New Potentials for Data-Driven Intelligent Tutoring System Development and Optimization

Kenneth R. Koedinger, Emma Brunskill, Ryan S. J. d. Baker,
Elizabeth A. McLaughlin, John Stamper

■ *Increasing widespread use of educational technologies is producing vast amounts of data. Such data can be used to help advance our understanding of student learning and enable more intelligent, interactive, engaging, and effective education. In this article, we discuss the status and prospects of this new and powerful opportunity for data-driven development and optimization of educational technologies, focusing on intelligent tutoring systems. We provide examples of use of a variety of techniques to develop or optimize the select, evaluate, suggest, and update functions of intelligent tutors, including probabilistic grammar learning, rule induction, Markov decision process, classification, and integrations of symbolic search and statistical inference.*

Technologies to support learning and education, such as intelligent tutoring systems (ITSs), have a long history in artificial intelligence. AI methods have advanced considerably since those early days, and so have intelligent tutoring systems. Today, intelligent tutoring systems are in widespread use in K–12 schools and colleges and are enhancing the student learning experience (for example, Graesser et al. [2005]; Mitrovic [2003]; VanLehn [2006]). As a specific example, Cognitive Tutor mathematics courses are in regular use, about two-days a week, by 600,000 students a year in 2600 middle or high schools, and full-year evaluation studies of Cognitive Tutor algebra have demonstrated better student learning compared to traditional algebra courses (Ritter et al. 2007).

In recent years, a range of types of interactive educational technologies have also become prominent and widely used, including homework support and tutoring systems, science simulations and virtual labs, educational games, online resources, massive open online courses, and highly interactive web-based courses. Some have experimentally established learning benefits (for example, Bowen et al. [2012]; Lovett, Meyer, and Thille [2008]; Roschelle et al. [2010]). These systems are increasingly being instrumented to collect vast amounts of "Big Data" and more and more of it is freely available. DataShop, an open data repository at the Pittsburgh Science of Learning Center[1] (Koedinger et al. 2011), currently stores more than 350 data sets that include more than 200,000 student hours of data from thousands of students at an average of 10 seconds per action, yielding more than 90 million stored student actions.

Such data can be used to help advance our understanding of student learning and create better, more intelligent, interactive, engaging, and effective education. To do so requires advances in artificial intelligence and machine learning and in our theories of human intelligence and learning, especially the rich, knowledge-based learning flexibility that allows humans to develop expertise in so many complex domains. This work is often being pursued in the new fields of educational data mining (Romero and Ventura 2007, Baker and Yacef 2009) and learning analytics (Long and Siemens 2011).

In this article, we discuss the status and prospects of this new and powerful opportunity for data-driven development and optimization of educational technologies, focusing on intelligent tutoring systems and illustrating techniques especially in the context of cognitive tutors.

We begin by summarizing and illustrating the key functions of an intelligent tutoring system. We then discuss techniques for using data to develop ITS functionality without extensive knowledge engineering efforts and, ideally, with greater fidelity to student experience and consequent pedagogical effectiveness. Explicitly directed at accurate modeling of student learning, student engagement, and improved instruction, we next discuss techniques that optimize ITS functionality. We conclude with some future possibilities for data-driven ITS development and optimization.

A Summary and Illustration of the Key Functions of Intelligent Tutoring Systems

Intelligent tutoring systems both guide students through a curriculum of instructional activities in an outer loop and monitor step-by-step progress on an activity within an inner loop (VanLehn 2006). As shown in figure 1, the outer loop starts with selecting and presenting an activity to the student. Such activities are often multistep problems to solve, but may also include interactions in a simulation, game, or a dialogue. Figure 2 shows an example of a complex activity selected from an algebra curriculum unit on systems of linear equations where students use table, graphical, and symbolic representations to model a problem scenario and answer questions about it (Ritter et al. 2007). Once an activity is selected, the inner loop takes over and, as shown in figure 1, persists until the student has completed the activity. Within the inner loop, a tutor must decipher and evaluate each student action given the context of prior actions and a cognitive model of student reasoning and performance. For example, in figure 2b, the student has been filling in the table and most recently entered a mathematical expression ($.13t$) in a column he or she previously labeled "Current cost" and using her choice of t to represent time. The tutor uses the cognitive model to evaluate this action (in the context of a plan) and determines it is incorrect (it should be $.13t + 14.95$). In addition to evaluating student actions (the left branch of the inner loop in figure 1), an intelligent tutor can also suggest a next action when a student is stuck (the right branch). This suggestion may come in the form of a series of as-needed hints that get increasingly specific. In figure 2c, the student gets stuck on question 4 of the problem and clicks the hint button (2d). The tutor replies with an initial general hint to enter an equation (within the equation solving tool). To perform the evaluate and suggest functions, the tutor uses a cognitive model that represents possible solutions to the activity, infers how a student's input may relate to common misunderstandings, and predicts what feedback or hints will best help the student complete the activity. Figure 3 illustrates how a cognitive model can be used, through a plan recognition algorithm called model tracing (Ritter el al. 2007), to both evaluate a student's responses and suggest hints.

The results of model tracing of student input as he or she completes the activity are used to update (bottom of the outer loop in figure 1) an estimate of the student's skills and knowledge in the target domain. This information is then used to aid in activity selection (top of the outer loop). While many representations of student knowledge are possible, a simple yet effective model involves representing the curriculum by a set of skills and concepts, known as knowledge components (KCs). Then student knowledge is represented by the probability that the student has mastered each KC. These probability estimates can be updated by using a probabilistic model of student learning, such as by knowledge tracing (Corbett and Anderson 1995), which is essentially Bayesian filtering performed on a two-state hidden Markov model (HMM). Figure 2e shows an estimate of the student's understanding of the five knowledge components used in this curriculum unit. These estimates are used to select the next activity for the student.

Machine Learning and Data-Driven ITS Development

Historically, most intelligent tutoring systems have been built through extensive knowledge engineering and, ideally, cognitive task analysis to develop models of student and expert skill and performance. These models are then used to generate hints and feedback (inner loop of figure 1). In particular, two classes of effective tutors, cognitive tutors (for example, Ritter et al. [2007]) and constraint-based tutors (for example, Mitrovic [2003]), rely on knowledge representations, "production rules" or "constraints," that require extensive programming, expertise, and often empirical research to develop. In contrast, data-driven methods can enable more rapid development of new intelligent tutoring systems. We now present different data-driven symbolic and statistical machine-learning approaches for automated or semiautomated development of the key components and functionalities of intelligent tutoring systems as illustrated in figures 1–3.

SimStudent: Developing Cognitive Models by Demonstration and Tutoring

SimStudent is a theory of student learning instantiated in a software tool

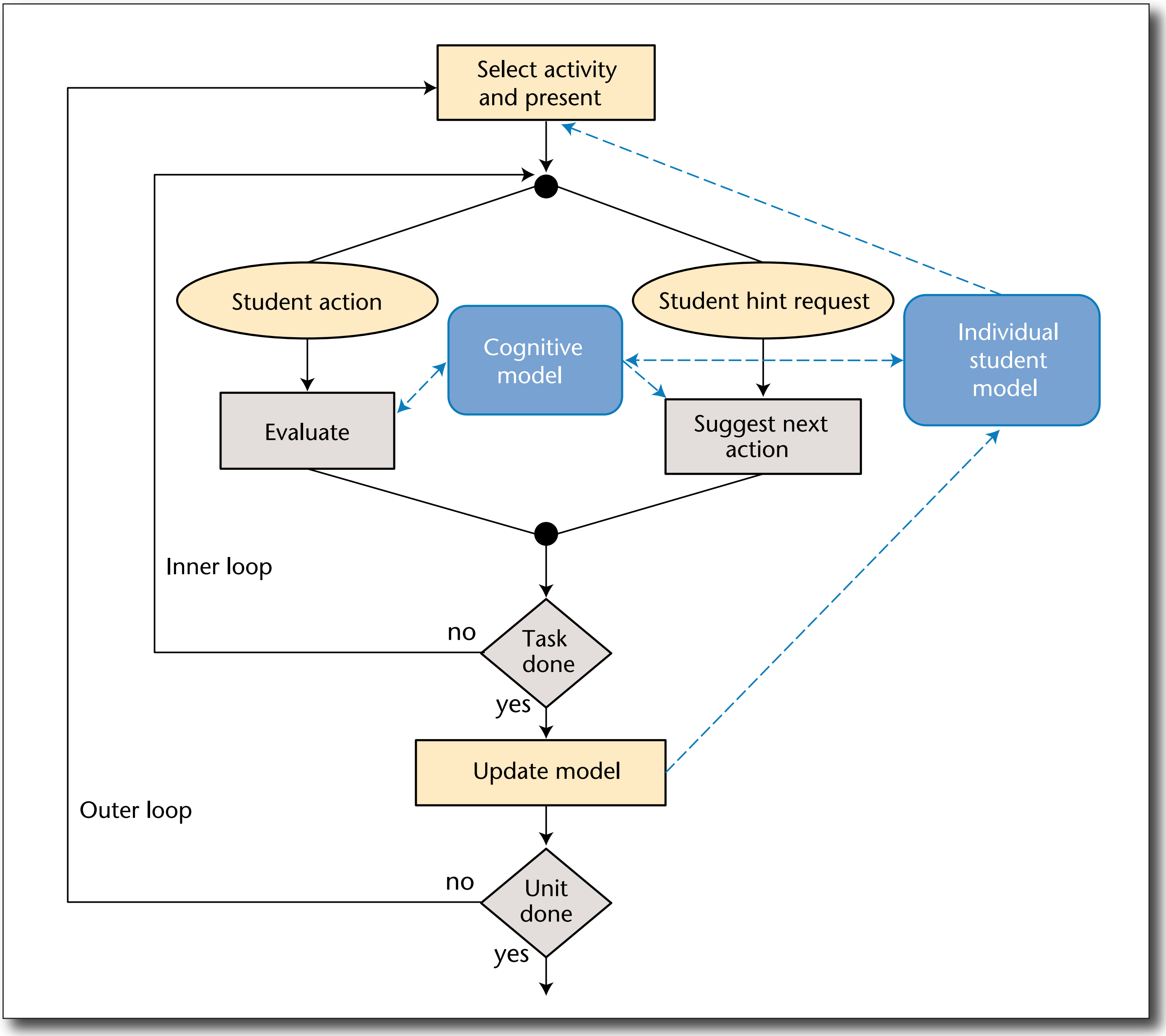
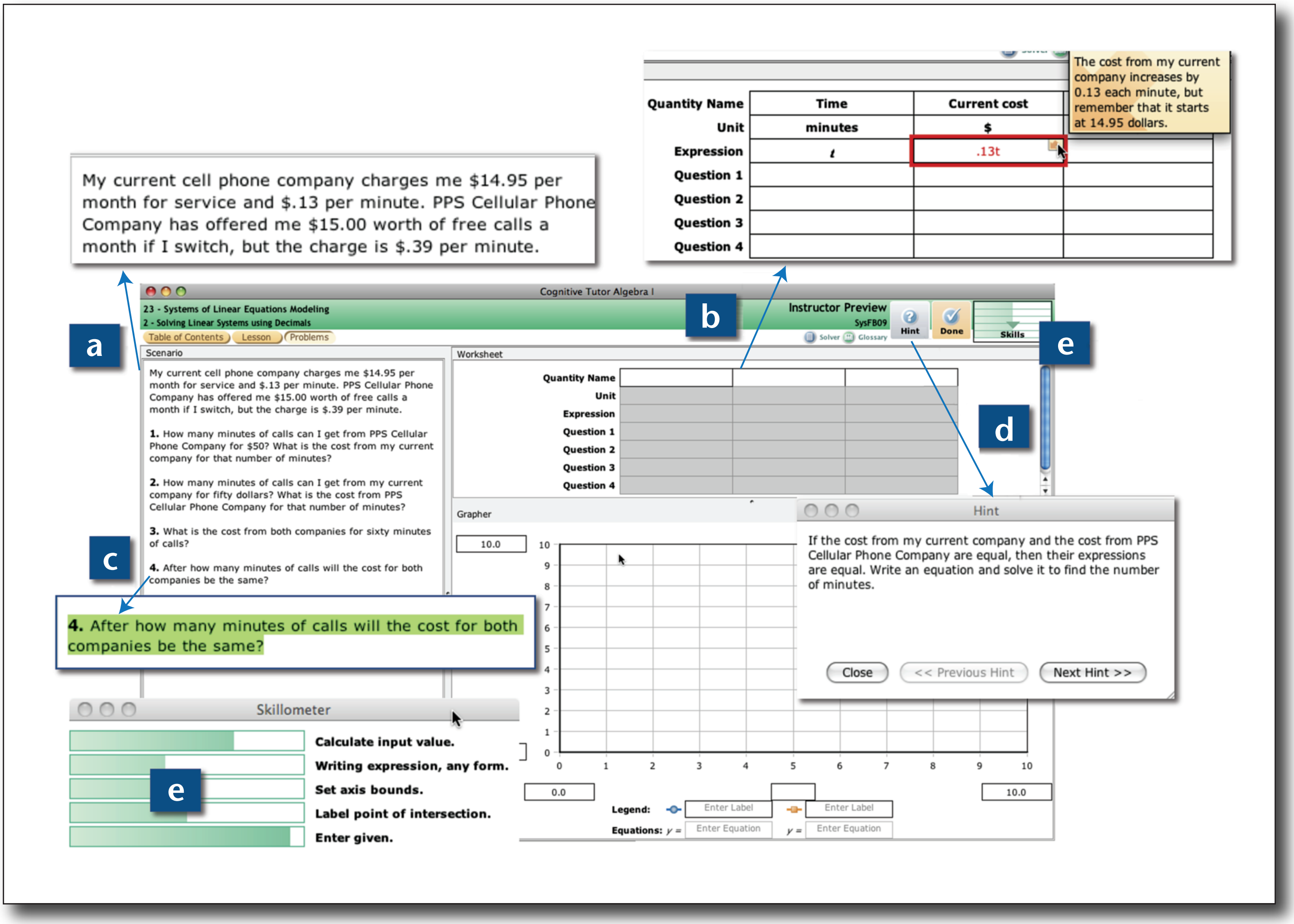


Figure 1. Key Functions of Intelligent Tutoring Systems.

Select, evaluate, suggest, and update (in rectangles) are supported by cognitive model and individual student model components (in rounded rectangles). They operate inside an across-activity "outer loop" and a within-activity "inner loop." Traditionally developed through knowledge engineering, these functions are increasingly being developed and optimized through data-driven machine learning.

that facilitates the development of cognitive models. A primary use is to allow non-AI-programmers to "program by tutoring" to create the central cognitive model component of an ITS. In this approach, authors first use cognitive tutor authoring tools (CTAT) (Aleven et al. 2009) to create a graphical user interface that students will use to solve tasks (for example, a table of rows for steps in an algebra equation solution). The author iteratively enters tasks into the interface (for example, an equation to solve) and then evokes SimStudent to solve each task. Initially, SimStudent has no relevant productions, so asks the author to demonstrate a step. The demonstration is used to induce a candidate production rule for accomplishing the step. On uture steps, previously induced production rules (which may be overly general) are used to generate candidate next steps and the author gives yes-no feedback on the correctness of the step. When the author states the step is incorrect, SimStudent relearns the production rule given the past history of demonstrations and feedback it has received and tries again until it either gets posi-



My current cell phone company charges me $14.95 per month for service and $.13 per minute. PPS Cellular Phone Company has offered me $15.00 worth of free calls a month if I switch, but the charge is $.39 per minute.

1. How many minutes of calls can I get from PPS Cellular Phone Company for $50? What is the cost from my current company for that number of minutes?

2. How many minutes of calls can I get from my current company for fifty dollars? What is the cost from PPS Cellular Phone Company for that number of minutes?

3. What is the cost from both companies for sixty minutes of calls?

4. After how many minutes of calls will the cost for both companies be the same?

Quantity Name	Time	Current cost	
Unit	minutes	$	
Expression	t	.13t	
Question 1			
Question 2			
Question 3			
Question 4			

The cost from my current company increases by 0.13 each minute, but remember that it starts at 14.95 dollars.

If the cost from my current company and the cost from PPS Cellular Phone Company are equal, then their expressions are equal. Write an equation and solve it to find the number of minutes.

Calculate input value.
Writing expression, any form.
Set axis bounds.
Label point of intersection.
Enter given.

Figure 2. A Problem Within the Algebra Cognitive Tutor.

This screen shot (with blowups) provides a concrete example of model tracing (a–d) and knowledge tracing (e) as implemented in an ITS. In (a) the student reads the problem statement and in (b) performs actions (filling in cells in table), which the tutor evaluates in comparison to the cognitive model and then provides feedback. (c) Later, the student reads question 4 and is stuck, (d) so she requests a hint. The cognitive model is run forward to generate a reasonable next step given the current solution state and the tutor suggests a corresponding hint (to set up an equation). (e) Student model updates are made based on student performance and these are used to select the next problem.

tive feedback or runs out of options. In the latter case, it asks the author for a demonstration of that step and induces a new production rule.

SimStudent employs multiple AI and machine-learning techniques to learn a rule-based production system (Li et al. 2012). Example problem and solution steps (for example, algebra equations) are used by probabilistic context-free grammar learning to generalize a hierarchical state representation that production rules manipulate. The if part of each production rule is acquired using a version space search for generalizing information retrieval paths and inductive logic programming for learning preconditions, which refine correctness and search control. The then part of production rules is acquired by an inductive search of function compositions that are consistent with prior action records. The acquired production system serves as the cognitive model component of an ITS that is used for all of its functions: evaluate, suggest, update, and select. SimStudent has been applied to learn cognitive models in many domains including algebra equation solving, stoichiometry, multicolumn addition and subtraction, tic-tac-toe, and fraction addition.

Hint Factory

The Hint Factory is a method of automatically generating context-specific hints by using previously collected student data (Barnes and Stamper 2008). The method is designed to be as specific as possible, derived on demand, and directed to the student's problem-solving goal, to provide the right type of help at the right time. In particular, the Hint Factory uses student attempt data to automatically evaluate

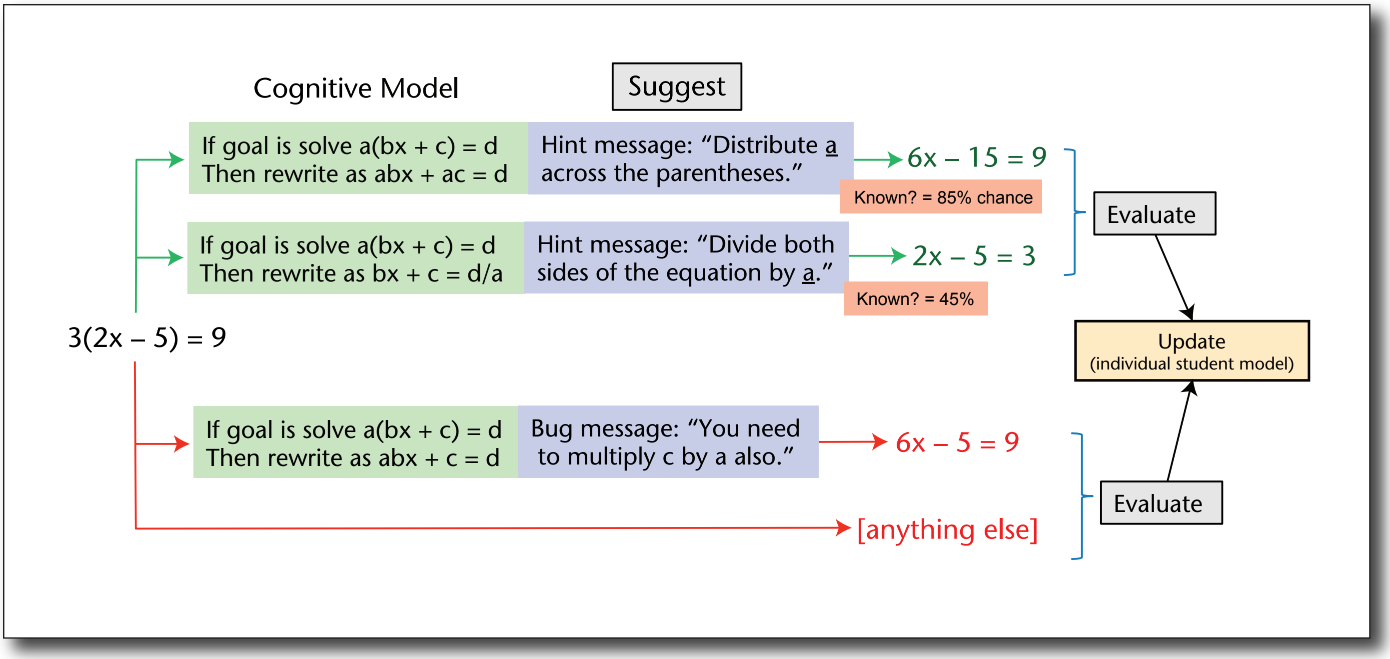


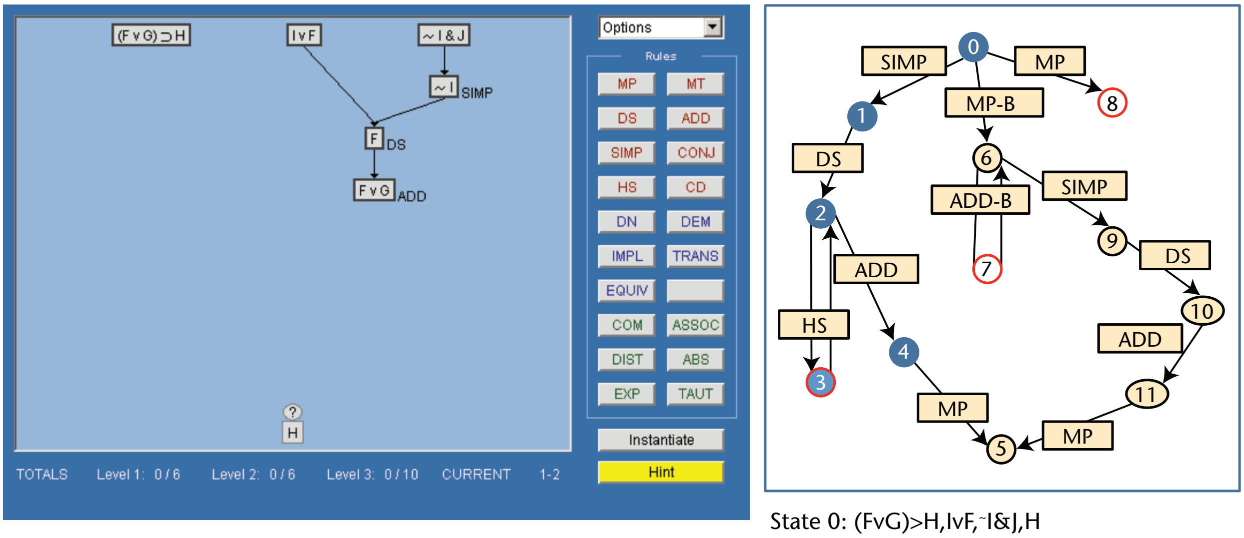
Figure 3. Example of the Use of Production Rules.

Production rules are used in a cognitive model (left-most boxes) for model tracing and knowledge tracing, which are specific implementations of the inner and outer loops in figure 1. In model tracing, the productions are used to evaluate student actions as correct (green or light gray arrows) or incorrect (red or dark gray arrows) with possible "bug" feedback (bottommost box) and to suggest possible next actions with associated hints (top right boxes). Knowledge tracing uses the evaluation of student actions in a Bayesian update of the individual student model of the chance a rule is known, which in turn is used to select future problems adapted to student needs.

student actions and to suggest next steps, that is, to provide hints within a problem. It achieves the inner loop of figure 1.

The Hint Factory provides direct, data-driven feedback in an environment where students can choose from a large space of actions to perform and many are correct. In order to deliver hints and feedback, the Hint Factory first constructs a graph of states and actions that represents all previous student approaches to a particular problem. Here the state describes what the student sees on the screen and the actions are what the student does. The state-action graph is transformed into a Markov decision process (MDP). An MDP is defined by its state set **S**, action set **A**, transition probabilities *T*, and a reward function *R* (Sutton and Barto 1998). A simple reward function is to provide a small negative reward for all nonsolution states: this encourages reaching the solution as efficiently as possible. Then the MDP is used to generate hints with the Hint Factory. The goal of using an MDP is to determine the best policy (that is, the best path through this graph) that corresponds to solving the given problem. This is achieved by calculating a value, the expected discounted sum of the rewards to be earned by following an optimal policy from state s, calculated recursively using value iteration. Once value iteration is complete, the optimal solution in the MDP corresponds to taking an expertlike approach to solving the given problem, where from each state the best action to take is the one that leads to the next state with the highest expected reward value (Barnes and Stamper 2008). The Hint Factory uses these values when a student is in a particular state to choose the next best state from which to generate a hint. When the hint button is pressed, the hint provider searches for the current state in the MDP and checks that a successor state exists. If it does, the successor state with the highest value is used to generate a hint sequence. A hint sequence refers to hints that are all derived based on the same current state. For each state, four distinct hints are generated. If a student requests a hint, then makes an error, and requests a hint again, the next hint generated is the next one in the current sequence. Once a student performs a correct step, the hint sequence is reset.

Barnes and Stamper (2008) demonstrated the feasibility of this approach on historical data, showing that extracted MDPs with the proposed hint-generating functions could provide correct next-step hints toward the problem solution more than 80 percent of the time. In a pilot study, Barnes and Stamper augmented a tutor to teach propositional logic with the Hint Factory and showed that students were able to solve more logic proof problems when hints were included. An example of the Hint Factory implemented in the logic tutor can be seen in figure 4. The figure shows a partially completed proof on the left



State 0: (FvG)>H,IvF,~I&J,H
State 1:(FvG)>H,IvF,~I&J, H, ~I
State 2:(FvG)>H,IvF,~I&J, H, ~I, F
State 3:(FvG)>H,IvF,~I&J, H, ~I, F, Error
State 4: :(FvG)>H,IvF,~I&J, H, ~I, F, FvG

Figure 4. Partial Proof with Hint Factory Added.

On the left is a partially completed proof in the logic tutor with the Hint Factory added. The student started with three given premises and the conclusion (State 0) and has performed four steps (states 1–4), whose state descriptions are in the lower right. The upper right shows a solution graph aggregated from past student solutions. The current student solution is traced against this graph; see the filled in circles for states 0–4. By comparing the student's current state (4) with the goal state (which is state 5), the Hint Factory can give students hints toward the next best state that is on path to the solution. In this case the student can reach the goal state by completing one more step (MP). When a student presses the hint button, a hint message is generated such as "Try using Modus Ponens(MP)."

and a small graph of previous attempts in the upper right. The states are represented with the numbers in circles. The start state is state 0, and the current student has completed the four steps shown in the lower right (filled-in states 1–4 in the graph). Error states are highlighted.

Since the Hint Factory is data driven, the system can be bootstrapped with expert solutions (Stamper, Barnes, and Croy 2010). The Hint Factory can evolve, providing at least some automatically generated hints initially and improving as additional expert and student problem attempts are added to the model.

The Hint Factory and MDP methods have also been used to augment tutors in other domains. Fossati and colleagues (2009) have used these MDP methods in the iList linked list tutor to deliver "proactive feedback" based on previous student work. Work is ongoing to build a Hint Factory to provide hints for novices in an open computer programming environment (Jin et al. 2012).

Detector Approach: Science Inquiry Tutor Example

In more ill-defined domains, it can be difficult to create an explicit cognitive model through typical knowledge-engineering processes, and it can be difficult even to conceptualize what an "item" or "practice opportunity" is. In recent years, data-driven approaches have proven useful for these domains as well. Machine learning has been used to develop the inner-loop evaluate functionality of an ITS in more ill-defined domains. In these cases, a "detector" or classifier is trained using human labels of desired and undesired student actions within an open-ended simulation or performance environment. The work of Sao Pedro et al. (2010) on developing a tutor for scientific inquiry illustrates this approach. In this approach, humans hand-labeled student data from use of a set of science microworlds, using text-based replays of segments of student interaction logs. The labels indicated whether an appropriate or inappropriate science inquiry strategy was present in the replayed segment, including designing controlled experiments, testing the stated hypothesis, and haphazard inquiry. Hand labels were validated for interrater reliability across multiple coders and then used as training labels for automated detectors of student science inquiry skill, using standard classification algorithms. The algorithm used a set of engineered features relevant to the timing and semantics of student actions, including features representing consistency and the number of times a specific action (such as changing variables or running the same experiment) occurred. The automated detectors were validated for effectiveness for new students and microworlds on different science topics and were found to be reasonably effective under these conditions. The detectors require some accumulation of data before they can respond effectively, but Sao Pedro and colleagues found the detectors could make accurate inferences about a specific student's inquiry skill after that student had run the simulation three times (each run takes under a minute), enabling reasonably rapid intervention. These detectors have now been built into automated interventions administered by a pedagogical agent, which evaluates student actions, identifies inappropriate strategies, and gives students feedback and advice on how to conduct more effective experimentation.

Other Future Possibilities for Automated ITS Construction

Another interesting issue is automated problem generation to provide suggested activities in the outer loop of figure 1. Recent work by Singh, Gulwani, and Rajamani (2012) drew upon results on generalized polynomial identity testing to automatically generate, from a given example algebra proof problem, a set of new similar but nontrivially different algebra proof problems. This and related techniques for automatically generating interesting related problems would help decrease the time required by domain experts, reduce concerns of cheating, and, in principle, lead to more finely constructed examples specifically designed to address a student's current misunderstanding.

Machine Learning and Data-Driven ITS Optimization

In addition to using data-driven methods to enable more rapid development of new intelligent tutoring systems, these methods can also be used to optimize the effectiveness of existing tutoring systems as we discuss in this section.

Optimizing the Cognitive Model

Recently, Koedinger, McLaughlin, and Stamper (2012) introduced an automated search process for optimizing cognitive model representations of student skill by hypothesizing alternative knowledge representations and testing them against data. Their approach was implemented using a version of the Learning Factors Analysis (LFA) algorithm (Cen, Koedinger, and Junker 2006). LFA makes use of the Q matrix representation (Tatsuoka 1983), a map of skills or knowledge components to tasks (for example, observed steps in problems). The KCs in a Q matrix are a latent variable simplification of the production rules or constraints in a cognitive model. LFA searches over Q matrices to find the one that best predicts student learning data, where that data is organized as success rate on knowledge components over time (encoded as number of opportunities to practice). The statistical prediction model is logistic regression with parameters for each student, KC, and KC by practice opportunity. Like a stepwise regression, LFA starts with a large set of candidate predictor variables (the so-called P matrix), which are hypothesized learning factors that may influence student performance difficulty or learning rate. Unlike stepwise regression, LFA uses specific symbolic operators (split, merge, and add) to create new variables (new knowledge component columns) thereby generating a huge space of possible Q matrices.

The version of LFA used in Koedinger, McLaughlin, and Stamper (2012) was run on 11 data sets in DataShop where human analysts had created alternative possible cognitive models (in the form of Q matrices) for that data. Instead of creating the P matrix directly by hand (as was done in prior LFA applications), here the P matrix is computed as the union of previously generated Q matrices associated with a data set created by learning scientists or domain experts. In this way DataShop facilitates a simple version of scientist crowdsourcing. LFA was then seeded with the simplest possible Q matrix, a single skill for all problem steps, and the split operator was used to generate new Q matrices. Figure 5 shows a simple example of a Q-matrix factor (*Sub*) being split by a P-matrix factor (*Neg-result*) resulting in the generation of a new model (Q') with two new KCs (*Sub-Pos* and *Sub-Neg)* replacing the old *Sub* KC. The LFA algorithm uses the input from a P matrix in a best first search process guided by a heuristic for model prediction accuracy (for example, the Akaike information criterion, AIC). It outputs a rank order of the most predictive Q matrices and, for each, the parameter estimates for student proficiency, KC difficulty, and KC learning rate.

Koedinger, McLaughlin, and Stamper (2012) applied the LFA algorithm to 11 data sets representing five domains and various technologies. Discovered models were compared to prior models using the root mean square error (RMSE) of predicted versus observed student correctness from a 10-fold item-stratified cross validation. In all 11 data sets the best

	Q		P		Q' split [Q,Neg result]	
Problem Step	Mult	Sub	Neg result	Order of Op	Sub-Pos	Sub-Neg
2*8–30 => 16-30	1	0	0	0	0	0
16–30 => -14	0	1	1	0	0	1
30–2*8 => 30-16	1	0	0	1	0	0
30–16 => 14	0	1	0	0	1	0

Figure 5. Example of a Q Matrix Being Split by a P-Matrix Faxtor.

The Q matrix and P matrix are mapped to problem steps. The resulting Q' matrix is created when *Sub* in the Q matrix is "split" by *Neg-result* from the P matrix.

machine-generated model outperformed both the original (in-use) model and the best hand-generated model. Because discovered models have much overlap with existing models, the overall improvement in prediction, while reliable, is small. However, differences between discovered and existing models provide a basis for meaningful improvements in tutor behavior, as we now describe.

One data set (Geometry9697) was used to illustrate how to interpret discovered models and guide tutor modification. Specific improvements of a discovered model over an original model were measured by percent reduction in cross-validated RMSE referenced to the original model's KCs. As expected, the original 15 KCs were largely unchanged by experts or LFA where, in fact, the discovered models replicate the base model. A substantial prediction error reduction was found in the 3 remaining KCs (5.5 to 11.1 percent). The improvement found in two of these was mostly captured in the hand-generated model and only slightly refined by LFA. However, the one remaining KC, *circle-radius*, realized a sizeable reduction from both the original to best-hand model (6 percent) and from the best-hand to best-machine model (4 percent). This discovery of LFA represents a genuine machine-based discovery not directly anticipated by human analysts.

A close look at the problem steps associated with the splits made to the original model revealed a distinction between forward versus backward application of a formula (for example, finding A in $A = 1/2bh$ versus finding b) that was unique to the circle area formula (that is, $A = \pi r^2$). The performance rate difference (80 percent forward versus 54 percent backward), parameter estimate differences (higher slopes and intercepts), and learning curve shape (smoother and declining) led LFA to discover that backward application of circle area (given the area, find the radius) is a separate skill from forward application. It is not only harder, but there is evidence that forward application practice does not transfer to backward application (or vice versa). However, for other area formulas, backward application is not a distinguishable skill. For these formulas, backward application is no harder than forward, and practice in one direction does transfer to better performance in the other. The unique feature of circle-area backwards is the need to evoke a square root operation. LFA thus produces practical recommendations for tutor optimization, in this example, (1) to change the update and selection functions to require separate mastery of forward and backward application of circle area but collapse this distinction for other area formulas and (2) to change the evaluate and suggest functions to specialize feedback and hint messages to the discovered challenge of seeing the relevance of square root. More generally, LFA has implications for theories of human learning providing an empirical methodology that demonstrates that student transfer of learning is often more narrow or broad than expected.

Better Statistical Student Models

The previous section described a method to automatically optimize the cognitive model, that is, the representation of the underlying curriculum content. Given such a representation, we can use statistical models to estimate and track student learning over the curriculum, as in the outer loop in figure 1. Some early, but still very successful, models of student learning, such as knowledge tracing (Corbett and Anderson 1995), used identical model parameters for all students. In such generic models the estimate of an individual's student progress still adapts to the individual's responses, but the parameters used to compute this estimate are the same for all students.

Other statistical models explicitly include a student variable with one parameter estimated per student. The additive factor, performance factor, and instructional factors analysis models (Chi et al. 2011) are all logistic regression models that include a single student parameter, which serves as a fixed offset in performance prediction. Similarly, Pardos and Heffernan (2010) extended the knowledge-tracing model to allow for different, student-specific initial probabilities of knowing the material. However, up to recently, almost no models attempt to account for wider possible variations among students, such as learning rates. Corbett and Anderson (1995) did describe fitting individual weights to each of the knowledge-tracing parameters, but this was done as a correction to the population parameters, rather than a direct parameter optimization for each student's data. In contrast, recent work by Lee and Brunskill (2012) allowed all parameters of a two-state, two-observation hidden Markov model (the KT student model) to vary by student and fit models for individual students directly. The authors were interested in whether significant variation in the different student model parameters existed among students, and if such differences existed, if they had significant implications for instruction.

In this work, Lee and Brunskill fit a separate knowledge-tracing model to each student's data. This involved fitting four parameters: initial probability of mastery, probability of transitioning from unmastered to mastered, probability of giving an incorrect answer if the student has mastered the skill, and probability of giving a correct answer if the student has not mastered the skill. Each student's model is fit using a combination of expectation maximization (EM) combined with a brute force search. It is well known that fitting a HMM suffers from an identifiability problem (for example, Beck and Chang [2007]) and that the resulting parameters may be implausible, such as if the probability of guessing correctly is higher than giving the right answer if the skill is mastered (Beck and Chang 2007). In addition, EM is only guaranteed to find a local optimum. To address identifiability, Baker et al. (2010) proposed performing a brute force search over a discretized set of parameters, which is computationally feasible in this model as there are only four parameters. Brute force search can also be used to enforce parameter plausibility, by constraining the search to a range of values considered plausible. This is the approach taken by Lee and Brunskill who used brute force search over the observation parameters to ensure plausibility, and used EM to compute the initial probability and learning probability. Lee and Brunskill also fit a single generic model to the combined set of all students' data.

Typically the quality of a student model is measured by a model's fit of the observed data, or its ability to predict student performance on a held out data set. Common methods include Bayesian information criterion (BIC) and cross-validated root mean square error. However, a key use of student models is to inform a tutor's instructional decisions: deciding the next activity to give to a student. Motivated by this, Lee and Brunskill proposed to evaluate modeling parameters on an individual level by whether this resulted in a significant change in the instructional decisions that would be made. Typically KT models are used in a mastery learning setting, where a tutor provides additional practice opportunities for a skill until the student's probability of mastering that skill (as tracked using the KT model) reaches a prespecified threshold (for example Corbett and Anderson [1995]). Given the same number of practice opportunities and a fixed trajectory of a student's responses, different model parameters will result in different probabilities of mastery. Therefore different model parameters could vary the amount of practice opportunities given to a student.

Lee and Brunskill evaluated whether the expected number of needed practice opportunities for a student to reach the mastery threshold varied significantly depending on whether a generic set of model parameters was used to evaluate mastery, or if model parameters fit to that individual's data were used. Computing this number can be viewed as policy evaluation in a continuous-state Markov decision process, where the state is the current probability of mastery, and the policy is to provide another practice opportunity if the state is below a threshold, or otherwise to halt teaching that skill. Lee and Brunskill performed this policy evaluation using a forward search algorithm. Note that in both cases the model parameters used to generate a student's responses was the individual model, since in real situations the student would be generating his responses based on his own internal knowledge and progression. The difference is in whether those observed responses are assumed (by the tutor) to have been generated using the set of generic model parameters or the individual's own parameters (see figure 6).

This approach was used to analyze data from more than 200 students using the ASSISTments system (Pardos and Heffernan 2011). Lee and Brunskill found that the fit individual parameters had a wide spread, and that the resulting parameters provided a significantly better fit of the observed data compared to a generic model (likelihood ratio test, $p < 0.0001$). Their results also showed that more than 20 percent of students would be expected to be given at least twice as many expected practice opportunities if the generic model was used compared to if their individual parameters were used, and 17 percent of students would be at ≤ 60 percent probability of mastery (compared to the threshold of 95 percent) when the generic model would declare the students as having mas-

Figure 6. The Difference Between Generic Model Parameters and the Individual's Own Parameters

Standard tutors use generic model parameters when selecting activities (left). On the right, the tutor knows the current student's specific parameters.

tered the material. This suggests that a significant number of students might be advanced before they have fully understood the material, or prevented from learning new material when ready, if a generic model of learning rate is used.

An implication of Lee and Brunskill's work is that model parameters that are fit to individual learning rates should yield better estimates of student learning and enable improved instruction. However, an interesting open question is how to perform this analysis in an online fashion. Lee and Brunskill's work was performed as a retrospective analysis. When tutors interact with real students, the tutors must perform this model fitting during tutoring. Indeed, this is a challenge for any model with student parameters. When only fitting a single parameter per student to specify the offset, prior researchers (Corbett and Anderson 1995, Pardos and Heffernan 2010) have proposed using the student's response on the first practice opportunity to fit a model. However, this approach will not suffice for fitting parameters that depend on the student's learning rate across skills. If prior diagnostic information exists about the student, or if these parameters are similar across many skills, then this model fitting could be performed in early stages, while using a generic model to inform instruction. More interesting is to consider how hierarchical models of student parameters might be trained and shared across multiple students and used to inform instruction. Such approaches could also be applicable beyond mastery learning approaches to teaching.

Modeling Engagement, Affect

Beyond the relatively well-defined construct of student knowledge, data-driven methods can also be used to model and adapt to constructs that have been historically difficult to define and model. One such area is individual differences in student engagement and affect (emotion in context — Corno [1986]). Engagement has long been seen as a critical factor in learning (compare with Corno and Mandinach [1983]), but in past decades has been seen as very difficult to operationalize in real time (Corno and Mandinach 1983), being a fairly ill-defined construct that can be seen as encompassing several constructs (for example several forms of engagement). However, automated detectors of engagement and affect have recently become an effective way to infer these constructs as a student is using online educational software such as intelligent tutoring systems.

One of the key challenges to making automated detectors of engagement and affect feasible for widespread use in intelligent tutoring systems has been the development of models that can infer these types of constructs solely from the types of interaction data readily available in the contexts where the software is used. Although reasonably effective models can be constructed using physical sensors such as cameras and posture sensors (see Calvo and D'Mello [2010] for review) these sensors can be challenging to deploy at scale in schools, due to issues of cost and breakage in these settings. However, multiyear efforts to understand and engineer the types of features associated with engagement and affect in interaction data have begun to produce automated detectors that are reliable and effective for these situations.

One of the first automated detectors of engagement was Baker, Corbett, and Koedinger's (2004) automated detectors of gaming the system, a disengaged behavior where students attempt to succeed in an educational task by systematically taking advantage of the intelligent tutor's feedback and help rather than by thinking through the material. In intelligent tutors, gaming the system can include misuse of hints to obtain answers or systematic guessing. Ground-truth training labels for this behavior can be obtained through systematic field observations (Baker, Corbett, and Koedinger 2004) or text replays (pretty-printed representations of student logs, designed to be easy to read; Baker and De Carvalho [2008]), checked across observers for inter-rater reliability. Then classification algorithms such as J48 decision trees are used to infer what actions the student was engaging in during the period of time labeled as gaming the system. Features such as repeated fast errors on poorly known skills were found to be indicative of gaming. The automated detectors were validated for effectiveness for new students and tutor lessons on different topics (within the same year-long mathematics curriculum), and were found to be reasonably effective (A'/AUC over 0.8) under these conditions. The automated detectors were first developed in the context of tutors for middle school mathematics, but detectors of gaming have now also been developed for other curricula by the same and other research groups.

A second disengaged behavior modeled in this fashion is off-task behavior, when the student completely disengages from the learning task, for instance by talking to another student about an unrelated topic. Though this behavior manifests in usage logs as inactivity, it can be inferred from the actions that occur immediately before and after. The first detector of this behavior, by Baker (2007), achieved a correlation of 0.55 between each student's predicted frequency of off-task behavior and the behavior''s frequency as noted by field observers, with cross-validation conducted at the student level. Baker's detector relied solely upon semantic actions within the interface and timing data. Cetintas and colleagues (2010) found that detection could be made more effective by also considering data from mouse movements.

A third disengaged behavior modeled in this fashion is careless errors, where a student knows the relevant skills to answer a question but produces an incorrect answer nonetheless. We infer training labels for detectors of this behavior by using the probability that a student knew the skill at a specific time computed from both student knowledge models (as discussed earlier) and data from future actions. For example, an error produced when a student had a 90 percent chance of knowing a skill, followed by two correct actions, is much more likely to represent a careless error than errors produced under different conditions. Once the training labels are obtained, detectors are developed to predict this behavior without using data from the future. Detectors developed in this fashion have been validated to transfer not only to new students, but to students from different countries (San Pedro, Baker, and Rodrigo 2011). Detectors of this type have been developed for intelligent tutors for both mathematics and science inquiry skill.

In addition to disengaged behaviors, a range of affective states have been modeled in intelligent tutoring systems (compare with Calvo and D'Mello [2010]), in recent years solely from student interaction with intelligent tutoring systems. The first such example is D'Mello et al. (2008), which modeled student affect in the AutoTutor intelligent tutoring system in a laboratory study. D'Mello and colleagues achieved better than chance agreement to ground-truth labels provided by human video coders, distinguishing students' frustration, boredom, confusion, and flow from each other. Conati and Maclaren (2009) and Sabourin, Mott, and Lester (2011) used a combination of interaction data and questionnaire data to infer a range of affective states. More recent work by Baker and colleagues (2012) found that better agreement to ground-truth labels could be achieved by explicitly using data from automated detectors of student disengaged behaviors when predicting affect. In specific, guessing was indicative of boredom and confusion, and failing to read hints was a negative predictor of engaged concentration.

In recent years, these detectors have been embedded into automated interventions that respond in various fashion to attempt to reengage students and mitigate the effects of negative affect. For example, an automated agent that addresses gaming behavior has been developed, based on an automated detector of gaming (Baker et al. 2006). This agent gave students supplementary exercises on material bypassed through gaming and displayed displeasure when the student games. In a classroom study in the USA, this agent reduced gaming by half and improved gaming students' learning relative to the control condition

(Baker et al. 2006). Recent work has also leveraged automated assessments of uncertainty (for example, Forbes-Riley and Litman [2011]), and affective states (compare with D'Mello et al [2009]; Arroyo et al. 2011), with promising results. For instance, D'Mello and colleagues (2009) found that supportive messages, given when negative affect occurred, improved affect for struggling students, while shake-up messages improved affect for more successful students.

Optimizing Instruction

A key aspect of the tutoring process is selecting the next activity to give to a student. This process can be considered as an optimization problem: what activity should be selected next, based on an estimate of what the student understands, in order to maximize some aspect of student learning, such as learning gains or engagement. This view allows the activity selection problem to draw on advances in sequential decision making under uncertainty. Early work by Beck, Woolf, and Beal (2000) and Murray, VanLehn, and Mostow (2004) used reinforcement learning and myopic decision utility maximization to inform instruction.

More recently, Chi et al. (2011) modeled physics tutoring as a Markov decision process to inform which type of activity to provide at certain junctures: whether to elicit student performance of a step (to practice the underlying skill) or tell the student how to perform the step (as an example to learn from), and whether to ask a student to justify a particular step of reasoning or to skip the request for justification. Chi et al. considered a rich set of features to model the student's state and performed automatic feature selection by identifying features associated with policies leading to high learning gains in a training set. Chi et al. found that an MDP policy that used these features and optimized to maximize expected learning gains resulted in higher empirical learning gains in a new lab student experiment compared to an MDP policy designed to minimize learning gains. The focus of Chi et al.'s work was on seeing if the tutorial policy used during microstep tutoring could influence learning gains, so this control policy was a reasonable comparison. However, the authors call for future work and, in particular, there is a need to see whether MDPs provide a significant benefit over other stronger control methods for making instructional decisions.

Recently several researchers (for example Brunskill and Russell [2010]; Brunskill et al. [2010]; Rafferty et al. [2011]) have taken an alternate approach of modeling the student state as being partially observable, making instructional decisions an instance of partially observable MDP (POMDP) planning. Many current tutoring systems do model student state as being partially observable, but such approaches often take (quite successfully) a myopic approach to planning. When there are multiple different activities, and there exists prerequisite structure about the skills in the curriculum, a myopic approach is not optimal. Performing POMDP planning in such domains is generally computationally challenging, since a curriculum may consist of many skills, each of which may be known or not known, and the skills are not independent. However, Brunskill and Russell (2010) provided an efficient algorithm for planning in such domains that have prerequisite structure, and showed in simulations that the resulting solutions could be substantially better than myopic approaches. Brunskill et al. (2010) also used POMDP planning to inform problem selection for students sharing a very simple groupware mathematical game. Their classroom pilot found that tailoring activities to individual students led to less instances of a single student consistently winning, suggesting potential benefits for engagement.

Although these initial findings are encouraging, there remains significant work to be done on optimizing instructional design. More comparisons need to be made to existing state of the art approaches. There also exists more machine-learning and artificial intelligence research to be done to handle the huge state, action, and observation (or feature) spaces involved and to close the loop of informing model design by instructional policies. To accomplish this work, partnerships between learning science and machine-learning researchers are likely to be particularly effective. Learning sciences researchers bring new and existing methods from psychology or education that can be built upon or used as strong comparisons to new approaches that may be developed in tandem with machine-learning researchers expert in algorithms for analyzing and optimizing policies for large, many-featured domains (such as education).

Conclusions

In recent years, there has been increasing work to leverage data to optimize and redesign intelligent tutoring systems. Within this article, we discuss a range of recent work illustrating this potential in a range of areas. Many of the examples are drawn from work with cognitive tutors, a mature platform used by large numbers of students, but as discussed throughout the article, there are relevant examples in a number of other research groups and platforms as well. The kinds of data-driven development and optimization techniques we have illustrated are relevant for both inner and outer loop functions of intelligent tutoring systems. Some of this work enhances existing functionality. For example, when Learning Factors Analysis is given student learning data from use of an ITS, it generates an improved cognitive model that better matches student performance. The improved cognitive model can be used to optimize the suggest, update, and select functions of the tutor. Similarly, Lee and Brunskill (2012) used student ITS data to build a better student model that improves the update function, which in turn could improve the effectiveness of the select function.

Beyond this, entirely new functionality can be supported through data-driven approaches. When automated detectors of student engagement and affect are developed, it then becomes possible to evaluate these aspects of the student in real-time, information that can be used to select different activities for the student — for example, through selecting alternate exercises designed to reengage students or mitigate the negative impacts on learning that disengagement and negative affect can cause. When the Hint Factory is given data on a dense set of alternative solution paths for a problem, it can perform the suggest function of generating next-step hints. When SimStudent is given data on problem solution demonstrations and feedback on its solution attempts, it learns new knowledge representation structures, particularly domain representations and production rules. These produce the cognitive model component of an ITS that is used for all of its functions: evaluate, suggest, update, and select.

As a whole, then, there are several ways that data can enhance the functionality of intelligent tutoring systems, leading to more personalized instruction. This work remains in early stages; every year, new approaches and uses for tutor data emerge, and new applications for enhancing intelligent tutors become possible. We have framed a set of challenges and some preliminary solutions toward the vision of fully optimized and personalized learning. The recent excitement and growth in online learning has the potential to produce the data needed to pursue this vision. The effectiveness of online courses can be vastly improved, but it will require going beyond compelling lecture videos and short follow-up questions to address learning by doing. Missing from today's most popular online courses and learning resources are complex problem-solving tasks, more open-ended interfaces, and AI back ends that can interpret students' step-by-step progress on these complex tasks. These changes will enrich the data and the opportunities for data-driven improvement. In general, meeting the challenges inherent in the vision of personalized learning will require richer data sources and advances in AI. Doing so could revolutionize educational practice.

Acknowledgements

Thanks to the National Science Foundation #SBE-0836012 for funding of LearnLab at the Pittsburgh Science of Learning Center (http://learnlab.org).

Note

1. See learnlab.org/datashop.

References

Aleven, V.; McLaren, B.; Sewall, J.; and Koedinger, K. R. 2009. Example-Tracing Tutors: A New Paradigm for Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education* 19(2): 105–154.

Arroyo, I.; Woolf, B. P.; Cooper, D. G.; Burleson, W.; and Muldner, K. 2011. The Impact of Animated Pedagogical Agents on Girls' and Boys' Emotions, Attitudes, Behaviors, and Learning. In *Proceedings of the 11th IEEE Conference on Advanced Learning Technologies*. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Baker, R. S. J. d. 2007. Modeling and Understanding Students' Off-Task Behavior in Intelligent Tutoring Systems. In *Proceedings of the ACM CHI 2007 Conference on Human Factors in Computing Systems,* 1059–1068. New York: Association for Computing Machinery.

Baker, R. S.; Corbett, A. T.; and Koedinger, K. R. 2004. Detecting Student Misuse of Intelligent Tutoring Systems. In *Proceedings of the 7th International Conference on Intelligent Tutoring Systems,* Lecture Notes in Computer Science 3220, 531–540. Berlin: Springer.

Baker, R. S. J. d.; Corbett, A. T.; Koedinger, K. R.; Evenson, S. E.; Roll, I.; Wagner, A. Z.; Naim, M.; Raspat, J.; Baker, D. J.; and Beck, J. 2006. Adapting to When Students Game an Intelligent Tutoring System. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems,* Lecture Notes in Computer Science 4053, 392–401. Berlin: Springer.

Baker, R. S. J. d.; Corbett, A. T.; Gowda, S. M.; Wagner, A. Z.; MacLaren, B. M.; Kauffman, L. R.; Mitchell, A. P.; and Giguere, S. 2010. Contextual Slip and Prediction of Student Performance After Use of an Intelligent Tutor. In *Proceedings of the 18th Annual Conference on User Modeling, Adaptation, and Personalization,* Lecture Notes in Computer Science 6075, 52–63. Berlin: Springer.

Baker, R. S. J. d., and De Carvalho, A. M. J. A. 2008. Labeling Student Behavior Faster and More Precisely with Text Replays. Paper presented at the 1st International Conference on Educational Data Mining, 20–21 June.

Baker, R. S. J. d.; Gowda, S. M.; Wixon, M.; Kalka, J.; Wagner, A. Z.; Salvi, A.; Aleven, V.; Kusbit, G.; Ocumpaugh, J.; and Rossi, L. 2012. Sensor-Free Automated Detection of Affect in a Cognitive Tutor for Algebra. Paper presented at the 5th International Conference on Educational Data Mining, Chania, Greece, 19–21 June.

Baker, R. S. J. d., and Yacef, K. 2009. The State of Educational Data Mining in 2009: A Review and Future Visions. *Journal of Educational Data Mining* 1(1): 3–17.

Barnes, T., and Stamper, J. 2008. Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems,* 373–382. Berlin: Springer.

Beck, J. E., and Chang, K.-M. 2007. Identifiability: A Fundamental Problem of Student Modeling. In *Proceedings of the 11th International Conference on User Modeling,* Lecture Notes in Computer Science. Berlin: Springer.

Beck, J.; Woolf, B.; and Beal, C. 2000. Advisor: A Machine Learning Architecture for Intelligent Tutor Construction. In *Proceedings of the 17th AAAI Conference on Artificial Intelligence* (AAAI), 552–557. Menlo Park, CA: AAAI Press.

Bowen, W. G.; Chingos, M. M.; Lack, K. A.; and Nygren, T. I. 2012. Interactive Learning Online at Public Universities: Evidence from Randomized Trials. New York: Ithaka S+R. (www.sr.ithaka.org/sites/default/files/ reports/sr-ithaka-interactive-learning-online-at-public-universities.pdfand-nid=464)

Brunskill, E., and Russell, S. 2010. Rapid: A Reachable Any-time Planner for Imprecisely-Sensed Domains. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. Redmond, WA: AUAI Press.

Brunskill, E.; Garg, S.; Tseng, C.; Pal, J.; and Findalter, L. 2010. Evaluating an Adaptive Multi-User Educational Tool for Low-Resource Regions. Paper presented at the International Conference on Information and Communication Technologies and Development, London, UK, 13–16 December.

Calvo, R. A., and D'Mello, S. K. 2010. Affect Detection: An Interdisciplinary Review of Models, Methods, and Their Applications. *IEEE Transactions on Affective Computing* 1(1): 18–37.

Cen, H.; Koedinger, K. R.; and 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,* 164–175. Berlin: Springer.

Cetintas, S.; Si, L.; Xin, Y.; and Hord, C. 2010. Automatic Detection of Off-Task Behaviors in Intelligent Tutoring Systems with Machine Learning Techniques. *IEEE Transactions on Learning Technologies* 3(3): 228–236.

Chi, M.; VanLehn, K.; Litman, D.; and Jordan, P. 2011. An Evaluation of Pedagogical Tutorial Tactics for a Natural Language Tutoring System: A Reinforcement Learning Approach. *International Journal of Artificial Intelligence in Education* 21(2): 83–113.

Conati, C., and MacLaren, H. 2009. Empirically Building and Evaluating a Probabilistic Model of User Affect. *User Modeling and User-Adapted Interaction* 19(3) 267–303.

Corbett, A. T., and Anderson, J. R. 1995. Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. *User Modeling and User-Adapted Interaction* 4(4) 253–278.

Corno, L. 1986. The Metacognitive Control Components of Self-Regulated Learning. *Contemporary Educational Psychology* 11(4): 333–346.

Corno, L., and Mandinach, E. B. 1983. The Role of Cognitive Engagement in Classroom Learning and Motivation. *Educational Psychologist* 18(2): 88–108.

D'Mello, S. K.; Craig, S. D.; Fike, K.; and Graesser, A. C. 2009. Responding to Learners' Cognitive-Affective States with Supportive and Shakeup Dialogues. In *Human-Computer Interaction. Ambient, Ubiquitous and Intelligent Interaction,* ed. J. A. Jacko, 595–604. Berlin: Springer.

D'Mello, S. K.; Craig, S. D.; Witherspoon, A. W.; McDaniel, B. T.; and Graesser, A. C. 2008. Automatic Detection of Learner's Affect from Conversational Cues. *User Modeling and User-Adapted Interaction* 18(1–2): 45–80.

Forbes-Riley, K., and Litman, D. 2011. Benefits and Challenges of Real-Time Uncertainty Detection and Adaptation in a Spoken Dialogue Computer Tutor. *Speech Communication* 53(9–10): 1115–1136.

Fossati, D.; Di Eugenio, B.; Ohlsson, S.; Brown, C.; Chen, L.; and Cosejo, D. 2009. I Learn from You, You Learn from Me: How to Make Ilist Learn from Students. In *Proceedings of 14th International Conference on Artificial Intelligence in Education,* 186–195. Amsterdam: IOS Press.

Graesser, A. C.; Chipman, P.; Haynes, B.; and Olney, A. M. 2005. Autotutor: An Intelligent Tutoring System with Mixed-Initiative Dialogue. *IEEE Transactions on Education* 48(4): 612–618.

Jin, W.; Barnes, T.; Stamper, J.; Eagle, M.; Johnson, M.; and Lehmann, L. 2012. Program Representation for Automatic Hint Generation for a Data-Driven Novice Programming Tutor. In *Proceedings of the Eleventh International Conference on Intelligent Tutoring Systems,* 304–309. Berlin: Springer.

Koedinger, K. R.; Baker, R. S. J. d.; Cunningham, K.; Skogsholm, A.; Leber, B.; and Stamper, J. 2011. A Data Repository for the EDM Community: The PSLC DataShop. In *Handbook of Educational Data Mining*, ed. C. Romero, S. Ventura, M. Pechenizkiy, and R. S. J. d. Baker. Boca Raton, FL: CRC Press.

Koedinger, K. R.; Mclaughlin, E. A.; and Stamper, J. C. 2012. Automated Student Model Improvement. Paper presented at the 5th International Conference on Educational Data Mining, Chania, Greece, 19–21 June.

Lee, J. I., and Brunskill, E. 2012. The Impact on Individualizing Student Models on Necessary Practice Opportunities. Paper presented at the 5th International Conference on Educational Data Mining, Chania, Greece, 19–21 June.

Li, N.; Schreiber, A.; Cohen, W.; and Koedinger, K. R. 2012. Efficient Complex Skill Acquisition Through Representation Learning. Advances in Cognitive Systems 2(December): 149-166.

Long, P., and Siemens, G. 2011. Penetrating the Fog: Analytics in Learning and Education. *Educause Review* 46(5): 31–40.

Lovett, M.; Meyer, O.; and Thille, C. 2008. The Open Learning Initiative: Measuring the Effectiveness of the OLI Learning Course in Accelerating Student Learning. *Journal of Interactive Media in Education* (May).

Mitrovic, A. 2003. An Intelligent SQL Tutor on the Web. *International Journal of Artificial Intelligence in Education* 13(2–4): 197–243.

Murray, B.; VanLehn , K.; and Mostow, J. 2004. Looking Ahead to Select Tutorial Actions: A Decision-Theoretic Approach. *International Journal of Artificial Intelligence in Education* 14(3–4): 235–278.

Pardos, Z. A., and Heffernan, N. T. 2010. Modeling Individualization in a Bayesian Networks Implementation of Knowledge Tracing. In *Proceedings of the 18th International Conference on User Modeling, Adaptation and Personalization,* Lecture Notes in Computer Science, 255–266. Berlin: Springer.

Rafferty, A.; Brunskill, E.; Griffiths,T.; and Shafto, P. 2011. Faster Teaching by POMDP Planning. In *Proceedings of 15th International Conference on Artificial Intelligence in Education.* Amsterdam: IOS Press.

Ritter S.; Anderson, J. R.; Koedinger, K. R.; and Corbett, A. 2007. Cognitive Tutor: Applied Research in Mathematics Education. *Psychonomic Bulletin and Review* 14(2): 249–255.

Roschelle, J.; Shechtman, N.; Tatar, D.; Hegedus, S.; Hopkins, B.; Empson, S.; Knudsen, J.; and Gallagher, L. 2010. Integration of Technology, Curriculum, and Professional Development for Advancing Middle School Mathematics: Three Large-Scale Studies. *American Educational Research Journal* 47(4): 833–878.

Romero, R., and Ventura, S. 2007. Educational Data Mining: A Survey from 1995 to 2005. *Expert Systems with Applications* 33(1): 135–146.

Sabourin, J.; Mott, B.; and Lester, J. 2011. Modeling Learner Affect with Theoretically Grounded Dynamic Bayesian Networks. In *Proceedings of the 4th International Conference on Affective Computing and Intelligent Interaction,* Lecture Notes in Computer Science, 286–295. Berlin: Springer.

San Pedro, M. O. C.; Baker, R.; and Rodrigo, M. M. 2011. Detecting Carelessness Through Contextual Estimation of Slip Probabilities Among Students Using an Intelligent Tutor for Mathematics. In *Proceedings of 15th International Conference on Artificial Intelligence in Education,* 304–311. Amsterdam: IOS Press.

Sao Pedro, M. A.; Baker, R. S. J. d.; Montalvo, O.; Nakama, A.; and Gobert, J. D. 2010. Using Text Replay Tagging to Produce Detectors of Systematic Experimentation Behavior Patterns. Paper Presented at the 3rd International Conference on Educational Data Mining. Pittsburgh, PA, 11–13 June.

Singh, R.; Gulwani, S.; and Rajamani, S. 2012. Automatically Generating Algebra Problems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence.* Palo Alto, CA: AAAI Press.

Stamper, J.; Barnes, T.; and Croy, M. 2010. Enhancing the Automatic Generation of Hints with Expert Seeding. In *Proceeding of the 10th International Conference on Intelligent Tutoring Systems,* Vol. II, 31–40. Berlin: Springer.

Sutton, S., and Barto, A. 1998. *Reinforcement Learning: An Introduction.* Cambridge, MA: The MIT Press.

Tatsuoka, K. K. 1983. Rule Space: An Approach for Dealing with Misconceptions Based on Item Response Theory. *Journal of Educational Measurement* 20(4): 345–354.

VanLehn, K. 2006. The Behavior of Tutoring Systems. *International Journal of Artificial Intelligence in Education* 16(3): 227–265.

Ken Koedinger is a professor in the Human-Computer Interaction Institute and the Psychology Department at Carnegie Mellon University. He directs LearnLab.org, which leverages cognitive and computational approaches to support researchers in investigating the instructional conditions that cause robust student learning.

Emma Brunskill is an assistant professor in the Computer Science Department at Carnegie Mellon University. She is also affiliated with the Machine Learning Department. Her research lies in artificial intelligence and machine learning, where she focuses on novel methods to automatically make good sequences of decisions under uncertainty. She is particularly interested in applications of this work to intelligent tutoring systems and healthcare.

Ryan S. J. d. Baker is the Julius and Rosa Sachs Distinguished Lecturer at Columbia University Teachers College. He is also president of the International Educational Data Mining Society and uses data mining methods to study student engagement and robust learning in educational software.

Elizabeth A. McLaughlin is a research associate at the Human-Computer Interaction Institute at Carnegie Mellon University. Her research interests include cognitive science and the application of cognitive science principles to educational technology.

John Stamper is a member of the research faculty at the Human-Computer Interaction Institute at Carnegie Mellon University. He is also the technical director of the Pittsburgh Science of Learning Center DataShop (pslcdatashop.org). His primary areas of research include educational data mining and intelligent tutoring systems.