderson, Koedinger & Corbett, 2007; VanLehn, 2006) and is a method used heavily by Carnegie Learning, Inc., to track skill progression in their various systems. Interestingly, however, many of the running systems do not realize the full complexity of Corbett's original models of individual differences (Corbett & Anderson, 1995).

A primary weakness of this type of approach for domains with many skills or misconceptions is that the skills and misconceptions must be reasonably enumerated in order to provide feedback (Brown & VanLehn, 1980). The creation of this domain model can be time consuming. Although development time is not widely reported, general ITS system design is estimated at 200–300 hours of development time per 1 hour of instructional content (Aleven, McLaren, Sewall, et al., 2006). Additionally, there is evidence that the remediation of specific bugs may not have added instructional benefit over the alternative of simply re-teaching the material that contained misconceptions (Sleeman, Ward, Kelly, Martinak & Moore, 1991). There is potential to mitigate this weakness with recent efforts to focus misconception database minimization, keeping the conceptual misconceptions to fewer than five in order to save development time (VanLehn et al., 2007). Sometimes, for well-defined domains with substantial prior data, the misconception or skill database may be automatically generated, saving a significant amount of development time (Burton, 1982; Cen, Koedinger & Junker, 2006). The GIFT domain module currently supports the communication of this information, but not the construction. Possibly, this type of knowledge component overlay could be accomplished automatically from the relationship and interactions inherent in textual information (Ahmed, Toumouh & Malki, 2012; Lahti, 2010). Practically, this type of knowledge overlay should either be supported via automatic generation, the merging of existing model structure, or a tool for authoring.

Perhaps more technically problematic than creating some list of skills is to describe in a meaningful way how the skills interact in a way that is computationally tractable and not overly simplistic. In a highly interactive system, the pedagogical model would be able to select items for practice that accounted for prerequisite relations automatically. We might want a model where we can make inferences about the item that is not yet mastered, but yet still the easiest item-type that the student might practice. For example, in a unit on fraction skills, we might want the system to detect (based on a history of success and failure) what basic skills a student has mastered (e.g., least common multiples) and avoid these, identify the skills students are ready for (e.g., equivalent fractions), and similarly hold in reserve those items that still have unmastered prerequisites. A related issue is when there are multiple skills needed to produce a single response. In such cases, whether we model the relationship of the skills as additive (one skill can compensate for others) or conjunctive (all skills must succeed to succeed in the response) has implications for parameter estimation and what skills to credit in the case of success or failure. However, in practice, it has been complicated to work out how skills should be combined (Ayers & Junker, 2006; Cen, Koedinger & Junker, 2008; Koedinger, Pavlik Jr., Stamper, Nixon & Ritter, 2011).

Constraint-Based Student Models

In constraint-based tutors, the domain model consists of a set of constraints representing the basic domain principles (Ohlsson, 1992). Each constraint is a declarative statement composed of a relevance condition (R) and a satisfaction condition (S). The relevance condition specifies when the constraint is relevant and only in these conditions is the constraint meaningful. The satisfaction condition specifies additional conditions that must be satisfied by the student's solution for which the constraint is relevant in order for the solution to be correct. A satisfied constraint corresponds to an aspect of the solution that is correct. A violated constraint indicates a mistake in the solution without explicitly representing the actual misconception the student might have; it simply means that the student's solution violates a domain

principle. Therefore a constraint set explicitly defines the space of correct knowledge, without requiring misconceptions to be identified and collected; the space of incorrect knowledge is represented implicitly by violated constraints (Ohlsson & Mitrovic, 2007).

An example constraint from the cooking domain might be "when making French fries, the oil must be hot before potatoes are added." The relevance condition of this constraint specifies the task (making French fries) and the current state of the solution (the student has put potatoes into the pan). The satisfaction condition then specifies that the temperature of the oil must be within the appropriate range. This constraint will be violated by various incorrect actions (like there is no oil in the pan, the oil is too hot or too cold, etc). However, the tutor only needs to teach the domain principle (the required temperature of the oil) to the student, rather than having to identify the exact misconception that caused the mistake.

One example constraint for the task of fraction addition might be relevant in situations when the student has added two fractions, a/b + c/d, and the student's solution is of the form (a+c)/n (Ohlsson, 1992). The satisfaction condition of the constraint states that b=d=n; only in that case is the student's solution correct. This constraint can be violated because of many misconceptions; however, the tutor can simply reinstate the violated domain principle (the numerators of the two given fractions can be added if their denominators are equal).

Constraints can be syntactic or semantic in nature (Mitrovic & Ohlsson, 1999). Syntax constraints are simpler than semantic ones, which need to ensure that the student solution is an appropriate solution for the given problem. In domains where there are multiple correct solutions, the required properties of the solution are captured in terms of a pre-specified (ideal) solution; the semantic constraints compare the student's solution to the ideal solution taking into account alternative ways of solving the same problem, in terms of equivalent domain operators (Mitrovic, 2012).

Constraints specify what ought to be so, so they are evaluative in nature. They are not prescriptive in the sense of generating behavior like production rules do. Therefore, a student's solution is diagnosed by matching it to the constraint set, identifying the relevant constraints, followed by checking the satisfaction conditions of the relevant constraints. This process corresponds to what VanLehn (2006) calls the inner loop. Information about satisfied and violated constraints is then used by the tutor to generate positive and negative feedback for the student (Mitrovic, Ohlsson & Barrow, 2013; Zakharov, Mitrovic & Ohlsson, 2005).

The previous two example constraints demonstrate that constraint-based modeling (CBM) does not presume any particular stepwise pedagogical approach. Instead, it is used to react to every action the student makes (to provide immediate feedback) or wait until the solution is complete and thereby provide delayed feedback. Usually feedback is provided on demand, when the student requires it, but can also be provided at times when the pedagogical model identifies that the student would benefit from the feedback. The granularity of constraints can also vary, and should be determined from the pedagogical point of view, by designing feedback that is effective for learning a particular task.

Student models in constraint-based tutors are also overlays. That is, the student's knowledge is represented in terms of constraints that the student does or does not know, as demonstrated by the submitted solutions. Each time the student's solution is matched to the constraints, information about violated and satisfied constraints is used to update the student model. The simplest way to represent the student's knowledge of a particular constraint is a simple frequency of correct use within a specified window of opportunities to use the constraint. Some constraint-based tutors have probabilistic student models, where information about the violated and satisfied constraints is used to update the probability of the student knowing each constraint (Mayo, Mitrovic & McKenzie, 2000).

This overlay student model is used by the pedagogical module to select problems of appropriate complexity for the student. Problem selection could be based on simple heuristics, such as the one which identifies the constraint(s) with which the student had the most difficulty (Mitrovic, 2003). Alternatively, in one version of SQL-Tutor, there is a simple Bayesian network for each problem, which makes predictions about the performance of a particular student on the problem. These multiple predictions are then combined to give an overall measure of the value of the problem for a particular student (Mayo et al., 2000). CAPIT, an ITS that teaches children about punctuation and capitalization rules in English, uses a data-centric methodology, in which the structure of the Bayesian network is induced from the student data (Mayo & Mitrovic, 2001). The Bayesian model predicts the student's performance on each problem, and utility functions are used to select the best problem for the student and also the best feedback to be given to the student if there are errors. The classroom evaluation shows that such a decision-theoretic pedagogical strategy results in reduced learning times and improved performance.

Early research on constraint-based tutors focused on a range of domains to which CBM is applicable. Successful constraint-based tutors cover a wide variety of instructional domains, ranging from well-defined to ill-defined design tasks (Mitrovic & Weerasinghe, 2009). Well-defined tasks have crisp rules that govern them and often have incremental performance steps, while ill-defined tasks frequently have performance described in terms of outcomes. Some examples from the latter category are the SQL-Tutor (Mitrovic, 1998) that teaches database querying using Structured Query Language (SQL) and EER-Tutor that teaches database design using the enhanced entity-relationship (EER) model (Mitrovic, Martin & Suraweera, 2007). Learning gains have been between 0.6 and 1.6 sigma (Amalathas, Mitrovic & Ravan, 2012; Mitrovic, 2012; Suraweera & Mitrovic, 2004) after short learning sessions (less than 2 hours). Later work includes using constraints to represent not only domain knowledge, but also collaborative skills (Baghaei, Mitrovic & Irwin, 2007), to support tutorial dialogues (Weerasinghe, Mitrovic & Martin, 2009), self-assessment and open student models (Mitrovic & Martin, 2007).

The CBM paradigm, like the other types of student models in this work, can be used to represent domain knowledge. It has shown strength in its ability to instruct, but the largest limitation remains that that the models must still be authored. However, there is evidence that the authoring of constraint-based tutors is more rapid than their traditional counterparts (Mitrovic, Koedinger & Martin, 2003). These development times have come down rapidly through the introduction of authoring tools: from 220:1 (Mitrovic & Ohlsson, 1999) down to 30:1 (Heffernan, Turner, Lourenco, Macasek & Nuzzo-Jones). In addition, ASPIRE is an authoring system for constraint-based tutors that is freely available (Mitrovic et al., 2009).

CBM is an attractive solution to the student knowledge modeling problem for several reasons. Firstly, CBM does not require the large amounts of development time traditionally associated with production rules. It does not require probabilistic estimates of student knowledge built upon a large database of previous interactions, so it is a practical approach for instruction. Secondly, CBM supports both procedural and open-ended task modeling because it does not have to exhaustively model task dynamics (Mitrovic et al., 2003). Finally, CBM systems have been shown to be effective on a wide variety of instructional tasks (Mitrovic, 2012).

Yet another advantage of CBM is that the created models of domain knowledge have the potential to be transitioned to another system that embodies the same concepts. This is neatly aligned with the principles of architecture creation. The method of model creation thereby has the capability to transfer. The lifespan of the model is tied to the lifespan of the ITS, although fractional components of the model may be used in the construction of a model for another system. The ability to cannibalize model components and model construction components into a new ITS is a highly desirable architectural enhancement.

At the time of writing, GIFT supports the communication of student state information through a hierarchy of "concepts" and "sub-concepts," which have varying levels of grading. This fits well with the student

modeling nature of evaluating the student model through mastered versus violated constraints. GIFT currently makes an effort to describe a student model using a concept breakdown, where each concept is not tied to a given domain of instruction, and is assessed simply (e.g., above standard, at standard, below standard, or unknown); however, it does not currently support the seamless authoring of module components which send these messages. The transfer of these tools and knowledge is the prime advantage of the CBM approach, but future development should support the plugin of the existing authoring tools to create CBMs for tracking student performance models.

Overall Considerations Underlying Overlay Models

Overlay models have two architectural components: the overlay objects and the mathematical tracking formalism. The overlay objects are typically linked to inner loop immediate pedagogy provided through the tutor's interface, e.g., immediate feedback when a constraint is violated, while mathematical formalisms are used by the outer loop to select which problem items are presented to students. Therefore, general systems like GIFT will benefit from providing explicit support for defining overlay objects that are linked both to tutor interface pedagogy and data recording that will enable the outer loop mathematical model to compute the next best problem to present at any time. While overlay models can function at one level or the other (e.g., with no immediate pedagogy or with no mathematical tracking), most modern overlay systems include both immediate pedagogy in response to violations of the assumption of the overlay objects and mathematical tracking of the overlay objects.

The strengths of the quantitative models underlying the overlays are not frequently realized in running systems because they typically do not live up to the goals of economic optimization and student-level individual difference tracking inherent in their configuration. Economic optimization takes account of the costs of each possible pedagogical action in addition to the gains (Atkinson, 1972a; Smallwood, 1962, 1971). While efficiency tends to be ignored, there has been renewed attention recently, with some researchers exploring how knowledge tracing models can be used to detect when skills are being over practiced, and instead spend that practice time on items that need practice. Reducing such over practice in existing systems has led to reduction in learning time, with no difference in gains (Cen, Koedinger & Junker, 2007). This failure to address costs in learner models is similar to the general failure to address student-level parameters in student modeling. Student-level parameters can be used for modeling individual differences, rather than making the assumption that there is a global, generalized model that fits all students. For example, despite work by Corbett showing that student-level parameters are important to produce tight fits to students' data (Corbett & Anderson, 1995), systems using BKT tend not to use subject level parameters, e.g., Carnegie Learning, Inc., tutors do not use general parameters for each student (Ritter & Anderson, 2006).

Because of the wide usage of component overlay models combined with mathematical tracing of components, GIFT will likely need to provide some support of this sort of student model. To start with, this implies support for a domain representation such as a Q-matrix, which assigns skills to each exercise in the domain. These Q-matrices may be universally needed for almost all overlay model variants. Secondly, GIFT needs to provide data structures to fully reload the students' prior history within a system to enable reconstruction of the user model. Third, GIFT should provide an API for mathematical functions that compute pedagogically relevant quantities using the history for the student. For example, in the Fact and Concept Training system, the mathematical model includes a function "choose-trial," which analyses the full history for all the items for the student to determine the item that is closest to being at the optimal point for rehearsal. In turn, this choose-trial function (called from the interface) depends on lower level model computations of the probability of successful performance and time costs for each item. Similarly, after practice, a model API function saves the result of prior practice as the history accumulates (Pavlik Jr. et al., 2007).

We advocate a three-pronged recommendation that the domain structure should be compatible with a Q-matrix structure (Domain Module), able to reload the history of practice so as to compute the current student model state (Student Module), and an interface for interpreting mathematical states of the student model to make pedagogical inference in a tractable way (Pedagogical Module). If a basic model API that includes some default models, GIFT users will be able to test and modify the mathematical component of the overlay models quickly and easily. Additionally, by providing a standard Q-matrix support in GIFT for the overlay of skills as a separate component of the architecture (a different API), it should be possible for users to test different mathematical formalisms to capture in the context of the same Q-matrix. This flexibility should speed development by modularizing the Q-matrix and mathematical tracing individually, thus allowing easier reuse/generalization of Q-matrices or mathematical models. By association, adherence to this recommendation will yield increased learning gains with decreased development times.

Knowledge Space Models

The domain model of knowledge space theory is a large number of possible knowledge states in a domain (or knowledge structure), whereas the student model is a record of which of the knowledge states are mastered. It is essentially a fine-grained overlay model that is not based on cognitive structures (e.g., skills), but is rather based on problem types that are or are not mastered. A student's competence is reflected in the student model as a probabilistic estimate of the types of problems that the student is capable of solving in the domain (Falmagne, Doignon, Cosyn & Thiery, 2003).

The knowledge structure in knowledge space theory is based on the precedence relation, in which ability to solve one type of problem tends to precede solution of another type. Such precedence relations may be due to the prerequisite structure of the domain, but also may be due to the order in which problem types are currently taught (Falmagne et al., 2003). These precedence domain models are then used in a tutoring context to select the next problem to work on that is sensitive to the student's competence as tracked by the overlay student model of the preceding problems currently known. Thus, depending on all the problem types previously mastered in the student model, there is "just prior" (inner fringe) and a "next subsequent" (outer fringe) of items in the problem. The outer fringe constitutes the pedagogical decision of the knowledge space student model given current the probabilities of being in the various knowledge states as inferred by the prior success on individual problem types (Falmagne et al., 2003).

A knowledge structure in knowledge space theory can be described as a Bayesian network without hidden nodes, since each of the nodes maps to a concrete class of problems (Desmarais & Pu, 2005). Knowledge spaces are often hand engineered because of the massive amounts of data needed to infer the complete graph of the domain knowledge precedence relations but such structures also need to be refined with data to enhance accuracy (Falmagne et al., 2003). Part of the reason for the large amounts of data that are needed is because of the extraordinary flexibility of the knowledge space formalism to represent AND/OR precedence relations in which the outer fringe can be arrived at through one or more pathways (Desmarais, Maluf & Liu, 1996). In other words, C can be known if either A and/or B are known.

Partial Order Knowledge Structures (POKS) have been developed as an alternative to account for the difficulties of inferring the AND/OR structure from data alone, as reviewed elsewhere (Desmarais et al., 1996). As an alternative to the AND/OR graphs of the knowledge structure, POKS use a simpler precedence relation that is less powerful but easier to estimate since it requires that all of the prior states be present. Therefore, it captures an AND graph that can be laid out as a directed acyclic graph (DAG) in which each problem type requires all prior problem types to be mastered with some threshold of certainty. Arguments have been made that such a formalism is adequate in part because situations in which there are alternative prerequisites can be quite rare (Desmarais et al., 1996). Conveniently, explicit mathematical

examples and explanation on how to construct knowledge spaces using POKS are detailed in the literature (Desmarais et al., 1996; Desmarais & Pu, 2005).

The best example of a fully deployed knowledge space based system is the ALEKS mathematics tutor (Doignon & Falmagne, 1999; Hu et al., 2012). In this system, there is a very large knowledge space model for mathematics. The system begins with a student model building phase in which it asks questions without pedagogical intent, but rather to determine the student state. This assessment process is also typically set to run after 5 hours or 20 item types are passed so as to recalibrate the state of the student model. After the student model is determined (the state of the student relative to the domain representation, i.e., the inner fringe), the outer fringe can be inferred from the domain structure. This inference of the outer fringe appears to be the primary pedagogical move in the ALEKS system. The pedagogical model is simply to give the student the choice of any problem type in their outer fringe. Once working on a problem type the student model is straightforward, because it simply involves allowing the student to branch to a new outer fringe problem type if they get the current problem type correct four times in a row, though they have the option of more practice if they wish (personal communication, ALEKS Inc.; http://www.aleks.com/about_aleks/research_behind).

While some ITSs focus on context sensitive adaption either through constraints or skill models, knowledge space theory systems (as exemplified by ALEKS) rely on a far more complex domain model. This domain model essentially presolves the appropriate possible branches from one state to another. This presolved domain-space solution maps to pedagogy through the detailed precedence relationships, leaving the system very little "intelligence" in deciding content order. This methodology harkens back to the branching systems discussed earlier. Essentially, ALEKS is a controlled branching network system with limited student choice of options from the outer fringe of the network. Importantly, knowledge space systems have not made a great effort to consider the different possible branching rules that might determine which outer fringe item is given to a student (this is left in the student's hands) or how many repetitions may actually be needed to transition from one type to another. So, while knowledge space student models provide a nice example of branching, this student modeling method, while parsimonious, could be made more flexible if the permitted pedagogical inferences were allowed to be more complex as a function of the student's history of performance.

Systems other than ALEKS have also been developed, most notably the work on the Catalyst system for chemistry (Arasasingham, Martorell & McIntire, 2011; Arasasingham, Taagepera, Potter, Martorell & Lonjers, 2005). This system seems to be quite effective, but relies on a strategy of breaking the knowledge space up into different representational foci, including numeric, symbolic, and visual problem types so as to be able to plan learning precedence relations that include attention to integrated representations of the content for the students. We might speculate that this improves effectiveness based on research on multiple representations, making it difficult to assess the pure contribution of the model. Student modeling with knowledge spaces appears to be a fertile area to the point where other independent groups are contributing significantly to development of the techniques (Albert & Lukas, 1999).

Intelligent tutoring frameworks like GIFT should certainly include support for knowledge space student modeling. Knowledge space student modeling uses the structure of the domain (perhaps represented by a matrix of problem to problem relationships) to assess where the students are in that domain (the student model). In GIFT the Q-matrix formalism might be expanded to include matrices that encode precedence relations, perhaps by referring to the matrix as the "domain map." The student model using knowledge space computes a likelihood across the possible states of the domain map based on some prior assessment. While the procedure of updating the likelihood seems like it will be conceptually complex, it also appears analogous to the process of inference in the overlay models that results in an update to their long-term model quantities. In both cases (knowledge spaces and overlay models), the system makes pedagogical inferences based on some numeric representation of the long-term student model state.

Dialogue Student Models

This type of student modeling is typical for ITSs that help students learn by holding a conversation in natural language, such as AutoTutor or Why-Atlas (Graesser, D'Mello, et al., 2012; VanLehn et al., 2007). Although these conversational ITSs based in natural language vary in terms of the pedagogical activities they support, they all share two defining attributes. First, these conversational ITSs are based on naturalistic observations and computational modeling of human tutoring strategies embedded in tutorial dialogue (D'Mello, Olney & Person, 2010; Graesser & Person, 1994; Graesser, Person & Magliano, 1995; Person, Graesser, Magliano & Kreuz, 1994). A common strategy is the so-called five-step dialogue frame (Graesser & Person, 1994; Graesser et al., 1995; Person et al., 1994): (1) Tutor asks a deep reasoning question, (2) Student gives an answer, (3) Tutor gives immediate feedback or pumps the student, (4) Tutor and student collaboratively elaborate an answer, and (5) Tutor assesses the student's understanding. This strategy reflects the other defining attribute of conversational ITSs, namely, the emphasis on collaborative, constructive activities based in theories of learning and tutoring (Aleven & Koedinger, 2002; M. T. H. Chi, 2009; M. T. H. Chi, Siler, Jeong, Yamauchi & Hausmann, 2001; Fox, 1993; Graesser et al., 1995; Moore, 1994; Shah, Evens, Michael & Rovick, 2002; VanLehn, Jones & Chi, 1992).

Because conversational ITSs are fundamentally rooted in dialogue, their corresponding student models are usually structured in dialogue-centric terms. A useful (though distinctly non-ITS) oriented framework for modeling dialogues has previously be established by McTear (2002). This framework identifies three major kinds of dialogue models: graph-, frame-, and agent-based. In a graph-based system, at any given moment there are a fixed set of alternative user actions; analogously, each user possible input is associated with an arc from a given node/state of a graph to a new node/state. A common example is a phone-menu dialogue system, e.g., "Press 1 for directory, 2 for billing, or 0 to speak with an operator." Graph-based systems are relatively rigid because they restrict the user's input. Analogously, one would have to slowly progress through a phone-menu dialogue to get to the desired state, rather than being able to say, "I need to speak to billing" at the first opportunity. Graph-based systems are largely comparable to branching student models in terms of the tutor behavior they support. Frame-based systems are slightly more flexible than graph-based systems. In a frame-based system, there is a single overall goal represented by a frame, a fixed attribute-value data structure. The goal of the system is to fill each of the value slots in the frame, in any order. For example, an online reservation system might have the slots [*departure date*, *departure time*, *destination city*, *departure city*, *arrival time*, *arrival date*]. An initial dialogue move from a frame-based system might be, "What reservation would you like to make today?" If a user says, "I'd like to go to Vegas tomorrow," the system would recognize and fill two of the needed slots in the frame, and follow up with a question like, "What time would you like to leave tomorrow?" Thus, if the user gives a full frame's worth of information on the first turn, the system would not need to ask any more questions. If the user gives some but not all of the information, the system would flexibly request the missing information. Frame-based systems are loosely comparable to constraint-based models in terms of the tutor behavior they support. Each slot in a frame specifies a constraint that must be met in order for the frame to be complete. Agent-based systems are the most complex of the three types of dialogue system identified by McTear. The hallmark of agent-based systems is that they are fully mixed-initiative: the user can introduce new topics and goals for the system. This often (but not always) requires that the system be able to recognize the intentions of the user. For example, if the user is trying to find the area under a curve and says, "I don't know how to do integration," the system would (1) recognize that the current goal requires integration, (2) recognize that the user is unable to perform integration (a subgoal of the current goal), and (3) attempt to help by, for example, explaining integration to the user. This is a stereotypical example of an agent-based system; however, there is substantial variation in how much artificial intelligence (AI) and symbolic modeling is actually implemented in such a system.

While the three kinds of dialogue modeling defined by McTear could theoretically be applied in a pure fashion, in practice dialogue systems and ITSs blend elements of these three models, to varying degrees, depending on the functionality of the system. Each of the three provides different functional properties. Graph-based systems provide highly specific and state-dependent actions, are rigid in structure, and are robust computationally because user input is unambiguous. Frame-based systems provide less specific, context-free actions and are a flexible means of pursuing a single goal, though they are less computationally robust than graph-based systems because user input is less constrained. Agent-based systems are theoretically the most flexible because they allow unconstrained user input and interaction patterns, but in practice they are computationally fragile because the AI behind agent-based systems is at or beyond the current state of the art.

To better explain how these dialogue models interleave with student models in a conversational ITS, we give a detailed example based on AutoTutor. However, similar examples may be given for a variety of conversational ITSs including Guru, GnuTutor, and Operation ARIES (Millis et al., 2011; Olney, 2009; Olney, Person & Graesser, 2012). The overall structure of an AutoTutor session is as follows:

Problem statement→ Pump → **Expectation coverage*** →Summary

In other words, each session begins with an introduction ending with a problem statement. When the student responds, the tutor typically follows with a *pump*, e.g., "What else can you say?" Then the tutor launches zero or more cycles of expectation coverage, and follows these with a summary of the answer to the problem. From the above description, the overall structure of the dialogue is graph-based, with no alternatives. Correspondingly, at this level, there is no real need for student modeling. However, *expectation coverage* is not an automatic action, but rather a composite action or *mode* of the tutor; this is where nearly all of the tutor-student interaction takes place.

AutoTutor's expectation coverage is modeled after *Expectation and Misconception Tailored* (EMT) dialogue found in authentic tutoring sessions (Graesser, D'Mello, et al., 2012). Human tutors typically have a list of expectations (anticipated good answers, steps in a procedure) and a list of anticipated *misconceptions* associated with each problem statement. For example, expectations E1 and E2 and misconceptions M1 and M2 are relevant to the following example physics problem:

> PHYSICS QUESTION: If a lightweight car and a massive truck have a head-on collision, upon which vehicle is the impact force greater? Which vehicle undergoes the greater change in its motion, and why?
>
> E1. The magnitudes of the forces exerted by A and B on each other are equal.
>
> E2. If A exerts a force on B, then B exerts a force on A in the opposite direction.
>
> M1: A lighter/smaller object exerts no force on a heavier/larger object.
>
> M2: Heavier objects accelerate faster for the same force than lighter objects

During expectation coverage, the goal of a conversational ITS is to elicit an explanation from the student with regard to the problem statement or seed question. Since student explanations often contain multiple propositions or sentences, this goal reduces to eliciting each sentence in the *expected* explanation. We label these subgoals expectations, because they are expected parts of the overall explanation. The similarity with frame-based systems should be evident: during expectation coverage, the goal of the ITS is to fill out a frame corresponding to the complete explanation to the initial problem statement. Each sentence of the explanation corresponds to a slot in the frame. The task for the ITS is to elicit each

expectation in the frame, which involves two subtasks. First, the ITS must choose a strategy that results in a student response matching an unsatisfied slot/expectation. Second, the ITS must evaluate the student response to determine which (if any) of the unsatisfied slots/expectations it satisfies. Thus, the student model during expectation coverage is largely a frame type structure representing what (if any) of the expectations the student "knows." The ITS uses this information to focus on slots the student does not know, creating a more adaptive and efficient learning experience. Slots may be selected using criteria such as semantic proximity to the current slot (i.e., zone of proximal development).

However, for each slot/expectation, the student model may be augmented with a measure of the student's ability, based on the strategy the tutor used that resulted in a satisfied slot/expectation. The connection between tutor strategy and student ability is based on the observation that some strategies require more *effort* from the student in terms of recall and cognitive processing (e.g., prediction, inference, causal reasoning). A major strategy is asking questions ranging from vague to highly specific (Graesser et al., 1995; Person & Graesser, 2003). For example, a pump is a vague question consisting of neutral back channel feedback (uh-huh, okay, subtle head nod) or explicit requests for more information (what else, tell me more). Pumping serves the functions of exposing student knowledge and encouraging students to construct content by themselves. Hints give context to the question, but do not lead the student to a specific answer, for example, "What about blood pressure in this situation?" In contrast, prompts are highly specific questions in which tutors supply a discourse context and prompt the student to fill in a missing word or phrase, for example, "As the heart beats faster, the blood pressure does what?" In addition to pumps and prompts, many other question types exist that vary in specificity and depth of processing required by the student, such as disjunctive questions or causal questions (Graesser & Person, 1994). For a given expectation, if a student generated an answer satisfying it when the tutor strategy was more difficult, then the student may have greater ability/mastery of that expectation than if the strategy was easy, for example a prompt. Thus, within an expectation coverage cycle, the point at which a student satisfies the expectation is another aspect of the student model. In AutoTutor, the expectation coverage cycle is as follows:

Hint → Prompt → Assertion

This provides two levels of mastery assessment, since a tutor *assertion* is a paraphrase of the expectation given to the student. This cycle is graph-based, but the cycle terminates when the expectation is satisfied.

As mentioned previously, a conversational ITS not only uses strategies to elicit a student response but also evaluates the student response to determine if it satisfies any expectation. In addition, a conversational ITS typically anticipates student misconceptions. An expectation or misconception is scored as being expressed by a student if the student articulates it in natural language with a high enough semantic match. Semantic matches can be assessed by a number of methods in computational linguistics, such as content word overlap, latent semantic analysis, regular expressions, semantic entailment, or Bayesian statistics (Cai et al., 2011; Graesser, Penumatsa, Ventura, Cai & Hu, 2007; Rus, McCarthy, Graesser & McNamara, 2009; VanLehn et al., 2007). Depending on the method used, the match between an expectation/misconception and the student response can range from a single number between 0 (no match) and 1 (identical), or can consider the response compositionally in order to debug the answer. In either case, the evaluation becomes part of the student model.

If the evaluation is a single number, then the frame becomes a "soft frame," where expectations are satisfied if the evaluation is above a numeric threshold. Even when the expectation is satisfied, the evaluation score can be used to indicate relative mastery. For example, if the evaluation score is above the threshold of 0.5, then the ITS may decide that the corresponding expectation is sufficiently "known" that the tutor can move on to other expectations. However, while evaluation scores of 0.6 and 0.9 would both satisfy the threshold and result in the same tutor behavior, the latter student may have mastered the

expectation to a higher degree than the former. Thus, the student model in this case is a soft frame where each slot is weighted by the final evaluation score that led to the satisfaction of the expectation. Complementary extensions to the student model include the entire history of evaluation scores for each expectation. Even though these single number evaluations may appear one-dimensional, as part of the student model they may be used to generate complex adaptive behavior by the ITS. For example, a conversational ITS will usually select a specific question in order to maximize the chance that a student will satisfy an expectation (assuming the student generates the correct response). Likewise, a conversational ITS will usually select the next expectation to cover based on its similarity to the current expectation.

The advantage to a numeric semantic match that it is very robust to noise. The standard approach is to use latent semantic analysis (LSA), an unsupervised technique that creates numeric vector representations for words in a large unstructured collection of texts (Landauer, McNamara, Dennis & Kintsch, 2007). The so-called LSA vector space, together with the expectations and misconceptions described above, represents the domain model for an AutoTutor session. The corresponding semantic match in LSA is the vector cosine. This approach is very robust to ungrammatical and fragmentary user inputs and requires no knowledge engineering expertise. LSA type approaches are generally applicable to natural language inputs in domains where the order of words in the student's answer have little impact on the meaning of the answer. Counterintuitively, this order-free property applies in most cases (Landauer et al., 2007), though would not apply in cases where the answer contained mathematical formulae or similar representations.

As mentioned above, agent-based systems support the most sophisticated dialogue behavior but come with the cost of added complexity and loss of robustness. While generally true, it is possible to satisfice the problem of mixed-initiative in such a way that avoids the larger issues addressed by agent-based systems. In some versions of AutoTutor, mixed-initiative dialogues are created by recognizing and responding to student questions. Every student input is first analyzed to determine if it is an answer to a current tutor question or is a new student question, using a speech act classifier (Olney et al., 2003). When AutoTutor answers a question, it enters a sub-dialogue that is nested in the larger dialogue of the tutoring session. In essence, AutoTutor temporarily suspends its tutoring agenda in favor of answering the question posed by the student. Different student question types indicate shallow (e.g., verification or definition) or deep level reasoning (e.g., causal consequence or inferential) and so can be used to inform the student model (Graesser & Person, 1994). However, while question asking is an important part of active learning, students tend to ask very few questions, even when specifically asked by AutoTutor, "Do you have any questions?" Thus, the utility of question classification and corresponding mixed-initiative dialogue may be low relative to the practical complexity of creating such a system. In addition, the potential of misclassifying a student input increases with the number of speech act categories, meaning that part of the cost of having a question-answering component is incorrectly classifying student answers as questions.

Dialogue-based student models have several advantages that make them attractive for a domain-independent, general purpose framework like GIFT. Numeric semantic matches (e.g., LSA) are highly robust when used to evaluate student input. Students can use fragmentary language or synonyms and still have their answers accepted by the tutor. However, numeric semantic matches are also prone to biases implicit in the corpus of texts used to create them. These biases can be invisible to authors who are creating dialogues for the tutor. For example in biology, the terms "prokaryotic" and "eukaryotic" are often mentioned in the same paragraph but compared and contrasted with each other. Because of this, a biology LSA space will give a high numeric semantic match between the two, even though they are quite different (eukaryotes have a nucleus and membrane-bound organelles, e.g., animals and plants, and prokaryotes do not, e.g., bacteria).

Because of this problem (and biases in semantic spaces more generally), we propose the following for GIFT as well as future work that uses numeric semantic matches in dialogue student models. First, the semantic space should be developed before the dialogues are created and incorporated into the dialogue authoring tool. Second, when the expectation and misconception dialogues are created, the authoring tool should use the semantic space to evaluate the dialogue as it is authored and offer real-time feedback. For example, suppose that the author has just created a prompt, "What do we call cells that have a nucleus?" with the correct answer "eukaryotic cells." If the authoring tool lists the near neighbors of "eukaryotic cells" in the semantic space, then the author will become aware that an incorrect answer, "prokaryotic cells," would be accepted as a correct answer for this prompt. This gives the author the opportunity to apply additional constraints, like regular expressions, to ensure that only correct answers will be accepted (Graesser, D'Mello, et al., 2012). This kind of feedback will help keep the student model from being too forgiving of student input.

A second kind of real-time feedback that we propose involves checking the correspondence between the expectation and the ideal student responses to questions involving that expectation. For example, if the expectation is, "Eukaryotic cells have a nucleus," with only the associated prompt "What do eukaryotic cells have that contains DNA?" then it's possible that the student could type "nucleus" but not actually cover the expectation. In other words, it's possible to write dialogue that would never guide the student to cover the expectation, leading to a malformed student model. A solution to this is to provide real-time feedback that computes the semantic match between all of the ideal student answers to the hints and prompts associated with an expectation and the expectation itself. If the match is very low, this could indicate a problem. Moreover, the threshold for the expectation itself is probably best determined by some percentage of this score rather than being held constant for all expectations in a problem. Following this recommendation should prevent the student model from being too harsh with student input.

State and Trait Identification Models

This category includes student models that capture affective/motivational constructs that are either transient or trait like. This category is distinct from overlay models that capture some sort of learnable skill (e.g., self-regulation). Since traits tend not to change (unlike skills) and states tend to fluctuate in reaction to stimuli (unlike skills), this category logically excludes any learned proficiency based models. Despite this, it seems likely that state and trait models would be used in conjunction with learned proficiency models as an additional input to infer student states and make pedagogical decisions.

Because of the role of affect and cognition in the learning process, a significant amount of ITS research is focused on measurement and detection (Cerri, Clancey, Papadourakis & Panourgia, 2012). This occurs through the incorporation of hardware sensors of software detectors into the standard learning system. Examples of these include an electroencephalography (EEG) head-cap (Goldberg et al., 2011), AI-based software models (Wixon, Baker, Gobert, Ocumpaugh & Bachmann, 2012), or a combination of both (Kapoor & Picard, 2005).

Human tutors are known to be as devoted to the motivation of the student as much as their cognitive and informational goals (Lepper & Hodell, 1989; Woolf, 2008) Affective characteristics have also been highlighted in the literature as an important area to learning (Woolf, 2010). While there has been significant research in their development, there has been limited transition to usable systems. One nice example of a strong success comes from feature detection in spelling learning (Baschera, Busetto, Klingler, Buhmann & Gross, 2011).

While this category has few example contexts where there are significant benefits to learning in a running ITS, the category has distinct potential, particularly as machined learning and sensor technologies develop

to the point of practical application. The state and trait distinction has particular relevance to the architecture of general systems such as GIFT. In particular, GIFT should be able to handle stable student parameters that feed into the student model as individual differences. For instance, a prior assessment of each student's self-efficacy could be passed to the dialog based student model which would lead to the pedagogical inference to adjust the scripts to be more encouraging with low efficacy students. Similarly, sensor modules data would feed into student models in a continuous fashion so that it could be incorporated into any long-term student model (e.g., BKT) and thus affect pedagogical decision making (e.g., D'Mello & Graesser, in press).

Discussion

An ITS functions by having a domain model which defines the space of possible student states. The student model keeps tracks of each individual student within this possible state space. The pedagogical model is a set of rules that define when interventions are triggered as a function of the student state and the tutor interface. The student model represents the student state as a function of prior actions, either tracking these prior actions dynamically so that prior states are not necessarily stored or by keeping track the history of prior practice in order to compute student prior knowledge as the context demands. Pedagogical rules aim to maximize a measure of learning amount or efficiency.

We began our discussion of student models in the late 1950s with the invention of branched programming. Branched programming addresses student state differences by applying different pedagogical moves at fixed branch points afforded by the domain model. Branching models are also discussed because *fundamentally* all student models "branch" in the sense that any pedagogical move we might take is in essence a branch in response to the student state as encoded in the possibilities expressed by the domain model. These branches or adaptive choices must depend on something, and the student model state is the independent input (derived from student actions) that leads to the inferences about adaptive pedagogy. However, a universal branching system in the context of GIFT, or more generally, as how to use them as an effective generic design pattern for tracking student states to make rich and powerful pedagogical moves, is currently unclear.

VanLehn helps provides a design pattern for us to consider in his proposal about the behavior of tutoring systems where he proposed inner and outer loops to characterize common ways that ITSs have been configured (VanLehn, 2006). In this taxonomy, the inner loop was construed as the individual problem, which would be intelligently tutored in small cycles of interaction between the student and ITS as the student works on the problem. This is also known as microadaption. In contrast, the outer loop was described as a problem selection loop, from which a problem selection algorithm might choose the next item based on higher criteria than behavior at the single problem level. This is known as macroadaption. Essentially, this is a hierarchy of kinds of branching, with the inner loop often representing stereotyped branching schema within problems for individual steps, and the outer loop specifying sequencing/ branching between exercises. Of course, within this hierarchy we might describe more levels. For example, Pavlik Jr. & Toth (2010) proposed an additional level of branching complexity, the curriculum loop, which assumes that a student model would track states for broad categories of problems (units of content) and would allow pedagogical models to execute actions at an additional level, as in the case of sending a student into a basic skills review unit if repeated arithmetic errors are made in an algebra unit.

In mature ITS, there is a blend of student models. For example, in cognitive tutors, problem selection occurs with some version of BKT (Corbett & Anderson, 1995) used as a student model for the outer loop, but within problems the student model tends to be more like a constraint-based model (even though often represented as production rules) with inner loop pedagogical responses to student actions disconnected from the outer loop BKT student model. For instance, error flagging or on-demand hints to the student

represent the within problem student model as a collection of response constraints. These "should" statements define what features of the interface need to be modified in what way to solve the problem, while the pedagogical model specifies what inference to make based on the current state of the interface.

The cognitive tutor example illustrates how a learning system could potentially have a student model that allows state branches at multiple levels of the domain (problem step, overall problem, curriculum unit), where the means of modeling the branching at each level of the student model may be different due to the pedagogical needs of that level of the domain. Similarly, we could imagine a ITS with a constraint-based student model for tutoring algebra problem solving as the inner loop, a knowledge space domain model used for the outer loop problem selection, and a curriculum branching loop that that branches students into a basic skills unit if arithmetic constraints are not met. The second arithmetic outer loop might use an overlay model for all the facts in the times tables, with no inner loop for these simple facts. This discussion places further constraints on GIFT by illustrating the advantage of having a hierarchical organization of student models for the multiple levels of domain structure so as to facilitate maximum flexibility for pedagogical strategies at these multiple levels.

Figure 5-1 shows the three levels at which pedagogy occurs that seem important to allow in GIFT. The figure attempts to convey the relationship between the three levels, in that the inner loop, which itself represents a collection of states and consequent pedagogical moves) is entered from the outer loop problem. Similarly, an outer loop sequence of problems is selected as the pedagogical decision from the curriculum student model. A simulation scenario example should help clarify. If the curriculum involves practicing team search and reconnaissance missions, we might suppose the curriculum loop would be a number of different contexts for reconnaissance, such as night missions, search and recovery missions, rough terrain missions, etc., each of which of needs to be completed with a certain criterion (according to the student model) before the next context is selected (according to the pedagogical model). Each outer loop problem might be viewed as a specific scenario within a context (e.g., rescue mission in Mogadishu) that itself has to be mastered or repeated, according to the decisions of the outer loop. Finally, within each scenario (the inner loop in this context) we might expect that there would be specific tasks, each with their own states for the student that warrant specific pedagogies.

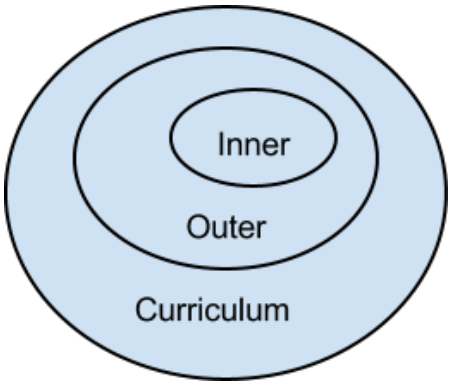

Figure 5-1. Pedagogical levels at which a student model may be used to make decisions

This discussion is useful because it helps set the stage for considering the specific usefulness of particular student models. Student models may generalize to content, but may not as easily generalize to different pedagogical levels. So, for instance, dialogue-based tutoring seems ill suited to handling the problems of curriculum sequencing or outer loop management (e.g., problem selection). Similarly, selection of steps based on some mathematical model makes little sense since the steps of a problem are more or less fixed once that problem is selected. Table 5-1 attempts to summarize our assessment of the student models we have reviewed.

Table 5-1. Summary evaluation comments

	Quantitative Fit	Ease of Understanding	Generality and Flexibility	Cost of Creation	Granularity	Time Scale	Learning gains in practice
Programmed Branching	N/A	High	Low	Low	Any	Short-term	Depends on adequacy of specific domain model
Rule Space	High	High	High	Low	Any	Short or Long-term	Little direct evidence
Model Tracing	High	Low	Moderate	High	Production rules	Potential long-term but not always realized	Yes, strong gains for particular domains
Constraint-Based Tutors	High	Moderate	High	Moderate	Any	Potential long-term but not always realized	Yes, strong gains for particular domains
Knowledge Spaces	Good at item-category level	Low	Low	High	Item-categories	Long-term	Yes, strong gains for particular domains
Dialogue	N/A	Moderate	Moderate	Moderate	Inner loop	Short-term	Yes, strong gains for particular domains
State and Trait	Generally low	Moderate	Unclear	Moderate	Any	Short- and/or long-term	Unclear

Conclusions

There is significant challenge in developing a standard, or framework, to support an existing field. The goal behind developing standards is that they will be adopted, which can only happen if the individual adopters find the new standard useful to their practice. As such, it is important to discuss the benefits of adopting such standards.

GIFT will enable new ITS creators to experimentally test their methods of student model creation with differing domains of instruction, pedagogical strategies, and ways of assessing user states through sensors. GIFT will enable the experimental comparison of one type of model against another, while keeping the other portions of the ITS exactly the same, thus enabling good unconfounded comparisons in the ITS. GIFT will enable quicker construction of ITS application through the ability to standardize user tools for student model construction.

However, these advantages come at a price. This is the price that all standards impose on the community of adoption. At the simplest level, conforming to a "Phillips" screw-head standard destroys the potential for the creation of alternate screw-head designs. No one framework, standard, architecture, or toolset can accommodate for all possible solutions. Despite this, to promote adoption, the created solution should support as many of the current practices as is possible. With these tradeoffs in mind, we conclude with a few points about the process of developing GIFT.

The first point is that the standard approach to instruction remains simple branching programs with multimedia content. Therefore, most student models are merely a record of the place in such a branching program structure. Such systems are still important because intelligent tutoring is not required for many kinds of instruction, such as first-time information presentation. This first assumption leads to the first recommendation: GIFT should support authoring of simple branching programs.

The second point is that the most mature and applicable forms of student modeling are those which have had the most research and software development effort. These are the methods for adaptive instruction that have been empirically proven to work, have multiple research teams working on them, have deployed ITSs to classrooms, and have developed authoring tools to speed developments and research. GIFT, in order to support successful adoption, needs to be inclusive of these models. Becoming inclusive of these models implies that GIFT becomes inclusive of the tools that are used to create them. This second implication is that GIFT should support the importation, with very minimal change, of student models and authoring tools created within CBM, model tracing, and dialog-based paradigms for inner loop mediation. They have proven to be successful and widely adopted.

A third point is that several of the methods, most notably overlay and knowledge spaces, employ "domain maps" that characterize the tasks and the relationships between these tasks. Because of this commonality, it seems likely that GIFT would benefit from providing such a domain map data structure that could be referenced when making pedagogical decisions in reference to the student model. Further, such domain maps are typically connected to a long-term computational student model that mathematically traces the probability of the tasks relative to the prior progress of the student. This implies GIFT will be well served by providing the capacity to mathematically compute pedagogically relevant student model quantities at any time by processing the student's prior data (which is encoded relative to the domain map).

References

Ahmed, K. B. S., Toumouh, A. & Malki, M. (2012). Effective Ontology Learning: Concepts' Hierarchy Building using Plain Text Wikipedia. In *Proceedings ICWIT* (pp. 171).

Albert, D. & Lukas, J. (1999). *Knowledge Spaces: Theories, Empirical Research, and Applications*: L. Erlbaum.

Aleven, V. & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science, 26*, 147–179.

Aleven, V., McLaren, B., Roll, I. & Koedinger, K. R. (2006). Toward meta-cognitive tutoring: A model of help seeking with a cognitive tutor. *International Journal of Artificial Intelligence in Education, 16*, 101-128.

Aleven, V., McLaren, B., Sewall, J. & Koedinger, K. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. In M. Ikeda, K. Ashley & T.-W. Chan (Eds.), *Intelligent Tutoring Systems* (Vol. 4053, pp. 61-70): Springer Berlin / Heidelberg.

Aleven, V., McLaren, B. M., Sewall, J. & Koedinger, K. R. (2009). A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. *International Journal of Artificial Intelligence in Education, 19*, 105-154.

Amalathas, S., Mitrovic, A. & Ravan, S. (2012). Decision-Making Tutor: Providing on-the-job training for oil palm plantation managers. *Research and Practice in Technology-Enhanced Learning, 7*, 131-152.

Anderson, J. R., Conrad, F. G. & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13*, 467-505.

Anderson, J. R., Corbett, A. T., Koedinger, K. R. & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *Journal of the Learning Sciences, 4*, 167–207.

Arasasingham, R. D., Martorell, I. & McIntire, T. M. (2011). Online Homework and Student Achievement in a Large Enrollment Introductory Science Course. *Journal of College Science Teaching, 40*, 70-79.

Arasasingham, R. D., Taagepera, M., Potter, F., Martorell, I. & Lonjers, S. (2005). Assessing the Effect of Web-Based Learning Tools on Student Understanding of Stoichiometry Using Knowledge Space Theory. *Journal of Chemical Education, 82*, 1251.

Atkinson, R. C. (1972a). Ingredients for a theory of instruction. *American Psychologist, 27*, 921-931.

Atkinson, R. C. (1972b). Optimizing the learning of a second-language vocabulary. *Journal of Experimental Psychology, 96*, 124-129.

Atkinson, R. C. & Crothers, E. J. (1964). A comparison of paired-associate learning models having different acquisition and retention axioms. *Journal of Mathematical Psychology, 1*, 285-315.

Ayers, E. & Junker, B. (2006). Do skills combine additively to predict task difficulty in eighth grade mathematics? In J. Beck, E. Aimeur & T. Barnes (Eds.), *Educational Data Mining: Papers from the AAAI Workshop* (pp. 14-20). Menlo Park, CA: AAAI Press.

Baghaei, N., Mitrovic, A. & Irwin, W. (2007). Supporting collaborative learning and problem-solving in a constraint-based CSCL environment for UML class diagrams. *International Journal of Computer-Supported Collaborative Learning, 2*, 159-190.

Baker, R. S. J. d., D'Mello, S. K., Rodrigo, M. M. T. & Graesser, A. C. (2010). Better to be frustrated than bored: The incidence, persistence, and impact of learners' cognitive–affective states during interactions with three different computer-based learning environments. *International Journal of Human-Computer Studies, 68*, 223–241.

Barnes, T. (2005). *The Q-matrix Method: Mining Student Response Data for Knowledge*. Paper presented at the American Association for Artificial Intelligence 2005 Educational Data Mining Workshop.

Barnes, T., Stamper, J. & Madhyastha, T. (2006). *Comparative Analysis of Concept Derivation Using the Q-matrix Method and Facets*.

Baschera, G.-M., Busetto, A., Klingler, S., Buhmann, J. & Gross, M. (2011). Modeling Engagement Dynamics in Spelling Learning. In G. Biswas, S. Bull, J. Kay & A. Mitrovic (Eds.), *Artificial Intelligence in Education* (Vol. 6738, pp. 31-38): Springer Berlin Heidelberg.

Beck, J. & Mostow, J. (2008). How Who Should Practice: Using Learning Decomposition to Evaluate the Efficacy of Different Types of Practice for Different Types of Students. In (pp. 353-362).

Brown, J. S. & VanLehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science, 4*, 379-426.

Burton, R. (1982). Diagnosing bugs in a simple procedural skill. In D. Sleeman & J. Brown (Eds.), *Intelligent Tutoring Systems* (pp. 157-184): Academic Press.

Cai, Z., Graesser, A. C., Forsyth, C. M., Burkett, C., Millis, K., Wallace, P., et al. (2011). Trialog in ARIES: User input assessment in an intelligent tutoring system. In W. C. S. Li (Ed.), *Proceedings of the 3rd IEEE International Conference on Intelligent Computing and Intelligent Systems* (pp. 429–433). Guangzhou, P.R. China: IEEE Press.

Calfee, R. C. & Atkinson, R. C. (1965). Paired-associate models and the effects of list length. *Journal of Mathematical Psychology, 2*, 254-265.

Carrier, M. & Pashler, H. (1992). The influence of retrieval on retention. *Memory & Cognition, 20*, 633-642.

Cen, H., Koedinger, K. R. & Junker, B. (2006). Learning Factors Analysis - A general method for cognitive model evaluation and improvement. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (pp. 164-175): Springer Berlin / Heidelberg.

Cen, H., Koedinger, K. R. & Junker, B. (2007). *Is Over Practice Necessary? – Improving Learning Efficiency with the Cognitive Tutor through Educational Data Mining*. Paper presented at the 13th International Conference on Artificial Intelligence in Education, Los Angeles, CA.

Cen, H., Koedinger, K. R. & Junker, B. (2008). *Comparing two IRT models for conjunctive skills*. Paper presented at the Proceedings of the 9th International Conference on Intelligent Tutoring Systems, Montreal, Canada.

Cerri, S. A., Clancey, W. J., Papadourakis, G. & Panourgia, K. (2012). *Intelligent Tutoring Systems - 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14-18, 2012. Proceedings* (Vol. 7315): Springer.

Chi, M., Koedinger, K. R., Gordon, G., Jordan, P. & VanLehn, K. (2011). *Instructional Factors Analysis: A Cognitive Model For Multiple Instructional Interventions*. Poster presented at the 4th International Conference on Educational Data Mining, Eindhoven, The Netherlands.

Chi, M. T. H. (2009). Active-Constructive-Interactive: A Conceptual Framework for Differentiating Learning Activities. *Topics in Cognitive Science, 1*, 73-105.

Chi, M. T. H., Siler, S. A., Jeong, H., Yamauchi, T. & Hausmann, R. G. (2001). Learning from human tutoring. *Cognitive Science, 25*, 471-533.

Corbett, A. T. & Anderson, J. R. (1992). Student modeling and mastery learning in a computer-based programming tutor. In C. Frasson, G. Gauthier & G. McCalla (Eds.), *Intelligent Tutoring Systems: Second International Conference on Intelligent Tutoring Systems* (pp. 413-420). New York: Springer-Verlag.

Corbett, A. T. & Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction, 4*, 253–278.

Crowder, N. A. (1959). Automatic tutoring by means of intrinsic programming. In E. Galanter (Ed.), *Automatic teaching: The state of the art* (pp. 109-116). New York: Wiley & Sons.

D'Mello, S. K. & Graesser, A. (2010). Multimodal semi-automated affect detection from conversational cues, gross body language, and facial features. *User Modeling and User-Adapted Interaction, 20*, 147-187.

D'Mello, S. K., Graesser, A. & King, B. (2010). Toward Spoken Human-Computer Tutorial Dialogues. *Human Computer Interaction, 25*, 289-323.

D'Mello, S. K., Olney, A. M. & Person, N. (2010). Mining Collaborative Patterns in Tutorial Dialogues. *Journal of Educational Data Mining, 2*, 1-37.

D'Mello, S. K. & Graesser, A. C. (in press). AutoTutor and affective AutoTutor: Learning by talking with cognitively and emotionally intelligent computers that talk back. *ACM Transactions on Interactive Intelligent Systems*.

Desmarais, M. C. & Baker, R. S. J. d. (2012). A review of recent advances in learner and skill modeling in intelligent learning environments. *User Modeling and User-Adapted Interaction, 22*, 9-38.

Desmarais, M. C., Maluf, A. & Liu, J. (1996). User-expertise modeling with empirically derived probabilistic implication networks. *User Modeling and User-Adapted Interaction, 5*, 283-315.

Desmarais, M. C. & Pu, X. (2005). A Bayesian Student Model without Hidden Nodes and its Comparison with Item Response Theory. *Int. J. Artif. Intell. Ed., 15*, 291-323.

Doignon, J.-P. & Falmagne, J.-C. (1999). *Knowledge spaces*: Springer.

Draney, K. L., Pirolli, P. & Wilson, M. (1995). A measurement model for a complex cognitive skill. In P. D. Nichols, S. F. Chipman & R. L. Brennan (Eds.), *Cognitively diagnostic assessment* (pp. 103–125).

Elsom-Cook, M. (1993). Student modelling in intelligent tutoring systems. *Artificial Intelligence Review, 7*, 227-240.

Falmagne, J.-C., Doignon, J.-P., Cosyn, E. & Thiery, N. (2003). The assessment of knowledge in theory and in practice. *Institute for Mathematical Behavioral Sciences, Paper 26*.

Fox, B. A. (1993). *The human tutoring dialogue project*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Goldberg, B., Sottilare, R., Brawner, K. & Holden, H. (2011). Predicting Learner Engagement during Well-Defined and Ill-Defined Computer-Based Intercultural Interactions. In S. D'Mello, A. Graesser, B. Schuller & J.-C. Martin (Eds.), *Affective Computing and Intelligent Interaction* (Vol. 6974, pp. 538-547): Springer Berlin Heidelberg.

Gong, Y., Beck, J. & Heffernan, N. T. (2010). Comparing Knowledge Tracing and Performance Factor Analysis by Using Multiple Model Fitting Procedures. In V. Aleven, J. Kay & J. Mostow (Eds.), *Intelligent Tutoring Systems* (Vol. 6094, pp. 35-44): Springer Berlin / Heidelberg.

Graesser, A. C., Conley, M. W. & Olney, A. (2012). Intelligent tutoring systems. In K. R. H. telligent tutoring systems, S. Graham, T. Urdan, A. G. Bus, S. Major & H. L. Swanson (Eds.), *APA educational psychology handbook, Vol 3: Application to learning and teaching* (pp. 451-473). Washington, DC, US: American Psychological Association.

Graesser, A. C., D'Mello, S. K., Xiangen, H., Cai, Z., Olney, A. & Morgan, B. (2012). AutoTutor. *Applied Natural Language Processing: Identification, Investigation, and Resolution.* pp. 169-187.

Graesser, A. C., Penumatsa, P., Ventura, M., Cai, Z. & Hu, X. (2007). Using LSA in AutoTutor: Learning through mixed initiative dialogue in natural language. In D. M. T. Landauer, S. Dennis & W. Kintsch (Ed.), *Handbook of latent semantic analysis* (pp. 234-262). Mahwah, NJ: Erlbaum.

Graesser, A. C. & Person, N. K. (1994). Question Asking during Tutoring. *American Educational Research Journal, 31*, 104-137.

Graesser, A. C., Person, N. K. & Magliano, J. P. (1995). Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology, 9*, 1-28.

Groen, G. J. & Atkinson, R. C. (1966). Models for optimizing the learning process. *Psychological Bulletin, 66*, 309-320.

Heffernan, N. T., Koedinger, K. R. & Razzaq, L. (2008). Expanding the Model-Tracing Architecture: A 3rd Generation Intelligent Tutor for Algebra Symbolization. *International Journal of Artificial Intelligence in Education, 18*, 153-178.

Heffernan, N. T., Turner, T. E., Lourenco, A. L. N., Macasek, M. A. & Nuzzo-Jones, G. The ASSISTment builder: Towards an analysis of cost effectiveness of ITS creation.

Hu, X., Craig, S. D., Bargagliotti, A. E., Graesser, A. C., Okwumabua, T., Anderson, C., et al. (2012). The Effects of a Traditional and Technology-based After-school Setting on 6th Grade Student's Mathematics Skills. *Journal of Computers in Mathematics and Science Teaching, 31*, 17-38.

Johnson, W. L. & Valente, A. (2008). Tactical language and culture training systems: using artificial intelligence to teach foreign languages and cultures, *Proceedings of the 20th national conference on Innovative applications of artificial intelligence - Volume 3* (pp. 1632-1639). Chicago, Illinois: AAAI Press.

Kapoor, A. & Picard, R. W. (2005). *Multimodal affect recognition in learning environments*. Paper presented at the Proceedings of the 13th annual ACM international conference on Multimedia, Hilton, Singapore.

Karpicke, J. D. & Roediger, H. L., III. (2008). The critical importance of retrieval for learning. *Science, 319*, 966–968.

Koedinger, K. R., Pavlik Jr., P. I., Stamper, J., Nixon, T. & Ritter, S. (2011). Fair blame assignment in student modeling. In M. Pechenizkiy, T. Calders, C. Conati, S. Ventura, C. Romero & J. Stamper (Eds.), *Proceedings of the 4th International Conference on Educational Data Mining* (pp. 91–100). Eindhoven, the Netherlands.

Lahti, L. (2010). *Personalized learning paths based on Wikipedia article statistics*. Paper presented at the CSEDU 2010.

Landauer, T. K., McNamara, D. S., Dennis, S. E. & Kintsch, W. E. (2007). *Handbook of latent semantic analysis*: Lawrence Erlbaum Associates Publishers.

Lepper, M. R. & Hodell, M. (1989). Intrinsic Motivation in the Classroom. *Research on Motivation in Education: Goals and cognitions, 3*, 73.

Litman, D. (2013). Speech and language processing for adaptive training. In P. Durlach & A. Lesgold (Eds.), *Adaptive technologies for training and education*.: Cambridge University Press.

Lumsdaine, A. A. & Glaser, R. E. (1960). *Teaching Machines and Programmed Learning, a Source Book*. Washington DC. : National Education Association, Dept. of Audiovisual Instruction.

Mayo, M. & Mitrovic, A. (2001). Optimising ITS Behaviour with Bayesian Networks and Decision Theory. *International Journal on Artificial Intelligence in Education, 12*, 124-153.

Mayo, M., Mitrovic, A. & McKenzie, J. (2000). CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation, *International Workshop on Advanced Learning Technologies* (Vol. 0, pp. 151-154). Palmerston North, New Zealand: IEEE Computer Society.

McTear, M. F. (2002). Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys (CSUR), 34*, 90-169.

Millis, K., Forsyth, C., Butler, H., Wallace, P., Graesser, A. & Halpern, D. (2011). Operation ARIES!: A Serious Game for Teaching Scientific Inquiry. *Serious Games and Edutainment Applications*, pp. 169-195.

Mitrovic, A. (1998). Experiences in Implementing Constraint-Based Modeling in SQL-Tutor. In B. Goettl, H. Halff, C. Redfield & V. Shute (Eds.), *Intelligent Tutoring Systems* (Vol. 1452, pp. 414-423): Springer Berlin Heidelberg.

Mitrovic, A. (2003). An Intelligent SQL Tutor on the Web. *International Journal of Artificial Intelligence in Education, 13*, 173-197.

Mitrovic, A. (2012). Fifteen years of constraint-based tutors: what we have achieved and where we are going. *User Modeling and User-Adapted Interaction, 22*, 39-72.

Mitrovic, A., Koedinger, K. R. & Martin, B. (2003). A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling. In *User Modeling 2003* (Vol. 2702/2003): Springer Berlin / Heidelberg.

Mitrovic, A. & Martin, B. (2007). Evaluating the Effect of Open Student Models on Self-Assessment. *International Journal of Artificial Intelligence in Education, 17*, 121-144.

Mitrovic, A., Martin, B. & Suraweera, P. (2007). Intelligent Tutors for All: The Constraint-Based Approach. *IEEE Intelligent Systems, 22*, 38-45.

Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., et al. (2009). ASPIRE: An Authoring System and Deployment Environment for Constraint-Based Tutors. *Int. J. Artif. Intell. Ed., 19*, 155-188.

Mitrovic, A. & Ohlsson, S. (1999). Evaluation of a Constraint-Based Tutor for a Database. *International Journal of Artificial Intelligence in Education, 10*, 238-256.

Mitrovic, A., Ohlsson, S. & Barrow, D. K. (2013). The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education, 60*, 264-272.

Mitrovic, A. & Weerasinghe, A. (2009). Revisiting the Ill-Definedness and Consequences for ITSs. In V. Dimitrova, R. Mizoguchi, B. du Boulay & A. Graesser (Eds.), *Proc 14th Int Conf AIED* (pp. 375-382).

Moore, J. D. (1994). *Participating in explanatory dialogues: interpreting and responding to questions in context*. Cambridge, MA, USA: MIT Press.

Nkambou, R., Mizoguchi, R. & Bourdeau, J. (2010). *Advances in Intelligent Tutoring Systems* (Vol. 308): Springer.

O'Reilly, R. C. (1998). Six principles for biologically based computational models of cortical cognition. *Trends in Cognitive Sciences, 2*, 455-462.

Ohlsson, S. (1992). Constraint-based student modelling. *International Journal of Artificial Intelligence in Education, 3*, 429-447.

Ohlsson, S. & Mitrovic, A. (2007). Fidelity and Efficiency of Knowledge Representations for Intelligent Tutoring Systems. *Technology, Instruction, Cognition and Learning (TICL), 5*, 101-132.

Olney, A. M. (2009). GnuTutor: An open source intelligent tutoring system, *Proceedings of the 14th International Conference on Artificial Intelligence in Education* (pp. 803). Brighton UK: Amsterdam: IOS Press.

Olney, A. M., Louwerse, M., Mathews, E., Marineau, J., Hite-Mitchell, H. & Graesser, A. C. (2003). Utterance Classification in AutoTutor, *Proceedings of the HLT-NAACL 03 Workshop on Building Educational Applications Using Natural Language Processing* (pp. 1-8). Philadelphia: Association for Computational Linguistics.

Olney, A. M., Person, N. K. & Graesser, A. C. (2012). Guru: Designing a Conversational Expert Intelligent Tutoring System. *Cross-Disciplinary Advances in Applied Natural Language Processing: Issues and Approaches*, pp. 156-171.

Pavlik Jr., P. I. (2007). Understanding and applying the dynamics of test practice and study practice. *Instructional Science, 35*, 407–441.

Pavlik Jr., P. I. & Anderson, J. R. (2008). Using a model to compute the optimal schedule of practice. *Journal of Experimental Psychology: Applied, 14*, 101–117.

Pavlik Jr., P. I., Cen, H. & Koedinger, K. R. (2009). Performance factors analysis -- A new alternative to knowledge tracing. In V. Dimitrova, R. Mizoguchi, B. d. Boulay & A. Graesser (Eds.), *Proceedings of the 14th International Conference on Artificial Intelligence in Education* (pp. 531–538). Brighton, England.

Pavlik Jr., P. I., Presson, N., Dozzi, G., Wu, S.-m., MacWhinney, B. & Koedinger, K. R. (2007). The FaCT (Fact and Concept Training) System: A new tool linking cognitive science with educators. In D. McNamara & G. Trafton (Eds.), *Proceedings of the Twenty-Ninth Annual Conference of the Cognitive Science Society* (pp. 1379–1384). Mahwah, NJ: Lawrence Erlbaum.

Pavlik Jr., P. I. & Toth, J. (2010). How to build bridges between intelligent tutoring system subfields of research. In J. Kay, V. Aleven & J. Mostow (Eds.), *Proceedings of the 10th International Conference on Intelligent Tutoring Systems, Part II* (pp. 103–112). Pittsburgh, PA: Springer.

Pavlik Jr., P. I., Yudelson, M. & Koedinger, K. R. (2011). Using contextual factors analysis to explain transfer of least common multiple skills. In G. Biswas, S. Bull, J. Kay & A. Mitrovic (Eds.), *Artificial Intelligence in Education* (Vol. 6738, pp. 256–263). Berlin, Germany: Springer.

Person, N. K. & Graesser, A. C. (2003). Fourteen facts about human tutoring: Food for thought for ITS developers, *AI-ED 2003 Workshop Proceedings on Tutorial Dialogue Systems: With a View Toward the Classroom* (pp. 335-344).

Person, N. K., Graesser, A. C., Magliano, J. P. & Kreuz, R. J. (1994). Inferring what the student knows in one-to-one tutoring: The role of student questions and answers. *Learning and Individual Differences, 6*, 205–229.

Peterson, L. R. (1965). Paired-associate latencies after the last error. *Psychonomic Science, 2*, 167-168.

Psotka, J., Massey, L. D. & Mutter, S. A. (1988). *Intelligent tutoring systems: Lessons learned*: Lawrence Erlbaum.

Rey-López, M., Fernández-Vilas, A., Díaz-Redondo, R., Pazos-Arias, J. & Bermejo-Muõz, J. (2006). Extending SCORM to Create Adaptive Courses. In W. Nejdl & K. Tochtermann (Eds.), *Innovative Approaches for Learning and Knowledge Sharing* (Vol. 4227, pp. 679-684): Springer Berlin Heidelberg.

Rickard, T. C. (1997). Bending the power law: A CMPL theory of strategy shifts and the automatization of cognitive skills. *Journal of Experimental Psychology: General, 126*, 288-311.

Rickard, T. C. (1999). A CMPL alternative account of practice effects in numerosity judgment tasks. *Journal of Experimental Psychology: Learning, Memory & Cognition, 25*, 532-542.

Ritter, S. & Anderson, J. (2006). *Cognitive Tutor: Tracking learning in real time. Testimony to the National Mathematics Panel*.

Ritter, S., Anderson, J. R., Koedinger, K. R. & Corbett, A. (2007). Cognitive Tutor: Applied research in mathematics education. *Psychonomic Bulletin & Review, 14*, 249-255.

Rus, V. & Graesser, A. C. (2009). *The Question Generation Shared Task and Evaluation Challenge*: University of Memphis.

Rus, V., McCarthy, P., Graesser, A. & McNamara, D. (2009). Identification of Sentence-to-Sentence Relations Using a Textual Entailer. *Research on Language and Computation, 7*, 209-229.

Schunn, C. D. (2005). Evaluating goodness-of-fit in comparison of models to data. In W. Tack (Ed.), *Psychologie der Kognition: Reden and Vorträge anlässlich der Emeritierung von Werner Tack* (pp. 115-154). Saarbrueken, Germany: University of Saarland Press.

Segalowitz, N. S. & Segalowitz, S. J. (1993). Skilled performance, practice, and the differentiation of speed-up from automatization effects - evidence from 2nd-language word recognition. *Applied Psycholinguistics, 14*, 369-385.

Shah, F., Evens, M., Michael, J. & Rovick, A. (2002). Classifying Student Initiatives and Tutor Responses in Human Keyboard-to-Keyboard Tutoring Sessions. *Discourse Processes, 33*, 23-52.

Sleeman, D. & Brown, J. S. (1982). *Intelligent tutoring systems*. New York: Academic Press.

Sleeman, D., Ward, R. D., Kelly, E., Martinak, R. & Moore, J. (1991). An overview of recent studies with PIXIE. In P. Goodyear (Ed.), *Teaching Knowledge and Intellgient Tutoring* (pp. 173-185).

Smallwood, R. D. (1962). *A decision structure for teaching machines*. Cambridge: MIT Press.

Smallwood, R. D. (1971). The analysis of economic teaching strategies for a simple learning model. *Journal of Mathematical Psychology, 8*, 285-301.

Spada, H. & McGraw, B. (1985). The assessment of learning effects with linear logistic test models. In S. Embretson (Ed.), *Test design: Developments in psycholgoy and psychometrics*. Orlando, FL: Academic Press.

Suraweera, P. & Mitrovic, A. (2004). An Intelligent Tutoring System for Entity Relationship Modelling. *International Journal of Artificial Intelligence in Education, 14*, 375-417.

Tatsuoka, K. K. (1983). Rule space: An approach for dealing with misconceptions based on item response theory. *Journal of Educational Measurement, 20*, 345-354.

Thomas, C., Davies, I., Openshaw, D. & Bird, J. (1963). *Programmed Learning in Perspective: A Guide to Program Writing*: Aldine De Gruyter.

Thompson, C. P., Wenger, S. K. & Bartling, C. A. (1978). How recall facilitates subsequent recall: A reappraisal. *Journal of Experimental Psychology: Human Learning & Memory, 4*, 210-221.

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education, 16*, 227-265.

VanLehn, K., Graesser, A. C., Jackson, G. T., Jordan, P., Olney, A. M. & Rosé, C. P. (2007). When are tutorial dialogues more effective than reading? *Cognitive Science, 31*, 3–62.

VanLehn, K., Jones, R. M. & Chi, M. T. H. (1992). A Model of the Self-Explanation Effect. *Journal of the Learning Sciences, 2*, 1 - 59.

Weerasinghe, A., Mitrovic, A. & Martin, B. (2009). Towards Individualized Dialogue Support for Ill-Defined Domains. *International Journal of Artificial Intelligence in Education, 19*, 357-379.

Wixon, M., Baker, R. S. J. d., Gobert, J. D., Ocumpaugh, J. & Bachmann, M. (2012). *WTF? detecting students who are conducting inquiry without thinking fastidiously*. Paper presented at the Proceedings of the 20th international conference on User Modeling, Adaptation, and Personalization, Montreal, Canada.

Woolf, B. P. (2008). *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. San Francisco, CA: Morgan Kaufmann.

Woolf, B. P. (2010). *A roadmap for education technology*: National Science Foundation.

Zakharov, K., Mitrovic, A. & Ohlsson, S. (2005). *Feedback Micro-engineering in EER-Tutor*. Paper presented at the Proceeding of the 2005 conference on Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology.