Computational Models of Human Learning: Applications for Tutor Development, Behavior Prediction, and Theory Testing

Christopher J. MacLellan

CMU-HCII-17-108

August, 2017

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This work was supported in part by a Graduate Training Grant awarded to Carnegie Mellon University by the Department of Education (#R305B090023 and #R305A090519), by the Pittsburgh Science of Learning Center, which is funded by the NSF (#SBE-0836012), and two National Science Foundation Awards (#DRL-0910176 and #DRL-1252440). I would also like to thank Carnegie Learning, Inc. for providing the Cognitive Tutor data that supported this thesis.

Keywords: Computational Modeling, Science of Learning, Educational Technology

For my family

Abstract

Intelligent tutoring systems are effective for improving students' learning outcomes (Bowen et al., 2013; Koedinger & Anderson, 1997; Pane et al., 2013). However, constructing tutoring systems that are pedagogically effective has been widely recognized as a challenging problem (Murray, 1999, 2003). In this thesis, I explore the use of computational models of apprentice learning, or computer models that learn interactively from examples and feedback, to support tutor development. In particular, I investigate their use for authoring expert-models via demonstrations and feedback (Matsuda et al., 2014), predicting student behavior within tutors (VanLehn et al., 1994), and for testing alternative learning theories (MacLellan, Harpstead, Patel, & Koedinger, 2016).

To support these investigations, I present the Apprentice Learner Architecture, which posits the types of knowledge, performance, and learning components needed for apprentice learning and enables the generation and testing of alternative models. I use this architecture to create two models: the DECISION TREE model, which non- incrementally learns when to apply its skills, and the TRESTLE model, which instead learns incrementally. Both models both draw on the same small set of prior knowledge for all simulations (six operators and three types of relational knowledge). Despite their limited prior knowledge, I demonstrate their use for efficiently authoring a novel experimental design tutor and show that they are capable of achieving human-level performance in seven additional tutoring systems that teach a wide range of knowledge types (associations, categories, and skills) across multiple domains (language, math, engineering, and science).

I show that the models are capable of predicting which versions of a fraction arithmetic and box and arrows tutors are more effective for human students' learning. Further, I use a mixed-effects regression analysis to evaluate the fit of the models to the available human data and show that across all seven domains the TRESTLE model better fits the human data than the DECISION TREE model, supporting the theory that humans learn the conditions under which skills apply incrementally, rather than non-incrementally as prior work has suggested (Li, 2013; Matsuda et al., 2009). This work lays the foundation for the development of a Model Human Learner— similar to Card, Moran, and Newell's (1986) Model Human Processor—that encapsulates psychological and learning science findings in a format that researchers and instructional designers can use to create effective tutoring systems.

Acknowledgments

This research would not have been possible without my family, who inspired me to pursue my dreams and provided me with the resources and support that I needed to do so. I am particularly thankful for my partner, Caitlin Tenison, who listened to all my nascent ideas and helped me find the ones worth pursuing.

This work also would not have been possible without my professional mentors, who have guided my thinking and are role models that I aspire to emulate. I am enormously grateful to Pat Langley for allowing me the pleasure of working with him, even though he had plans to leave the country when we first met. I am also grateful to Ken Koedinger, who was always patient with me and gave me the support, guidance, and encouragement I needed. I could not imagine a better advisor. Finally, I am grateful to Vincent Aleven, John Anderson, David Klahr, Sharon Carver, Noboru Matsuda, Douglas Fisher, and all the others who have helped me on my journey.

I am also grateful to all of my friends, who made graduate school a more enjoyable experience. I am especially grateful for Queenie Kravitz, who was a constant source of guidance and support, Erik Harpstead, who was always willing to listen to my ideas and made some of my best work possible, and Jonathan McBride, who kept me going through some of my rough patches. I also could not have made it without the support of my PhD cohort (Jenny, Dan, Nikola, Anthony, Tatiana, Brandon, David), who always understood what I was going through in a way that no one else did, Beka and Jeff, who adopted me as an officemate, and the other HCII and PIER PhD students, who formed my second family. Lastly, I would be remiss if I did not thank the baristas at Tazza D'Oro and Commonplace coffee who introduced me to good espresso and greeted me to work on a daily basis.

Contents

List of Figures

List of Tables

Preface

In this dissertation, I explore the use of computational models of learning from examples and correctness feedback, what I call *apprentice learner* models, to aid in the design, building, and testing phases of tutor development. I also explore the use of human data, collected using tutoring systems, for evaluating apprentice learner models and guiding a search for models that better align with human behavior. The work presented was conducted while I was a graduate student at Carnegie Mellon University.

All of the results in this thesis are new, unpublished work; however, some portions of the text have been taken from prior publications. In particular, some of the text in Chapter 4 has been taken from work that was published in collaboration with Erik Harpstead, Eliane Stampfer Wiese, Menfan Zou, Noboru Matsuda, Vincent Aleven, and Ken Koedinger (MacLellan et al., 2015). In this work, I was the lead investigator responsible for problem framing, study design, data collection, analysis, and manuscript preparation. Noboru met with me multiple times to help frame the work and provided insight into how to evaluate the cognitive models. Mengfan was an undergraduate research intern who worked with me to developed the original interface for the experimental design tutor. Eliane and Vincent provided feedback on the original manuscript. Finally, Ken Koedinger (my advisor) supervised all phases of the work and provided support with problem framing and manuscript preparation.

A portion of the text from chapter 5 was published in collaboration with Erik Harpstead, Rony Patel, and Ken Koedinger (MacLellan, Harpstead, Patel, & Koedinger, 2016). In this work, I was the lead investigator responsible for problem framing, architecture development, simulation design, data collection, analysis, and manuscript preparation. Erik Harpstead assisted me in framing the paper for the Educational Data Mining community and in manuscript preparation. Rony Patel developed the tutor that collected some of the human data, supported me in designing the simulation study, and helped to prepare the final version of the manuscript. Finally, Ken Koedinger supervised the entire process and provided support in framing the work and preparing the manuscript.

Finally, it is worth noting that Erik Harpstead and I have published multiple papers related to the Trestle algorithm (MacLellan, Harpstead, Aleven, & Koedinger, 2016) and the RumbleBlocks domain (Christel et al., 2012). Across these works, Erik and I have been equal partners. However, we have roughly divided our contributions along methodological and application lines. Erik has taken the lead on papers related to better understanding RumbleBlocks (and more broadly educational games) and providing insights to instructional designers and game developers. In contrast, I have taken the lead on method development and evaluation papers and on exploring the use of our methods for better understanding human learning.

Chapter 1

Motivation

Intelligent tutoring systems have been shown to improve student learning across multiple domains (Beal et al., 2007; Graesser et al., 2001; Koedinger & Anderson, 1997; Mitrovic et al., 2002; Ritter et al., 2007; VanLehn, 2011), but designing and building tutoring systems that are pedagogically effective is difficult and expensive (Murray, 2005). In an ideal world, tutor design is an iterative process that consists of multiple phases of design, building, and testing. However, each phase of this design process requires time, expertise, and resources to execute properly, which, in general, makes tutor development a cost prohibitive endeavor (Murray, 1999). As a result, many researchers have created tools to support the tutor development process (Aleven, McLaren, Sewall, & Koedinger, 2006; Murray, 1999, 2003, 2005; Sottilare & Holden, 2013). Unfortunately, these tools provide little support for the design and testing phases of development and instead focus primarily on supporting the building phases. They provide capabilities for authoring interfaces and domain content but give little insight into what content should be authored or tutor effectiveness. Further, while existing tutor authoring tools have been shown to reduce the expertise requirements and time needed to build a tutor (e.g., Example-Tracing has been shown to reduce authoring time by as much as 4 times, Aleven et al., 2009), they still struggle to support non-programmers trying to build tutors for complex domains (MacLellan et al., 2015). Thus, how technology can support all three phases of the tutor development process (designing, building, and testing) remains an open research question.

To develop technology to support these phases, I first looked at current tutor development practices. Instructional designers begin the process in the design phase, where they must address two high-level questions: what is the material to be taught and how should it be presented? In theory, designers should work with subject-matter experts to identify relevant domain content (e.g., using Cognitive Task Analysis techniques, Clark et al., 2008) and draw on prior science of learning findings (e.g., from the Knowledge-Learning-Instruction framework, Koedinger et al., 2012) to answer questions about instruction. In practice, however, domain experts and students are not always accessible for task analyses and existing learning theories are often difficult to translate into the specific contexts faced when designing a tutor (Koedinger, Booth, & Klahr, 2013)—particularly in situations where multiple instructional factors interact. In these situations, designers must rely on their prior experiences and self-reflections to guide design decisions.

However, there are pitfalls to using intuition to guide tutor design. For example, Clark et al. (2008) argue that much of an expert's knowledge is tacit—as people gain expertise their performance improves, but it becomes harder for them to verbally articulate the intermediate skills they use. Thus, domain experts (and instructional designers) often have "expert blind spots" regarding the intermediate skills novices need in order to reach proficiency in a particular domain (Nathan et al., 2001). Additionally, the instruction that designers received when they were learning may not be the best model for good instruction, and their learning experiences may not be representative of others. This insight is aptly captured in the instructional design mantra, "the learner is not like me".[1] However, even if instructional designers remember this mantra, they still face situations where they have no choice but to rely on their own experiences to guide design.

Luckily, design is an iterative process, and poor design decisions can be identified and corrected through testing and redesign. The current best practice for testing a tutor design and improving its pedagogical effectiveness is to conduct a close-the-loop study (Koedinger, Stamper, et al., 2013; Liu et al., n.d.); i.e., to create an initial tutor, deploy it in a classroom, analyze the data from the deployment, use findings from the analysis to redesign the tutor, and deploy it again to test the effectiveness of the redesign. The close-the-loop approach is essentially a data-driven cognitive task analysis technique that guides the redesign of both the material being taught (e.g., the underlying model of skills necessary for domain expertise) and how the material is being presented (e.g., emphasizing certain steps in the interface or providing more practice on certain skills). Koedinger et al. (2013) applied this approach to a geometry tutor and showed that students more quickly achieved mastery in the redesigned tutor. While this approach, and other similar approaches, like A/B testing (Lomas et al., 2013), are effective for testing and improving initial tutor designs, closing-the-loop is expensive. Classroom studies take time to arrange and run, and often multiple classroom study iterations are required to achieve a good tutor design.

Given these current practices, what is needed is a tool that leverages learning science theory to guide the initial design phase, to support designers in the build process, and to test the pedagogical effectiveness of tutor designs without having to run classroom studies. In this thesis, I explore the use of computational models of human learning from examples and feedback, what I call *apprentice learner* models,[2] for these purposes (VanLehn et al., 1994). These models encode problem-solving and learning theory into self-contained computer programs that can simulate human behavior in a tutoring system, as defined by VanLehn (2006). Similar to how engineers simulate and test bridge designs using Computer-Aided Design software, instructional designers can simulate single students or even entire classroom studies (e.g., an A/B test or a close-the-loop study). These models are similar in spirit to Card, Moran, and Newell's (1986) Model Human Processor, which attempted to encapsulate scientific findings from cognitive psychology in a format that could be used to guide interface design. However, they center the attention on learning—they might be viewed as Model Human Learners rather than Processors—and attempt to encapsulate learning science findings in a format that can be used to guide instructional design.

To support the initial design and build phases, apprentice learner models facilitate the expert model authoring process. Similar to human apprentices (Collins et al., 1987), domain experts train models by providing them with examples and feedback. These models translate this

[1]This is Ken Koedinger's variation of Bonnie John's user-centered design mantra "the user is not like me."

[2]I use this term slightly differently than prior work on *learning apprentices* (Dent et al., 1992) or *apprenticeship learning* (Abbeel & Ng, 2004), which typically centers on learning from examples, but not from feedback.

instruction into expert models that can power tutoring systems or, more generally, model expert behavior. Thus, they provide a means for non-programmers to build expert models. Prior work suggests that these models can enable efficient expert-model authoring (Jarvis et al., 2004; MacLellan et al., 2015), which should, in theory, make it possible for non-programmers to author more complex tutoring systems than would be practical with other non-programmer approaches. Also, unlike authoring tools that provide support for designers to construct expert models directly, such as Example-Tracing (Aleven et al., 2009) or the Generalized Intelligent Framework for Tutoring (Sottilare & Holden, 2013), apprentice learner models act as a check against expert blind spots because they start without any of the target skills and they struggle to acquire them if key intermediate steps are missing during training. In support of this idea, Li et al. (2013) showed that skill models discovered by training SIMSTUDENT (a particular model) fit human data as well as, or better than, expert-constructed skill models across three domains.

These models also support the testing phase of tutor development because they can simulate students interacting with initial tutor prototypes. The data generated from these simulations has an identical format to the data generated from real classroom studies (i.e., tutor transaction log files), so instructional designers can analyze it using existing tools and techniques. For example, designers might apply learning curve analysis techniques, such as Additive Factors Modeling (Cen, 2009), to gain insights into which skills will be more difficult for students to learn, so they can author additional practice problems that exercise challenging skills. Similarly, a designer might analyze simulated data to test the overall effectiveness of a particular tutor design, or to compare alternative designs to determine which are most effective for learning. Thus, developers can analyze simulation data to guide subsequent tutor redesigns, analogous to Koedinger et al.'s (2013) close-the-loop approach. However, unlike classroom studies, simulation studies can be performed at a much lower cost and can be easily applied to many alternative tutor designs. Thus, apprentice learner models can act as a tool to leverage prior learning theory to cost effectively test and improve initial tutor designs prior to actual classroom deployments.

While prior work has explored how these models, particularly the SIMSTUDENT model (Jarvis et al., 2004; Li et al., 2014; Matsuda et al., 2014), support the designing and building phases of tutor development, their capacity for testing tutors has never been explored. In particular, it is unclear whether existing models will produce behavior that has good agreement with human behavior. Prior studies with SIMSTUDENT have centered on demonstrating learning efficiency (Jarvis et al., 2004; Li, Cohen, & Koedinger, 2012; Li et al., 2014; Li, Schreiber, et al., 2012; Matsuda et al., 2014), which is sensible for the purposes of authoring. However, for the purposes of supporting tutor design and testing, it is important to model both correct and *incorrect* human behavior. If apprentice learner models are too efficient, then they might overcome issues that would cause problems for human students, making them less useful as tools for identifying expert blind spots, detecting likely human bugs, or differentiating which tutor designs would be better for learning.

Fortunately, there exist many human tutoring system data sets in public repositories, such as DataShop (Koedinger et al., 2010), that could enable the evaluation of current models and guide the search for more human-like models. In particular, researchers could evaluate a model by comparing its simulated behavior within a tutoring system to existing human behavioral data from a classroom study with the same tutor. Repeating this process with multiple alternative models would enable researchers to determine which yield behavior that better aligns with the human behavior. In isolation, a classroom study will ultimately yield only a single refined tutoring system. However, if researchers use data from these classroom studies to discover better apprentice learner models, then these improved models can support future tutor design.

The ultimate goal of this research program is to produce a high-fidelity computational model of human learning (i.e., a Model Human Learner) that can predict human behavior in tutoring systems across a wide range of domains and support tutor development. Towards this end, this thesis does the following:

- Defines the apprentice learning problem and review prior attempts to solve it (chapter 2);
- Presents the Apprentice Learner Architecture, a theoretical framework that outlines the key apprentice learning knowledge structures and the learning and performance components that operate over them (chapter 3);
- Develops the DECISION TREE and TRESTLE models of learning and discusses their relation to prior models (chapter 3);
- Showcases the efficiency of the DECISION TREE model for authoring an experimental design tutors (chapter 4);
- Demonstrates the use of both models for predicting the effect of problem ordering on students' tutor and posttest performance, making learning curve predictions, and for testing alternative learning theories using tutor data (chapter 5); and
- Highlights the generality of these models by evaluating their simulation and authoring capabilities in seven tutors spanning a wide range of content domains (chapter 6).

Although this thesis does not present a Model Human Learner that is consistent with all available educational data, it makes four key contributions. First, it presents a theory of apprentice learning that defines a space of potential apprentice learner models and provide an evaluation approach that could, in future work, lead the to discovery of such a model within this space. Second, it demonstrates that preliminary models are efficient for authoring tutors that would be difficult to build with existing approaches (e.g., experimental design or stoichiometry). Third, it demonstrate that these models are capable of predicting main instructional effects (e.g., the effect of different problem orderings on learning) and accounting for students' behavior within a tutor. Finally, it show that these preliminary models are sufficiently general to learn a wide range of knowledge types (associations, categories, and skills) across a wide range of domains (Chinese character learning, RumbleBlocks stability, English article selection, stoichiometry, fraction arithmetic, boxes and arrows, and equation solving) with a small, fixed set of prior knowledge.

Chapter 2

Apprentice Learning

Humans learn new skills in a variety of ways. One format in which they regularly engage is *apprentice learning*, or learning from the examples and feedback provided by expert instructors. From the perspective of the learner, there are practical benefits to this approach over more formal schooling methods, such as lectures. In particular, prior studies of human apprenticeship find that it situates learned skills within their context of use, whereas lecture-based formats typically abstract context away—making it harder for students to later retrieve and apply learned skills in practice (Collins et al., 1987). Additionally, this paradigm is often easier for instructors. Notably, skill knowledge is typically tacit and thus difficult to verbally communicate (Clark et al., 2008). In contrast, apprenticeship facilitates a more natural transference of skills because demonstrating and coaching skills is often much easier for teachers than formalizing and describing them.

These benefits make apprentice learning a natural approach for skill transfer. Authoring tools equipped with this capability should let developers efficiently author tutors with examples and feedback, rather than having to formalize and program knowledge directly into a tutor, which is time consuming and requires programming expertise (Matsuda et al., 2014). Additionally, tutoring systems utilize this format of instruction; i.e., they provide students with immediate correctness feedback on their problem-solving steps and examples[1] when they are stuck (VanLehn, 2006). Thus, developers should be able to use models of human learning to stimulate student interactions with their tutors, in order to test the pedagogical effectiveness of these systems before deployment.

2.1 The Problem

In order to build a model that supports tutor authoring and that simulates students, the first step is to formally define the learning problem. The starting point for my formulation is VanLehn's (2006) characterization that apprentice learning within tutors consists primarily of two loops: an outer loop, in which an instructor selects problems for a student to attempt, and an inner loop, in which students attempt steps and instructors provide hints and feedback. Figure 2.1 further divides these behaviors into five distinct interactions between the tutor and tutee. The first

[1]Examples are often referred to as *bottom-out hints* within the tutoring literature because they are usually provided to students as the final hint, in a sequence of hints, for a given problem-solving step.

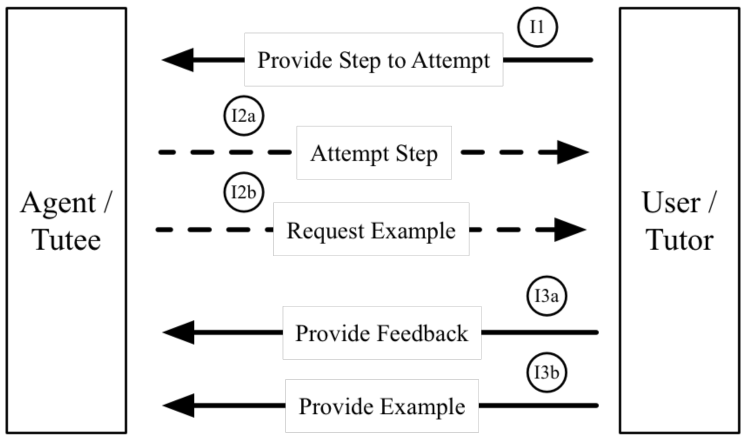
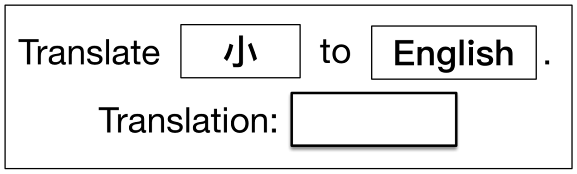


Figure 2.1: The five apprentice-learning interactions. First the tutor provides the learner with a step to attempt (I1). In response the learner either attempts to solve the step (I2a) or requests an demonstration (I2b). If the learner makes an attempt, then the tutor provides feedback on its correctness (I3a). In the case of a an example request, the tutor provides a demonstration of the step (I3b). Note, the solid lines denote actions performed by the tutor and the dashed lines denote actions performed by the tutee.

interaction (I1) corresponds to the outer loop, in which a tutor selects steps for a tutee to perform. Note that VanLehn (2006) originally characterized the outer loop as selecting problems, not steps, but the current characterization is slightly more general. In particular, a tutor that selects steps can achieve identical behavior to a tutor that select problems—by always selecting the successive steps within a problem. However, it also allows for tutors that can assign students to work on specific steps within problems, potentially skipping some steps. The other four interactions (I2a, I2b, I3a, and I3b) correspond to the inner loop. In particular, in response to receiving a step, the student can either attempt it (I2a) or request an example of how to perform it correctly (I2b). The tutor then either provides feedback on the student's attempt (I3a) or an example (I3b), depending on the student's action. At the end of this sequence (I1 $\rightarrow$ I2 $\rightarrow$ I3), the tutor typically begins interaction again by providing the student with a new step (e.g., if they got the last step correct) or the same step again (e.g., if their attempt was incorrect and the tutor wants to give them another chance).

A key characteristic of these interactions is that they frame apprentice learning as an *incremental* and *interactive* process. In other words, learners are given training examples and feedback sequentially over the course of their problem solving, rather than in a single batch up front. Further, the steps, feedback, and examples they receive are chosen by a tutor in response to their actions. For example, instructors only provide feedback on the steps a student actually attempts, and, ideally, they select steps that challenge and advance a student's knowledge, rather than those that they already know how to solve. Together, these characteristics enable tutors to *personalize* each student's instruction—continuously assessing their skill knowledge and focusing attention on skills that have the most room for improvement. This personalization is hypothesized to be one of the key reasons that tutoring systems are so effective for human learning (VanLehn, 2006).

Visual Representation of Step



Relational Description of Step

```
(text-label l1)  (contains p1 l1)  (value l1 “Translate”)
(text-label l2)  (contains p1 l2)  (value l2 “to”)
(text-label l3)  (contains p1 l3)  (value l3 “.”)
(text-label l4)  (contains p1 l4)  (value l4 “Translation:”)
(text-field f1)  (contains p1 f1)  (value f1 “小”)
(text-field f2)  (contains p1 f2)  (value f2 “English”)
(text-field f3)  (contains p1 f3)  (value f3 “”)
(problem p1)
```

Figure 2.2: A visual representation of a step in the Chinese tutor and its accompanying relational description.

In order to build a model that supports these interactions, the next step is to define the structure of the information exchanged. During the first interaction, the learner is provided with a description of a step that the tutor has selected. Although many representations for this description are possible, the current work takes the stance that this description is provided in a *relational representation*, such as first-order logic. Figure 2.2 shows an example of a step in a Chinese character tutor and a relational description.

An important assumption of the current work is that the learner is not *explicitly* provided with problem-solving goals (i.e., a partial description of a desirable state).[2] However, there is always an implicit goal during apprentice learning that the student should attempt to take actions that receive positive feedback and avoid actions that receive negative feedback. Additionally, information about what a learner should do (i.e., goal information) is typically embedded in the state description. For example, in Figure 2.2 the student is tasked with translating the provided Chinese character into English, and the accompanying state description contains this information. I make this assumption because novice students typically are not aware of explicit goals a priori, so they must get this information from the provided problem statement. Thus, in addition to learning new skills, students must learn the mappings between the information available in the state description and their learned skills.

Once a learner receives a description of a step to work on, they respond with either an attempt or a request for an example. When making an attempt, the learner specifies the action that should be taken within the tutor, which for the current work I represent using a Selection-Action-Input (SAI) triple (Ritter & Koedinger, 1996). For example, given the step in Figure 2.2, the student might respond with (sai f3 UpdateTextField “small”), which specifies that interface element f3 should be updated with the text “small” (the correct English translation for “小”). Typically, when the tutor receives an SAI, it checks its correctness[3] and responds with a Step-SAI-Feedback triple, which contains the relational description of the current step, the SAI from the student, and the correctness feedback on this SAI. In situations where the learner requests an example, the tutor also responds with an Step-SAI-Feedback triple, but the SAI is one that is generated by the tutor, and the feedback is marked as correct (assuming the tutor is providing a correct

[2]More generally, goals are not modeled as explicit cognitive structure in this work, but future work should investigate how explicit goals can be incorporated into the current theoretical framework.

[3]The current work assumes that correctness is Boolean, but in theory it could be a continuous value.

example). By specifying that the response from the tutor for both the feedback and examples has the same format (a Step-SAI-Feedback triple), the learner can process them in similar ways. In particular, receiving feedback on a correct step is analogous to receiving the same correct step as an example. Also, it is worth noting that even though the current work is theoretically committed to apprentice learning as an incremental and interactive process, the current formulation supports non-incremental and non-interactive instruction as well. In particular, an instructor could provide a student with multiple examples, either positive or negative, at any point within, or outside, of the typical interactive loops.

One final point that is worth discussing is that steps *are* the granularity at which tutors provide feedback, but step size is not necessarily consistent across tutors. At one extreme, tutors might present problems as a single step and only provide feedback on the final answers. For example, a tutor might present a student with the problem $3x+2 = 11$ and only provide correctness feedback when they give the correct final answer ($x = 3$). In this case, a student might need to take multiple intermediate, non-tutored, actions in order to produce the final answer. In this scenario, when a student gets feedback, they are faced with the *credit-assignment* problem (Sleeman et al., 1982); more specifically, they need to determine which of their intermediate actions are responsible for their answer's correctness. At the other extreme, a tutor might present problems with steps for each intermediate problem-solving action and provide feedback on each. In this scenario, the student is effectively faced with a supervised learning problem and does not have to handle credit assignment.

There is an added complication that what constitutes a single action might be different between learners. For example, updating the answer field with the word "small" in the step shown in Figure 2.2 might be a single action for most English students. However, for students less experienced with English keyboards, this might require multiple actions for inputting the individual letters. In general, VanLehn (2006) hypothesizes that decreasing the grain size of steps is better for students learning because it reduces the amount of credit assignment that a student must perform. However, a super-fine grain size is inefficient because it requires students to process more feedback for domain-independent skills, such as typing, and draws attention and feedback away from domain skills, such as Chinese translation. In practice, most tutors assume basic prior knowledge for skills like typing and choose a step size that usually consists of one to two actions, depending on a student's prior knowledge.

2.2 Prior Work

Given this formulation of the problem, I next turn to the literature and review prior models that conform to it. While there are a number of modeling efforts that fit at least some aspect of the current formulation, reviewing all of these works is beyond the scope of this thesis. Instead, I focus on orthogonal methods within the literature and discuss prototypical examples that leverage these approaches in order to situate the current work. It is important to note that many of these examples do not attempt to model *human* learning, but they all utilize apprentice learning in some form.

The predominant view within the literature is that problem-solving consists of heuristic search through a problem space (Newell & Simon, 1972), which is typically defined using a

STRIPS-like representation (Fikes et al., 1972). Within this representation, a problem space is defined by:

- a state representation (i.e., relations and features) that describe possible states;
- operators, which present as IF-THEN rules for moving from one state to another; and
- goals, which can appear explicitly as partial descriptions of desirable states or implicitly as black-box tests for assessing whether a particular state achieves the goals.

Given this view, prior work characterizes apprentice learning as the translation of problem-solving traces (i.e., particular paths through a problem space) and feedback on them into knowledge of the structure of the underlying problem space and how best to search it.

Although most work can be unified under this characterization, the prior work differs on the source and format of the problem-solving traces used for learning. One common assumption is that complete traces—consisting of all state-operator pairs—are provided by experts in one batch upfront. Despite the restrictiveness of this requirement, many systems adopt it. For example, Abbeel and Ng (2004) explore an approach for learning to solve a variety of common toy AI problems given anywhere from 100-100,000 complete expert traces for each problem up front. However, not all systems subscribe to these restrictions. In particular, some systems, such as LEAP (Mitchell et al., 1985) and ODYSSEUS (Wilkins et al., 1986), relax the constraint that all of the traces must be provided in one batch. Instead, these systems learn incrementally by observing human experts over the course of their normal problem solving. Other systems further relax the requirement that problem-solving traces must be complete. For example, ICARUS (Li et al., 2009) is capable of learning from traces that lack some of the state-operator pairs, some of the operators (i.e., that contain only the intermediate states), or both, by filling in the gaps using means-ends problem solving. Taken to the extreme, this problem-solving approach, and others like it (Anderson, 1993; Fikes et al., 1972; Laird et al., 1987), enable systems to generate their own problem-solving traces.

Most systems that get their traces from experts assume that the latter also provide the feedback on them. In particular, expert traces are usually implicitly assumed to be correct, although it is possible for experts to also explicitly label traces as positive or negative. When systems generate their own traces, however, there is some variation in the source of feedback. Systems that possess explicit descriptions of problem-solving goals often provide their own feedback by labeling traces that achieve the goals as correct and those that fail (e.g., dead ends or loops) as incorrect (Langley, 1985). When systems lack explicit knowledge of goals, they must instead rely on implicit goal tests (i.e., experts or other black-box sources) to label generated traces. One challenge faced by these systems is that navigating a problem space without any knowledge of the goals can be difficult, if not impossible. To overcome this challenge, many systems (Martínez et al., 2015; Nason & Laird, 2005; Tadepalli et al., 2004) leverage feedback on earlier problem solving to direct search to more promising parts of the space.

Despite variety in the origin and presentation of traces and feedback, once they are in hand the prior literature has coalesced around a number of methods for translating them into new problem-solving knowledge. In particular, methods exist for learning operator preference knowledge, learning heuristic operator conditions, learning composite operators, and even learning new primitive operators. I will review each in turn.

One of the most common types of knowledge extracted from solution traces is operator preference knowledge, which describes the operators that should be preferred during search. In particular, at each state in the problem space, an agent must decide which of its operators to execute. If the agent is informed and makes good choices, then it will quickly achieve its goals. However, when an agent is uninformed, then it will spend time exploring unfruitful portions of the problem space and often fail to solve difficult problems in a reasonable amount of time. To combat this challenge, prior work explores how to leverage correct and incorrect traces to learn which operators are preferable in each state. In particular, most systems decompose traces into their constituent state-operator-feedback triples, which become training data for learning (Sleeman et al., 1982). Systems using this approach typically label state-operators along successful paths as correct and those that lead to failures, or sometimes are just off path, as incorrect.

Given these training data, one common type of learned knowledge is operator preference relations, which specify preferences over potential operators, such as which to accept or reject, which are good or bad, or which should be tried first (e.g., try Operator-1 before Operator-2). ELM (Brazdil, 1978) is an early example of a system that learns ordering relations by analyzing which operators occur before others in experts' algebra and arithmetic problem-solving traces. One limitation of this early approach was that learned relations were oblivious to state information, often resulting in contradictions (i.e., Operator-1 $>$ Operator-2 in some states, but Operator-1 $<$ Operator-2 in others). Other systems, such as Soar (Laird et al., 1987), correct this limitation by learning preferences (modeled as selection and rejection rules) that condition on the current situation (the current problem space, goal, operator, and state in Soar). Another common type of learned knowledge is operator preference functions, which specify numeric values (typically reward or cost) over states, operators, or state-operator pairs that determine the order to expand operators during search. This approach is commonly used in reinforcement-learning systems, such as the work of Abbeel and Ng (2004), which fits a reward function to state-operator-feedback triples extracted from expert traces, or Soar-RL (Nason & Laird, 2005), which utilizes *Q-learning* to learn a reward function from its own problem solving.

Another kind of knowledge commonly extracted from solution traces is heuristic conditions, which constrain, beyond their original, legality conditions, the number of situations in which operators apply—effectively reducing the branching factor of the search. For example, when playing chess, legality conditions on an agent's move operator express when they *can* move a piece. In contrast, the heuristic conditions express when they *should* move a piece. The acquisition of this type of knowledge is similar to that of preference knowledge in that solution traces are decomposed into constituent state-operator-feedback triples, which are then used for supervised learning. For example, given these triples, LEX (Mitchell & Banerji, 1983) applies version-space learning and SAGE (Langley, 1985) applies discrimination learning to discover heuristics for a variety of domains, ranging from Towers of Hanoi to symbolic integration.

Like preference knowledge, heuristic conditions direct search to more promising parts of the space. However, unlike preferences, they completely exclude operators from consideration at the onset, which tends to improve search efficiency because evaluating heuristic conditions is often much faster than computing preferences over all legally-applicable operators (and each of their bindings). A key limitation of heuristic conditions, though, is that they can sometimes be over constraining. In particular, incorrect heuristic conditions can make it impossible for systems to generate their own correct problem-solving traces. Some systems, such as SAGE, try to avoid this problem by biasing learning towards more general, rather than more specific, heuristic conditions, so that operators are only constrained when there is sufficient evidence for doing so. Additionally SAGE and other systems, such as ACT (Anderson, 1982), retain more general rules for use when none of the more specific ones apply. This problem is also somewhat alleviated in circumstances where correct expert traces are available because the training data they contain can provide the basis for correcting overly-specific conditions.

In addition to learning preference knowledge and heuristic conditions, which change how an agent searches a problem space, many researchers have investigated how agents might acquire new operators, which change the underlying structure of the space they search. Within this area, macro-operator learning, or learning operators composed from other operators, is one of the more common approaches. Systems that engage in this kind of learning compile sequences of operators that appear in successful problem-solving traces into new operators that can be used in subsequent search. Many systems, such as STRIPS (Fikes et al., 1972), ODYSSEUS (Wilkins et al., 1986), and GENESIS (Mooney, 1987) utilize explanation-based learning to construct macro-operators from successful problem solving. These approaches analyze a given sequence of operators and determine the minimal conditions that must be met for successful execution of the entire sequence. Learned operators are more efficient to execute than their constituents because condition matching only needs to happen once up front, rather than once for each constituent. Other approaches, such as chunking in Soar (Laird et al., 1987), knowledge compilation in ACT (Anderson, 1982), and hierarchical task network learning in ICARUS (Li et al., 2009), use different approaches and representations, but ultimately serve a similar purpose of learning compositional operator structures. In general, learned macro-operators enable agents to take bigger steps across the problem space, essentially decreasing the search depth needed to find a solution. However, sometimes the cost of using new operators outweighs their benefit resulting in worse performance (e.g., new operators increase the branching factor of search and make pattern matching more expensive), something known as the *utility problem* (Minton, 1990).

One limitation of macro-operators learning is that it requires sufficient knowledge of primitive operators. Most systems assume all primitives are provided, but some researchers have explored techniques for directly acquiring new primitives from expert[4] problem-solving traces. In general, the work in this areas centers on analyzing the differences between subsequent states in a trace in order to induce new operators. For example, ALEX (Neves, 1985) learns new primitive operators by searching for built-in symbol manipulating functions (e.g., insert-after or string-concatenate) that explain the differences between states. Once it finds functions to explain the differences—which constitute the THEN part of learned rules—it applies condition learning techniques to discover the IF part. It is important to note that this technique is analogous to macro-operator learning where the agent has primitive operators to represent built-in symbol manipulating functions; however, the approach is slightly different in that it maintains a clear distinction between domain operators and functions, similar to VanLehn's (1999) *overly-general* operators, and only falls back on the later when necessary. An alternative approach is to directly learn new functional mappings between states. For example, KnoMic (van Lent & Laird, 2001) learns new primitive air-combat operators by applying specific-to-general learning to the differences between states in expert traces (along with specific-to-general learning to acquire condi-

[4]Generally, it is difficult for an agent to learn primitives from its own problem-solving traces because it does not have the knowledge necessary to generate them in the first place, although it may be possible.

tions on these operators). One limitation of this technique is that it requires complete expert traces that include information about which operators are being applied in each state, so training data can be appropriately grouped for learning. CASCADE (VanLehn, 1999) overcomes this limitation by only learning primitive operators with effects that correspond exactly to observed differences between states (i.e., effects are not generalized). Expert operator annotations are not required in this case because all pairs of subsequent states that exhibit the same differences are assumed to be applications of the same primitive operator. VanLehn (1999) provides evidence that the combination of simple primitive operator learning with macro-operator learning explains much, although not all,[5] of the learning observed in human physics students.

Collectively, this prior research, which often centers on the specific learning subproblems, begins to form an overall picture of how an agent might engage in apprentice learning. In particular, an expert provides an a learner with problem-solving traces and feedback, the learner generates their own traces and feedback, or some combination of the two. Once a learner has these training data, they can learn four kinds of knowledge:

1. primitive operators, which grow the problem space (increasing branching factor);
2. macro-operators, which enable an agent to take bigger steps through the problem space (decreasing search depth, but increasing branching factor);
3. heuristic conditions, which specify when an agent should, or should not, consider operators during search (decreasing the branching factor); and
4. operator preferences, which guide search to more promising portions of the problem space (typically decreasing overall search).

As these kinds of knowledge have complementary benefits for problem solving, many systems exist that learn some combination of these knowledge types. One relevant subset of this work centers on modeling students' buggy problem-solving knowledge (VanLehn, 1983). Although these systems do not model human learning, they uses apprentice learning to discover knowledge that explains human behavior. For example, ACM (Langley & Ohlsson, 1984) discovers heuristics that explain students' erroneous multi-column subtraction behavior. This system is provided with a collection of overly-general operators—operators that could hypothetically explain a student's behavior, but that only have minimal legality conditions, such as only type conditions on inputs—and traces of a student's incorrect problem solving. It then searches for additional heuristic conditions on the overly-general rules that constrain their application to the situations observed in the provided traces. These more constrained rules constitute a model of the buggy knowledge that generated the provided traces. Langley and Ohlsson (1984) show that this system is capable of discovering heuristics that account for many of the common subtraction bugs exhibited by human students.

Another line of work centers on human learning. Prototypical examples in this line include CASCADE (VanLehn et al., 1991) and STEPS (Ur & VanLehn, 1995), which models students learning physics from self explanation and from a physics tutoring system. Both systems make a distinction between domain rules (for problem solving) and overly-general rules (for explaining provided traces, which are correct in this case) and use explanation-based learning to compile the latter into the former (i.e., they learn domain-specific macro-operators). However, there

[5]Humans regularly engage in reflective behaviors to test, debug, and optimize their learned rules, something CASCADE did not model.

are some differences between the two systems. In particular, CASCADE can learn primitive operators with constant effects by studying examples, but STEPS cannot. They also differ in how they direct their problem solving; CASCADE uses preference knowledge it learns from examples, whereas STEPS learns heuristic conditions from its problem-solving traces.

Another major system in this line of work is SIMSTUDENT (Li et al., 2013; MacLellan et al., 2015; Matsuda et al., 2009, 2011), which models human learning across multiple tutor domains. Like the other systems, it makes a distinction between functions (i.e., overly-general rules) and skills (i.e., domain rules), and it discovers new skills by explaining an expert's steps using functions and compiling these explanations into new skills (i.e., it learns domain-specific macro-operators, as in CASCADE and STEPS). SIMSTUDENT also learns conditions on these skills, but, unlike the other systems, it makes a distinction between learning retrieval patterns (relational conditions that specify how relevant elements relate to other elements in the state) and feature conditions (non-relational conditions over elements bound in these retrieval patterns). This distinction is both practical and theoretical in that separating relational and non-relational conditions lets it leverage specialized learning approaches for each and to employ separate learning biases for each (retrieval pattern learning is specific-to-general and feature condition learning is general-to-specific).

These systems each represent major progress towards modeling humans in apprentice learning contexts. However, they each perform only a subset of the four types of learning and they implement different approaches for each. What is needed is a theory that unifies the different types of learning and provides a means of testing which implementations best align with human behavior. Additionally, the generality of these prior systems remains unclear. In particular, how much prior knowledge needs to be authored to employ these systems in new domains? For example, CASCADE and STEPS focus mainly on physics, but do these models also explain behavior in language or engineering domains? If so, how much additional prior knowledge must a researcher (or teacher or domain expert) develop in order to apply them?

SIMSTUDENT has been applied to multiple tutoring domains and, consequently, might have a better claim to generality. However, each SIMSTUDENT model possesses a specialized set of domain-specific prior knowledge, suggesting that a would-be user needs to author additional content for their domain. Li (2013) investigates how to automatically discover prior knowledge using unsupervised learning, but these approaches require access to training data in one batch upfront, which may not be available for novel domains. Given this limitation, it seems reasonable for researchers to apply these techniques to learn domain-general prior knowledge, such as how to parse English sentences or mathematical formulas, but it seems unlikely that teachers, or other non-technical domain experts, could apply them to authoring domain-specific knowledge. It seems more feasible for non-technical users to hand author the prior knowledge than to collect/clean-up domain-specific training data and apply unsupervised learning to them; although, this is an open empirical question. In contrast, while CASCADE and STEPS focus mainly on physics, they suggest that the combination of access to overly-general prior knowledge, heuristic condition learning, preference knowledge learning, macro-operator learning, and primitive operator learning (learning primitives with constant effects) might be sufficient to model human learning across a wide range of domains. I investigate these ideas in the current work.

Chapter 3

The Apprentice Learner Architecture

In this chapter, I present a modular architecture that builds on prior systems, such as ACM, CASCADE, STEPS, and SIMSTUDENT, and unifies their mechanisms and theories. This architecture was designed to support the five apprentice-learning interactions presented in the previous chapter (see Figure 2.1). In particular, agents within the architecture can receive steps from a tutor (I1), attempt these steps (I2a) or request examples of how to perform these steps (I2b), and receive feedback (I3a) or examples (I3b). Given these interactions, the architecture (see Figure 3.1) embodies a theory about the knowledge structures needed to support them and about the performance and learning components that operate over these structures.[1] Beyond these theoretical commitments, the framework remains intentionally abstract on implementation specifics for each component, as different implementations may be preferable in different circumstances. For example, implementations of the learning components that are good for tutor authoring (e.g., that learn efficiently and do not forget) may not be good for simulating students (e.g., if they fail to produce human-like mistakes) even though apprentice learning occurs in both circumstances. In this sense, the architecture is similar to other efforts to identify the mechanisms humans use for learning (Ohlsson, 2011), but it aims to characterize apprentice learning more generally (both human and computational).

Under this characterization, models are defined as implementation choices for each architectural component as well as initial knowledge configurations. While explaining each component, I discuss the implementations currently supported by the architecture and, at the end of this chapter, two models that are the focus of the remainder of this thesis. One of the risks of instantiating theory in computational systems, such as the current architecture and models set within it, is the introduction of assumptions beyond the target theory. These auxiliary assumptions are often innocuous, but sometimes they unintentionally provide explanatory power to systems that incorporate them. To counter this risk, I have attempted to restrict such assumptions primarily to models and to concentrate the theoretical assumptions in the architecture. However, a clean division is not possible, so I try to present all major assumptions underlying both the architecture and the models built within it.

[1]The architecture is open source and available at `https://github.com/cmaclell/apprentice_learner_api`.

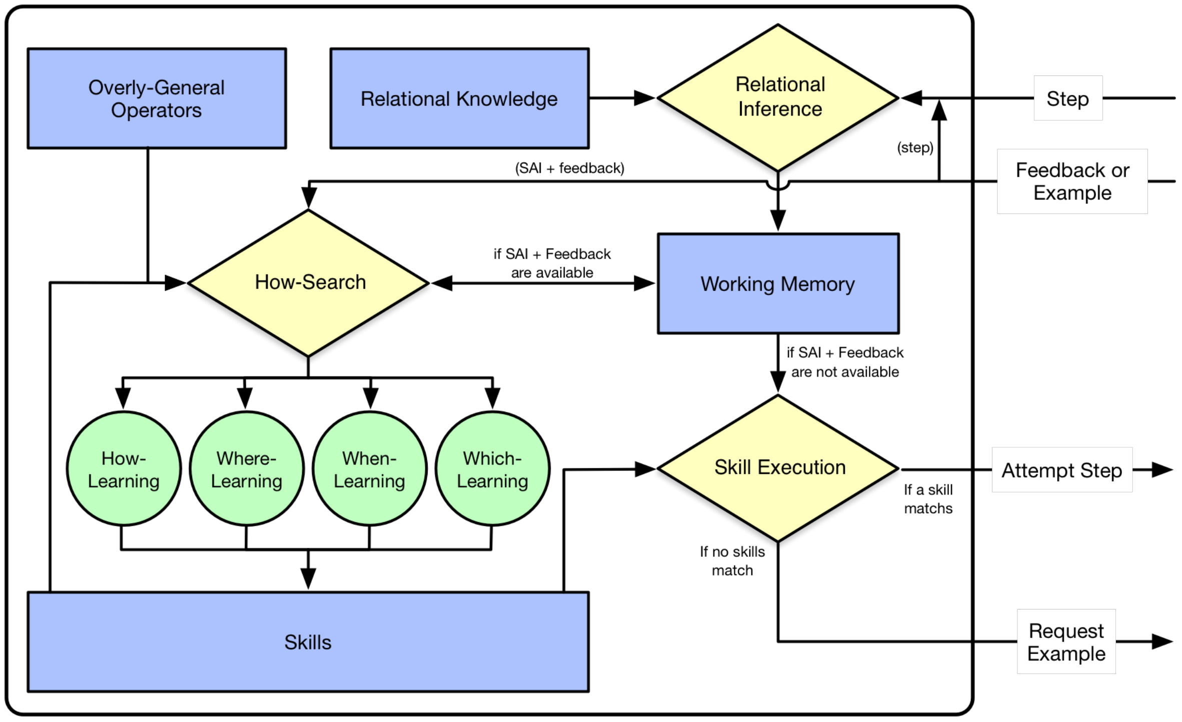


Figure 3.1: The Apprentice Learner Architecture and its interactions with a tutor. Blue boxes represent knowledge structures, yellow diamonds represent performance components, and green circles represent learning components.

3.1 Knowledge Structures

Like many prior cognitive systems, the architecture possesses both short- and long-term memories to support problem solving and learning. In particular, it has a single, short-term, *working memory* that describes the current step as provided by the tutor. This memory contains elements representing particular objects that occur in a given step, such as fields in a tutor interface, and other information that describes them. This information is stored using a relational representation, similar to STRIPS (Fikes et al., 1972) or PROLOG. Figure 2.2 shows a simple example of a step's encoding in working memory.[2] Elements in this memory only exist for the duration of a single step before they are cleared and the information for the next step is provided and stored.

In addition to short-term memory, the architecture has three long-term memories. The first contains *relational knowledge*, which matches against elements in working memory and augments these elements with additional relations. Table 3.1 shows two examples of relational knowledge. The first evaluates whether two elements have equal values. If they do, then it augments working memory to represent this relationship. The second matches all elements with non-empty, string values, and augments working memory with unigram relations that describe these elements.

In general, these structures have three components: a head, conditions, and effects, which

[2]Typically elements also have type information and slightly more complex hierarchical organization.

Table 3.1: Two examples of relational knowledge. The first evaluates equality of two elements and the second computes unigram relations. Functions are shown in italics.

Head	(value-equality ?x ?y)	(unigrams ?x)
Conditions	(value ?x ?xv), (value ?y ?yv)	(value ?x ?xv), (*is-string* ?xv), (*neq* ?xv “”)
Effects	(value-equality ?x ?y (*eq* ?xv ?yv))	((unigram ?x ?w) for ?w in (*extract-words* ?xv))

can contains pattern matching variables (preceded by a “?”) that bind with elements in working memory. The head contains a name for the knowledge element, as well as arguments that constrain how it is applied. In particular, each knowledge structure is evaluated once per a given binding of its head arguments, ignoring alternative condition matches that share identical head bindings.[3]

The conditions determine how pattern matching variables are bound and describe when a particular piece of knowledge applies. There are two types of conditions: relations and functions. Both contain pattern matching variables, but relations match against working memory elements to determine possible bindings for variables, whereas functions are only evaluated after all variables they refer to are bound. Functions appearing in conditions can return any value, but if they return the Boolean value *False*, then the current match fails.

For each successful match, the effects determine how working memory is augmented in response. Like conditions, effects can contain relations and functions. However, only variables bound in the conditions or constants can occur in either. Also, if a function fails to execute (i.e., raises an exception), then the match fails. In this regard, they represent a kind of additional implicit condition; however, they are typically less efficient to evaluate than explicit conditions because pattern matching can logically exclude whole classes of possible matches whereas functions can only be applied to invalidate matches one match at a time. Relational knowledge is similar to conceptual knowledge in ICARUS (Langley et al., 2009). However, unlike conceptual knowledge, the head and effects are kept distinct because effects can produce multiple elements in working memory, as shown Table 3.1. This type of knowledge is also analogous to primitive feature knowledge in SIMSTUDENT, but is more expressive. In particular, SIMSTUDENT capabilities that required separate architectural components, such as those for augmenting working memory with grammar features (Li et al., 2014), can simply be encoded as additional pieces of relational knowledge (using an approach similar to the unigram knowledge shown in Table 3.1).[4]

The second kind of long-term knowledge structures, *overly-general operators*, are domain-independent primitives for explaining examples. These structures are similar in composition to relational knowledge—they are described by a head, conditions, and effects, and have an identical syntax—but they only apply during explanation, when domain skills fail. Table 3.2

[3]This feature is primarily for efficiency and prevents the systems from entertaining all possible ways to match conditions if it is not necessary to do so. However, if all pattern matching variables are included in the head, then all possible matches will be considered.

[4]Although relational knowledge does support the inference of grammar features from an existing grammar, it does not currently support grammar learning. However, prior approaches (e.g., Li et al., 2014) could be applied to generate these grammars.

Table 3.2: Two examples of overly-general operators. The first adds two values and the second concatenates them. Italics are used to represent functions.

Head	(add ?x ?y)	(concatenate ?x ?y)
Conditions	(value ?x ?xv), (value ?y ?yv)	(value ?x ?xv), (value ?y ?yv)
Effects	(value (add ?x ?y) (*str-float-add* ?xv ?yv))	(value (concatenate ?x ?y) (*str-append* ?xv ?yv))

Table 3.3: An example fraction arithmetic skill for adding two numerators.

Skill Label	add-numerator
Utility	0.8
Legality Conditions	(value ?n1 ?n1val), (value ?n2 ?n2val), (value ?an "")
Heuristic Conditions	(name ?n1 Numerator1), (name ?n2 Numerator2), (name ?an AnswerNumerator)
Classifier:	<*learned-classifier*>
Effects:	(SAI ?an "UpdateField" (*str-float-add* ?n1val ?n2val)))

shows two examples of these operators, one that adds two values and one that concatenates them. The first relies on the *str-float-add* function, which takes a pair of numbers represented as either two strings or two floats, adds them, then returns the sum in a format that mirrors the inputs (either a string or float). The second uses *str-append*, which coerces the values into strings, concatenates them, and returns the resulting string. Note, if either function raises an exception, then the operator will fail to execute. These types of operators are analogous to the overly-general operators that occur in STEPS (Ur & VanLehn, 1995) and CASCADE (VanLehn et al., 1991) in that they are domain-independent operators with minimal conditions. Thus, they outline possible computations, but do not specify when these computations should be performed. They are also analogous to the *primitive functions* that occur in other systems, such as ELM (Brazdil, 1978) and SIMSTUDENT (Li et al., 2014), in that they often perform a single function. However, unlike these systems, they support additional conditions that exclude nonsensical bindings of function arguments, such as attempting to add elements with the wrong types. This feature greatly improves runtime performance.

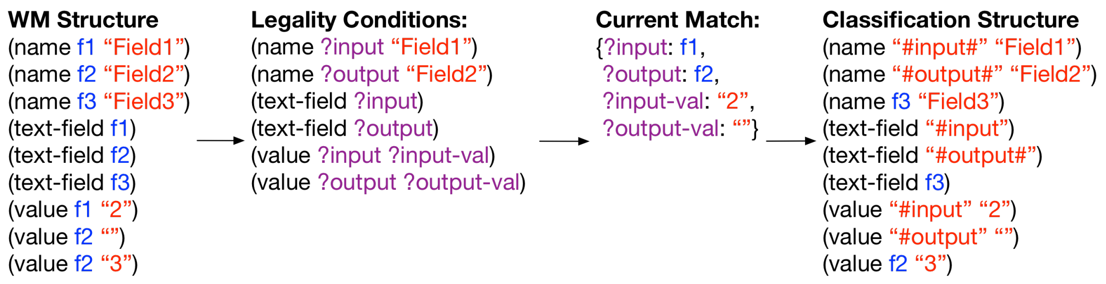
The final long-term memory contains domain-specific *skills*, which are acquired from examples and feedback. Table 3.3 shows an example of a fraction arithmetic skill for adding numerators. These structures are acquired via apprentice learning and contain six parts. First, they have a label, which is an optional, non-unique, human-readable name that is useful for interpreting learned skills and debugging problem-solving traces. Second, they maintain a single numeric value representing their utility. Like relational knowledge and overly-general operators, skills have conditions. However, unlike the other types, they distinguish between two types of conditions: legality conditions, which describe when a skill *can* execute (successful execution also depends on all functions in effects evaluating successfully), and heuristic conditions, which describe when the skill *should* execute. These later conditions do not need to be met for successful execution, but constrain when a skill applies. In addition to conditions, skills also have a classifier (e.g., a learned decision tree) that specifies whether the skill should activate given the current working memory structure and condition bindings. Although classifiers serve a similar purpose to heuristic conditions, they are less efficient to apply because each potential match is evaluated individually (unlike heuristic conditions, which leverage their logical representation to efficiently exclude entire classes of potential matches). In return, however, they have access to a fully bound state representation when deciding whether a skill applies. Thus, classifiers are analogous to operator preference knowledge for accepting or rejecting operators. Finally, skills have a single relational effect that triggers an attempt of the current step when deposited in working memory. This Selection-Action-Input (SAI) effect specifies an interface element (selection), an action to apply to it (action), and an input to this action (input). One limitation of skills is that they do not support a hierarchical organization, like those found in other architectures (Carbonell et al., 1991; Laird et al., 1987; Li et al., 2009). However, skills do refer to overly-general operators and future work should explore how these operators might be organized hierarchically.

3.2 Performance Components

The architecture includes three performance components that operate over these knowledge structures. The first handles *relational inference*, in that it elaborates any steps provided to the system using available relational knowledge and deposits the result in working memory. Although alternative approaches are possible, all the models in this thesis utilizes a forward-chaining approach that applies relational knowledge until quiescence or until a user-specified depth limit is reached.[5] This component matches each piece of knowledge once per depth and aggregates any new relations that result from successful matches. Once all matching is complete, the new relations are added to working memory, the depth is increased, and the process is repeated. Inference terminates when the depth limit is reached or when no new elements are added at a given depth. This process is analogous to the one used by ICARUS (Langley et al., 2009) for conceptual inference.

Once inference finishes, if the tutor only provided a step to attempt (no accompanying SAI and feedback), then *skill execution* begins. This process begins by sorting skills by their utilities (highest to lowest) and incrementally matching them against the updated working memory. As soon as a match is found, it is executed to generate an SAI (i.e., an attempt). If no skills match, then the architecture issues a request for a example. Skill matching differs slightly from relational inference. Like inference, the conditions (both legality and heuristic) are matched against working memory. However once a match is found, a skill must also evaluate its classifier to determine if it applies with the current bindings. The classifier takes as input a modified version of the current working memory structure, where any object symbols (i.e., non-constant elements) bound to pattern-matching variables from the legality conditions are replaced with new internal constants that are uniquely determined by the names of the variables (see Figure 3.2). This

[5]For the relational knowledge in the current work, quiescence is always reached at depth one.



```
WM Structure             Legality Conditions:            Current Match:        Classification Structure
(name f1 "Field1")       (name ?input "Field1")          {?input: f1,          (name "#input#" "Field1")
(name f2 "Field2")       (name ?output "Field2")          ?output: f2,         (name "#output#" "Field2")
(name f3 "Field3")       (text-field ?input)              ?input-val: "2",     (name f3 "Field3")
(text-field f1)          (text-field ?output)             ?output-val: ""}     (text-field "#input")
(text-field f2)      →   (value ?input ?input-val)    →                   →    (text-field "#output#")
(text-field f3)          (value ?output ?output-val)                           (text-field f3)
(value f1 "2")                                                                 (value "#input" "2")
(value f2 "")                                                                  (value "#output" "")
(value f2 "3")                                                                 (value f2 "3")
```

Figure 3.2: An example of how the current working memory structure, and the current match of the conditions are used to generate the structure passed to a skill's classifier. Objects (i.e., non-constant elements) are colored blue, constants are colored red, and pattern matching variables are colored purple. All object elements in the current match are replaced with new constants uniquely determined by the variable they match. Note, constants and unmatched objects remain unchanged.

renaming scheme ensures that objects matched to the same variables are aligned; for example, different fields across different examples that match to the $?input$ variable are made to be identical for classification. This approach embeds the current match within the classification structure and enables classifiers that only support simple attribute-value representations to be sensitive to the current match.[6] If classification fails, then pattern matching continues. However, if classification succeeds, then the skill fires and its effects are deposited in working memory (after any functions are evaluated and replaced with their resulting values).

Note, that unlike most production systems, which compute all possible skill matches and then perform conflict resolution over these matches, the current system front-loads conflict resolution by selectively matching skills in the order of their utility. This approach is substantially more efficient, particularly when there are a large number of possible matches (e.g., when matching skills learned in a general-to-specific fashion). However, the current approach is somewhat limited in terms of its expressiveness because it cannot describe which matches of a particular skill are preferable to others. Currently, the system treats all matches of a skill as equivalent and executes the first match found, with matching using depth-first search to bind the most constrained variables to their least constraining values first, a common constraint-satisfaction heuristic (Russell & Norvig, 2009).[7] However, future work should explore how to incorporate binding preferences into the pattern-matching process, so that more preferable matches are generated first.

The final performance component, *how-search*, activates after relational inference in situations where the tutor has provided an SAI and feedback (i.e., when they are providing examples or feedback on prior attempts). This component constructs explanations of how the provided SAI could have been generated given the updated working memory structure. This process is effectively a self-explanation process (see VanLehn et al., 1991) and is similar in spirit to ICARUS's use of means-ends analysis to complete partial problem-solving traces provided by experts (Li et

[6]When using these simple classifiers, each unique relation is treated as a non-relational feature. For example, (name f3 "Field3") becomes a Boolean feature with the name '(name f3 "Field3")'.

[7]When multiple variables are equally constrained, or when possible values are equally constraining, expansion order is non-deterministic.

al., 2009). In this case, the expert provides a single SAI (e.g., SAI ”Field1” “UpdateText” “5”) to be explained. In response, the system applies skill knowledge in a forward-chaining fashion up to depth one (all skills have a single SAI effect, so search does not need to proceed further) using the same matching procedure as skill execution.[8] However, unlike skill execution, SAI effects are not added to working memory. Instead, they are compared with the provided SAI and all skill instantiations that yield the provided SAI are returned as explanations. If no explanations are found, then the search is repeated a second time, ignoring the skill’s heuristic conditions and classifier (i.e., skills are matched based only on the legality conditions). This second search terminates as soon as the first skill instantiation that explains the SAI is found; it does not compute all matching skill instantiations because it is often too expensive to compute them all.

If no explanations are found using the existing skill knowledge, then the component engages in a forward-chaining search using overly-general operators. For this search, all overly-general operators are applied to working memory up to a user-specified depth (the models in the current work expand operators to a depth of two). This procedure is similar to relational inference, but it keeps track of the depth at which inferred elements are added and the operators that generated them. Elements that already existed in memory prior to how-search are assigned a depth of zero. Unlike skills, overly-general operators do not produce SAI effects to compare, so instead the system searches for any occurrences of the terms in the SAI being explained (e.g., “Field1”, “UpdateText”, and “5”) in the updated working memory structure. If occurrences of a particular term are found, then the system selects the shallowest occurrence[9] and generates a trace of the operators that produced the element containing it. If a selected element occurs at depth zero of working memory, then no operators support it. In these cases, the system generates a trace containing a special “variablize element” operator, which has a single condition and a single effect that is a variablized version of the element it supports (see Figure 3.3 for an example). These discovered traces constitute explanations of the provided SAI. If there are no occurrences of a particular constant, then the system leaves them unexplained. Once all constants have been processed, how-search returns the traces it found and terminates. The current implementation also allows the tutor to provide simple type constraints on the SAI constants (e.g., that “5” must be a text value). These optional constraints are useful for excluding nonsensical explanations, such as explaining a text value as the x position of one of the elements in the interface (if it happened to have “5” as an x position).

3.3 Learning Components

Once how-search has completed, it passes the explanation traces it discovered to the four learning components, which create and update skills in response. The first component, *how-learning*, creates new skills in situations where the explanation traces from how-search contain overly-general operators. In these cases, the component uses a form of explanation-based generalization (Dejong & Mooney, 1986) to compile the how-search traces into legality conditions and effects for a new skill. In particular, how-learning regresses over the trace structures to replace constants

[8]The tutor can optionally provide a skill label with the SAI and feedback, which restricts the search to only include skills with the same label.

[9]If there are multiple occurrences at the shallowest depth, then one is randomly chosen.

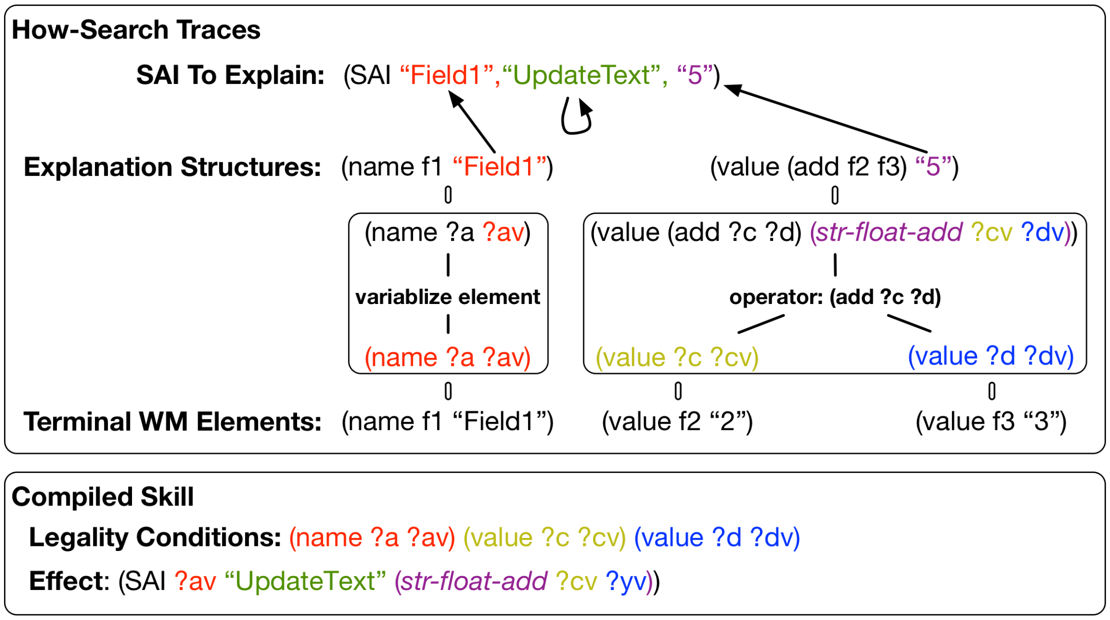

How-Search Traces
SAI To Explain: (SAI "Field1","UpdateText", "5")
Explanation Structures: (name f1 "Field1")
(value (add f2 f3) "5")
(name ?a ?av)
variablize element
(name ?a ?av)
(value (add ?c ?d) (str-float-add ?cv ?dv))
operator: (add ?c ?d)
(value ?c ?cv)
(value ?d ?dv)
Terminal WM Elements: (name f1 "Field1")
(value f2 "2")
(value f3 "3")
Compiled Skill
Legality Conditions: (name ?a ?av) (value ?c ?cv) (value ?d ?dv)
Effect: (SAI ?av "UpdateText" (str-float-add ?cv ?yv))

Figure 3.3: An example how-search trace and the resulting skill that is compiled from them. This skill has a single effect, which is generated by replacing constants in the Selection-Action-Input (SAI) being explained with the variablized elements that support them, and legality conditions, which are extracted from the leaves of the explanation structure. The arrowed lines show which elements support the SAI constants, the double-stroke lines represent unifications, and the single lines without arrows represent the mappings between conditions and effects.

in the SAI with appropriate variablizations. The variablized SAI becomes the effect for a new skill. Additionally, this process computes the legality conditions that must be met to successfully evaluate this new effect. To start this process, all pattern-matching variables that appear across the trace structure are renamed uniquely to ensure there are no name collisions during regression. Next, constants in the SAI are replaced with the particular effects elements that explain them.[10] The system then performs a breadth-first regression over the trace structure, replacing all occurrences of each variable with their unifying elements (for unification details see Dejong & Mooney, 1986). This process continues to the leaves of the traces, stopping just before replacing variables with their respective working memory elements, and then the conditions at the leaves of the traces are extracted as legality conditions for the new skill. Figure 3.3 shows an example of how-search traces and the skill that is compiled from them. Once a skill has been compiled, it is added to skill memory and the how-search output is updated to reflect how the new skill (and its particular instantiations) explains the SAI. Note, if the tutor provided a skill label, then it is assigned to the newly created skill.

Once skill applications that explain the provided SAI have been identified, either from pre-existing skills or those just created via how-learning, the *where-learning* component triggers, which takes the tutor feedback (positive or negative) and each skill application as input and updates the heuristic conditions of the underlying skills so that they better cover their positive

[10]Unexplained SAI are ignored—they are left as constants in the SAI and no conditions are created.

applications, but not their negative ones.[11] There are many possible approaches to relational condition learning, but all the models in this thesis utilize a very simple tutor-specific learner that memorizes name-relations that refer to elements bound in the legality conditions of applications labeled by the tutor as positive. For example, for the compiled skill in Figure 3.3, the learner might acquire disjunctive[12] heuristic conditions given two different positive application: (name ?a "Field1"), (name ?c "Field2"), (name ?d "Field3") OR (name ?a "Field1"), (name ?c "Field2"), (name ?d "Field4"). This simple approach, which learns separate disjuncts for unique sets of name relations occurring in positive applications, is extremely fast and works surprisingly well in most tutors. However, the approach produces effectively no generalization across interface elements. To support more powerful generalization, the system could instead use incremental specific-to-general or general-to-specific relational learners. However, pilot studies suggested these approaches perform only marginally better (in tutoring domains) and take substantially longer to run, so I did not explore them further.

Once where-learning completes, the system activates its third learning component, *when-learning*, which learns heuristic conditions for skills. This component takes the tutor feedback and each skill application as input. A classification structure is built from each skill instantiation by duplicating the current working memory structure and replacing objects bound to pattern-matching variables from the legality conditions with constants uniquely determined by the names of the variables (see Figure 3.2). Then, each of these structures is paired with the tutor feedback and passed as training data to a concept learning algorithm. Currently, the system is designed to support incremental algorithms, such as COBWEB (Fisher, 1987) or TRESTLE (MacLellan, Harpstead, Aleven, & Koedinger, 2016), but it also has an interface to the Scikit-learn library (Pedregosa et al., 2011), which implements many non-incremental concept learning approaches.[13] In order to utilize these non-incremental approaches, the system maintains a separate memory of all previous training data and, whenever it gets a new datum, it updates this memory and re-learns an entirely new classifier from all the available examples. Additionally, for any approaches that do not support relational representations (e.g., COBWEB or any of the Scikit-learn approaches), each relation is simply treated as a Boolean attribute.

Once the skill classifiers have been updated, the *which-learning* component activates. This final component updates each skill's utility given the available applications and feedback. Currently, the system updates the utility to reflect the average correctness (i.e., accuracy) of each skill based on counts of the positive and negative applications of each skill that it maintains. However, in future work, this system might implement alternative approaches, such as reinforcement learning.

[11]When updating a newly created skill, the component first initializes the heuristic conditions as determined by the particular where-learning strategy.

[12]Conditions currently only supports top-level disjunction, where each disjunct is separately evaluated at performance time.

[13]I do not believe these non-incremental approaches are good models of human learning, but many of these approaches have high accuracy, which is desirable for tutor authoring.

3.4 The Decision Tree and Trestle models

The remainder of this thesis explores two models, the Decision Tree and Trestle models, cast within the Apprentice Learner Architecture. These models implement the performance and learning components using the approaches described in the previous two sections, but differ in their approach to when-learning: the first uses a non-incremental decision tree learner (Pedregosa et al., 2011; Quinlan, 1986) and the second uses Trestle, an incremental learner (MacLellan, Harpstead, Aleven, & Koedinger, 2016). Both models are given identical initial knowledge configurations. In particular, they have relational knowledge for inferring equality of values (see value-equality in Table 3.1), determining editability of interface elements (such as text fields or drop-down menus), and computing grammar relations on values. The last kind of knowledge is similar to the unigram knowledge shown in Table 3.1; but instead of extracting words, it parses the provided value using a pre-trained probabilistic context-free grammar and adds elements describing generated parse trees to working memory. Specifically, it adds elements representing the nodes in the parse tree as well as left-tree and right-tree relations that describe how these nodes relate. Finally, it adds value relations that describe the strings these nodes represent. The probabilistic context-free grammar used by both models was trained on a large corpus of both English text and math equations, extracted from the "Self Explanation sch_a3329ee9 Winter 2008 (CL)" (Ritter et al., 2007) and "IWT Self-Explanation Study 0 (pilot) (Fall 2008)" datasets downloaded from DataShop (Koedinger et al., 2010), using the approach described by Li, Schreiber, et al. (2012).[14] In addition to this relational knowledge, both models have overly-general operators for adding, subtracting, multiplying, dividing, rounding, and concatenating values. Table 3.2 shows the add and concatenate operators; the other operators are almost identical, but perform their respective operations.

These preliminary models were chosen to be similar to the SimStudent model (Li, 2013; Matsuda et al., 2007), but different in a number of key respects. In particular, the depth-limited how-search in the current models (where the shallowest explanations are chosen) is similar to the iterative-deepening function search used by SimStudent. However, the current approach (for both models) is incremental, only searching for explanations of the most recent example, whereas SimStudent maintained all prior examples and searched for single explanations that cover all examples with the same skill label. This non-incremental approach is particularly troublesome when no single explanation exists for all the examples with a particular label. In these cases, SimStudent executed iterative deepening to the maximum depth before splitting the most recent examples off into separate disjuncts (of the same skill label) and repeated iterative deepening again on these disjuncts. This entire process was repeated each time a new example was received. While this non-incremental approach sometimes leads to more concise skills, the incremental approach almost always converges to the same explanations, is substantially faster, and aligns better theoretically with other models that use incremental explanation-based approaches to acquire new skills, such as CASCADE (VanLehn et al., 1991) and STEPS (Ur & VanLehn, 1995).

The new models also differ from SimStudent in the approach to where-learning. In particular, SimStudent requires the tutor to annotate any fields that are relevant to the current

[14]This grammar, however, is not part of the theory and could have been built by hand.

skill applications and learns generalized retrieval patterns for extracting these "foci of attention." These patterns are learned in a specific-to-general fashion given positive examples of each focus of attention.[15] In contrast, the new models do not require the tutor to annotate relevant elements because general patterns for retrieving necessary elements (i.e., the legality conditions) are automatically extracted using explanation-based generalization. Additionally, the name-memorizing approach from the current model yields identical behavior to the specific-to-general approach for tutor interfaces where little generalization is necessary. For example, with skills that only apply in two different situations, both approaches require at least one positive example of each situation and yield equivalent generalizations. The approaches differ in situations where more generalization is necessary, but, as I will show in the following chapters, this mechanism is sufficient for a wide range of tasks.

Both models differ from SIMSTUDENT in their approach to when-learning. The decision-tree learner is most similar to SIMSTUDENT, which uses FOIL (Quinlan & Cameron-Jones, 1995) to induce a classifier given the values of the tutor provided foci of attention. Although FOIL supports relational learning, this capability is not actually necessary within SIMSTUDENT because all object mappings are provided by the tutor (the foci of attention) or generated when matching skills. Thus, a decision tree learning approach that uses grounded relations as features produces nearly equivalent behavior. The key differences between these models and the TRESTLE model is that the latter learns incrementally, whereas the former are both non-incremental. Beyond these differences, SIMSTUDENT's approach to when-learning also has a few key limitations that have been addressed in the current models. Specifically, it requires the tutor to annotate interface elements (i.e., provide foci of attention) and only trains on the values of these elements (e.g., the text they contain) and features computed over these values. In contrast, the new models do not require the tutor to annotate elements, and for classification they use the entire working memory structure, where any objects bound in the legality conditions are replaced with constants uniquely determined by the variable names. This approach enables the newer models to be sensitive to both the values of interface elements, as well as how they relate to one another. This additional information is particularly useful with approaches that explicitly leverage structural and relational information, such as Trestle, but the renaming scheme also enables algorithms that do not explicitly support these kinds of information to benefit.

Finally, the new models also share many relations to other previous work. At the architecture-level, how-learning acquires macro- and primitive operators, where-learning acquires heuristic conditions, when-learning acquires preference knowledge for accepting or rejecting operators, and which-learning acquires numeric preference knowledge for ordering operators. Given these mappings, the architecture unifies much of the previous apprentice learning theory, and it should be possible to instantiate many of the prior learning models within it. For example, the STEPS and CASCADE systems use explanation-based learning of correctness for how-learning, which is similar to the how-learning approach of my models. These prior accounts also use analogical (CASCADE) and discrimination (STEPS) approaches to learning operator preferences, which are similar to the approaches used by the TRESTLE (analogical) and DECISION TREE (discrim-

[15]This approach only uses hierarchical relations between interface elements (a subset of the working memory structure), so is limited somewhat in its expressiveness. For example, it cannot learn conditions that pertain to the values of interface elements.

ination) models. It should also be possible to produce model configurations that are analogous to ACM (Langley & Ohlsson, 1984) or BUGGY and DEBUGGY (VanLehn, 1983), and support similar kinds of investigations. In particular, researchers could train agents by providing them with buggy, rather than correct, student traces, and the agents would induce skill models that explain students' buggy behavior.

Chapter 4

Efficiently Authoring Expert Models: A Case Study

4.1 Introduction

Now that I have described the Apprentice Learner Architecture, and the models set within it, I next turn to showcasing how one of the latter, the DECISION TREE model, supports efficient tutor authoring.[1] In prior work, Matsuda et al. (2014) showed that SIMSTUDENT can acquire an equation solving expert model given demonstrations and feedback. Subsequent work (MacLellan et al., 2014) estimated the time it would take the average trained developer to author an equation solving expert model using either SIMSTUDENT or Example-Tracing, a widely used authoring-by-demonstration approach. This work showed that authoring with SIMSTUDENT takes substantially less time than Example-Tracing because it generalizes from its training, whereas Example-Tracing does not perform any generalization.

These initial results are promising, but they come with a number of caveats. First, equation solving is a well-studied tutor domain and, as a result, this prior work was able to provide SIMSTUDENT with domain-specific prior knowledge (e.g., how to extract coefficients from terms in the provided equations) that bolstered its efficiency. It remains to be seen how viable this authoring approach is for domains where domain-specific prior knowledge is unavailable. Additionally, for comparison purposes, this prior work ignored one of the key capabilities of the Example-Tracing approach, namely mass production, which lets authors variablize previously authored problem-specific content and then instantiate it for many different problems. This approach is essentially a way for authors to manually generalize Example-Tracing expert models (called *behavior graphs*) to all problems that share isomorphic problem spaces. As generalizability is one of the key dimensions on which SIMSTUDENT outperformed Example-Tracing in prior work, it is unclear how the two approaches would stack up when authors can use mass production.

Based on these limitations, the current chapter investigates two questions: (1) is authoring with simulated students a viable approach when domain-specific knowledge is not available, and

[1]For clarity, the current chapter only focuses on one model, but the TRESTLE model could also be used for this purpose.

(2) how does the approach compare to Example-Tracing with mass production? To investigate these questions, I describe how to author a novel tutor for experimental design that teaches the control of variables strategy using both the DECISION TREE model and Example-Tracing, then evaluate the efficiency of each approach. If the DECISION TREE model can learn this task, then it suggests that authoring with simulated students is viable even when domain-specific prior knowledge is not available. Additionally, when evaluating the efficiency of authoring with Example-Tracing, I assume that, whenever possible, mass production happens for *free*. This optimistic estimate of the time needed to mass produce content provides a more aggressive Example-Tracing benchmark for assessing the DECISION TREE model's efficiency. After presenting my evaluation of these two expert-model authoring approaches, I conclude the chapter by discussing the limitations of each approach.

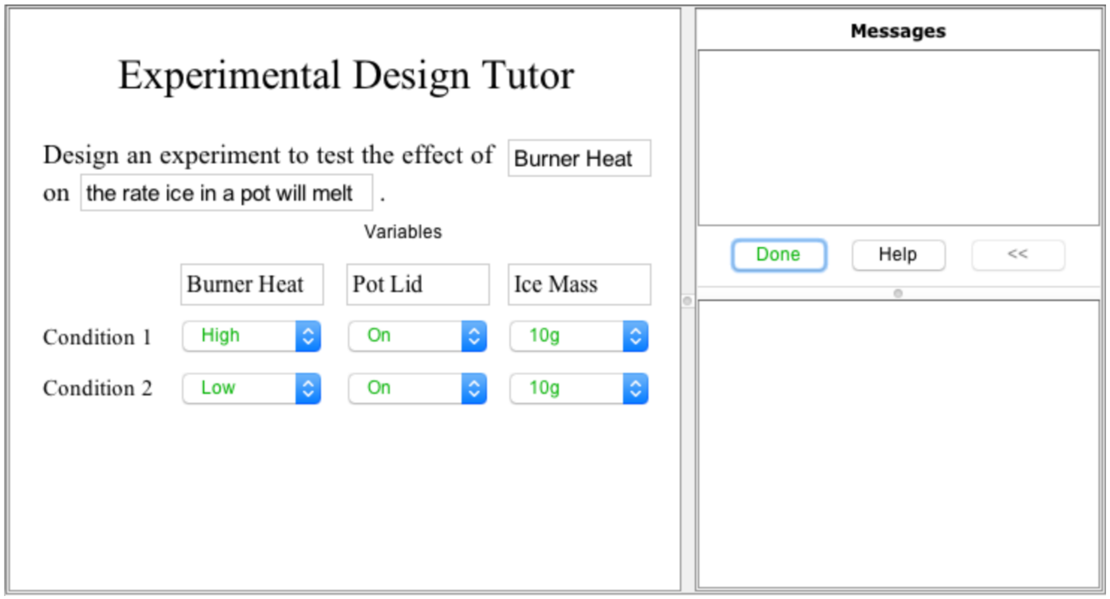
4.2 Experimental Design Task

Prior work has found that the ability to create well-designed experiments using the control of variables strategy can be improved by direct instruction (Chen & Klahr, 1999), and that tutoring middle school students on this strategy improves their ability to design good experiments (Sao Pedro et al., 2009). Thus, to demonstrate the authoring capabilities of the DECISION TREE model and Example-Tracing approaches, I decided use them to author a novel tutor for experimental design.

To coach students in designing good experiments, I created the tutor interface shown in Figure 4.1, which scaffolds students in constructing two-condition experiments that test the causal relationship between a particular independent variable and a particular dependent variable. A problem within this interface presents as a relationship to test (the effect "Burner Heat" on "the rate ice in a pot will melt"), available independent variables to manipulate ("Burner Heat," "Pot Lid," and "Ice Mass"), and values that these variables can take (the heat can be "high" or "low", the lid can be "on" or "off', and there can be "10g" or "15g" of ice). Within this framework, the desired system tutors students on how to solve problems using the control of variables strategy, which states that the only way to causally attribute change in a dependent variable is to manipulate the value of an independent variable while holding all other variables constant. More specifically, it gives students positive feedback when they pick values for the target independent variable that differ across conditions and values for non-target independent variables that are the same across conditions.

Although it appears simple to build an expert model for this task, from an authoring-by-demonstration perspective it is deceptively challenging. The key difficulty lies in the combinatorial nature of problems in this interface. For example, for the problem shown in Figure 4.1 there are eight unique solutions to the problem. Each solution requires seven steps (setting the six variable values and pressing the done button). Because the order of variable selection does not matter, there are then 721 ways to achieve each solution ($6! + 1$). Thus, there are approximately 5,768 ($721 * 8$ final solutions) paths each of length 7. This yields 40,376 ($5,768 * 7$) correct actions in the problem space.

This large number of correct actions, even for a simple problem with only three variables, each with two values, presents a challenge for non-programmers attempting to build an expert


Experimental Design Tutor

Design an experiment to test the effect of Burner Heat on the rate ice in a pot will melt .

Variables

	Burner Heat	Pot Lid	Ice Mass
Condition 1	High	On	10g
Condition 2	Low	On	10g

Figure 4.1: The experimental design tutor interface.

model using approaches that require them to demonstrate all correct ways to solve each problem (e.g., vanilla Example-Tracing). A common strategy authors use to overcome this problem is to constrain the number of correct paths by reframing the problem. The following tutor prompts for the interface in Figure 4.1 highlight how different problem framings affect the number of correct solutions and paths:

One solution with one path Design an experiment to test the effect of burner heat on the rate at which ice in a pot will melt *by assigning the first legal value to the variables in left to right, top down order as they appear in the table*.

One solution and many paths Design an experiment to test the effect of burner heat on the rate at which ice in a pot will melt *by assigning the first legal value to each variable, starting with condition 1*.

Many solutions each with one path Design an experiment to test the effect of burner heat on the rate at which ice in a pot will melt *by assigning values to variables in left to right, top down order as they appear in the table*.

Many solutions with many paths Design an experiment to test the effect of burner heat on the rate at which ice in a pot will melt.

These examples highlight how authors can change the number paths that are treated as correct. It is also possible for them to change the underlying problem space. For example, adding a fourth variable to the interface in Figure 4.1 would require two more steps per correct path (setting the variable for each condition), while adding another value to each variable increases the number of possible options at each step of the solution path. These examples illustrate that the number of correct actions in the problem space is not an inherent property of the domain, but rather arises from the author's design choices about particular problems and how they are presented.

When building tutors, authors typically have a number of pedagogical and theoretical goals, and authoring tools are a means by which these goals are achieved. However, if authoring tools fail to support authors in achieving these goals, then they are forced to make compromises. For example, they might be forced to choose problems that are realistic to author, but that are less pedagogically effective. This challenge can be described in terms of threshold and ceiling, from research on user interface software tools (Myers et al., 2000). More specifically, the threshold of a tool refers to how easy it is to learn and start using, while the ceiling refers to how powerful the tool is for expressing an author's ideas. I argue that, for authoring tools with low threshold (i.e., for non-programmers), the ceiling is not well understood.

4.3 Expert Model Authoring

To investigate the authoring capabilities and efficiency of the DECISION TREE model and Example-Tracing with mass production, I built an experimental design tutor using each approach. To create the tutor interface and author the expert models, I used the Cognitive Tutor Authoring Tools (CTAT) (Aleven et al., 2009). This toolkit provides a drag-and-drop interface builder, which I used to create the interface shown in Figure 4.1. The toolkit also supports two modes for authoring expert models without programming, Example-Tracing and Simulated Student.[2] Next I will describe how to author the experimental design expert model using each mode.

4.3.1 Authoring with Example-Tracing

When building an Example-Tracing tutor in CTAT, the author simply demonstrates steps directly in the tutoring interface. These demonstrated steps are then recorded in a *behavior graph*, which graphically represents the demonstrated portions of the problem space. Each node in the behavior graph denotes a state of the tutoring interface, and each link encodes an action that moves the student from one node to another. Many legal actions might be demonstrated for each state, creating branches in the behavior graph. Typically an author will demonstrate all correct actions, but they can also demonstrate incorrect actions, which they label as incorrect or buggy in the behavior graph. Once a behavior graph has been constructed for a particular problem, the tutoring system can use it to train students on that problem. In particular, the tutor traces a student's actions along the behavior graph and any actions that correspond to correct links are marked as correct, whereas off-path actions (i.e., that do not appear in the graph) or that correspond to incorrect or buggy links are marked as incorrect.

Figure 4.2 shows a behavior graph I authored for the experimental design tutor. The particular prompt chosen has eight unique configurations, so I demonstrated each unique configuration in the interface (corresponding to the eight paths in the figure). Along each path, the variable values can be set in any order. However, instead of requiring authors to demonstrate each unique ordering, the Example-Tracing approach lets authors specify that groups of actions can be executed in any order—drastically reducing the number of demonstrations necessary. Using this approach,

[2]I updated CTAT's SIMSTUDENT authoring mode, so that it sends all state and interaction information to the Apprentice Learner Architecture, which runs as a separate process outside of CTAT. This separation should make it easier to interface other authoring tools and systems with the architecture.

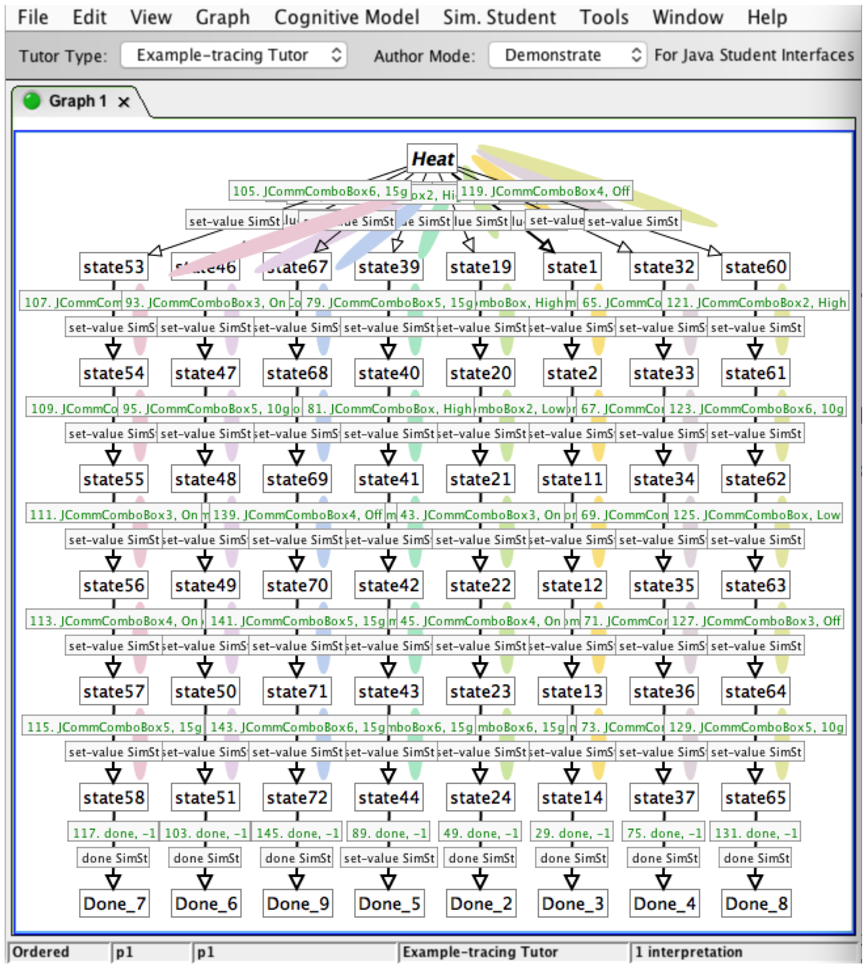

Tutor Type: Example-tracing Tutor
Author Mode: Demonstrate

Figure 4.2: A behavior graph for the experimental design tutor. The colored ellipsoids represent groups of actions that are unordered.

I specified that the actions to set variable values along each path are unordered (denoted in the behavior graph by colored ellipsoids).

Once I had successfully authored a behavior graph for the first problem, I next turned to generalizing it with mass production, so it could support other problems, such as designing an experiment to determine how the slope of a ramp affects the rate at which a ball will roll down it (Chen & Klahr, 1999). To create a template for mass production, I first authored the same problem as before, but instead of entering specific values in the interface, I entered variables, such as "%(variable1)%" instead of "Burner Heat." Then I created an Excel spreadsheet that had a row for each variable and a column for each problem, where each value in the table corresponds to a particular value for a particular variable in a particular problem. Using CTAT, I then mass produced my template and spreadsheet, which created separate grounded behavior graphs for each problem like the one shown in Figure 4.2.

This approach supports different problems that have identical behavior graph structure, such as replacing all instances of "Burner Heat" with another variable, "Ramp Slope". However, if a problem varies in the structure of its behavior graph, such as asking the student to manipulate a variable in the second column instead of the first (e.g., “Pot Lid” instead of “Burner Heat”) or to solve problems with a different number of variables (e.g., letting the burner heat be “high”, “medium”, or “low”), then a separate mass production template must be authored for each unique behavior graph structure. Given this limitation, to support experimental design problems with two conditions and three variables, each with two values, I ultimately had to author three separate mass production templates, one for each variable column being targeted.

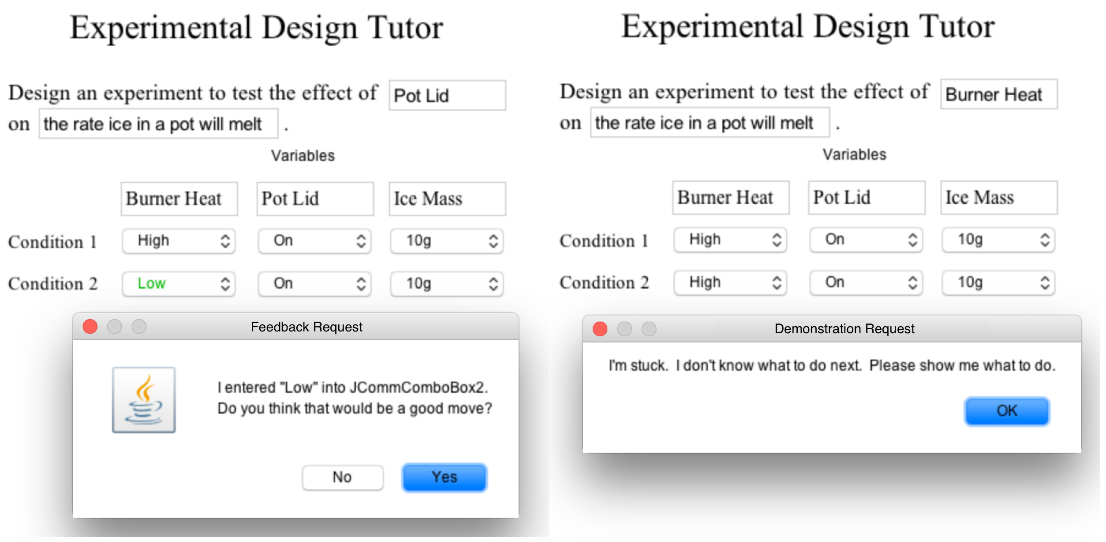
Next, I turn to evaluating the efficiency of the Example-Tracing approach. The completed model consists of 3 behavior graph templates (one for each of the three variable columns that could be manipulated). Each graph took 56 demonstrations and required eight unordered action groups to be specified. Thus, the complete model required 168 demonstrations and 24 unordered group specifications. Using estimates from a previously developed keystroke-level model (MacLellan et al., 2014), which approximates the time needed for the average trained author to perform each authoring action, I estimate that the behavior graphs for the experimental design tutor would take 26.96 minutes to build using Example-Tracing.[3] It is worth noting that the ability to specify unordered action groups offers substantial efficiency gains—without it, authoring would require 40,376 demonstrations, or 98.69 hours. Furthermore, with mass production, this model can generalize to any set of variables (although the number of variables or values cannot be changed).

4.3.2 Authoring with the DECISION TREE Model

To author an equivalent tutor using the DECISION TREE model, authors interactively train a computational agent to perform the task via demonstrations and feedback. In turn, the agent induces an expert model of the task. Rather than representing expert knowledge using a behavior graph, this model represents it as skills. Like action links in the behavior graph, skills describe the correct paths through the problem space. However, they are compositional and often much more compact. For example, the knowledge encoded in the three behavior graphs for the experimental design tutor might instead be represented using just three skills: one for setting the value of a variable to its first value, one for setting the variable value to its second value, and one for specifying that the problem is done (other skill decompositions are also possible).

For the DECISION TREE model to function, it needs access to a relational representation of each step in the interface. Fortunately, CTAT supports the ability to automatically generate these representations from interfaces that were authored using its drag-and-drop tools. In particular, it generates a representation with elements for each object in the interface (an element for each text label, text field, and drop-down menu) and relations to describe them (name, type, value) and how they relate (contains, for elements that contain other elements, and before, which describe the order elements appear when contained within another). Thus, authoring an interface in CTAT is essentially a way for non-programmers to author the sensors and effectors through which an agent perceives and interacts with the world. A key caveat of this approach is that particulars of how the interface was authored impact the agent’s learning and performance. For example, if

[3]The keystroke-level model estimates that it takes the average trained expert 8.8 seconds to demonstrate an action and 5.8 seconds to specify a group of actions as unordered. Additionally, I assumed mass producing problems took 0 seconds.



I entered "Low" into JCommComboBox2. Do you think that would be a good move?

I'm stuck. I don't know what to do next. Please show me what to do.

Figure 4.3: The DECISION TREE model asking for feedback (left) and for a demonstration (right).

the author creates the interface using a table (which contains rows and columns, which contain cells), then CTAT will generate a relational representation that affords generalization over rows and columns. In contrast, if the author creates a similar interface using multiple text fields, generalization will not be as easy for the agent, although it is usually still possible.[4] When authoring the experimental design tutor for the current work, I used multiple individual text labels, text fields, and drop-down menu elements, in order to show that even the worst case interface structures (i.e., that lack hierarchical structure) are still sufficient for agents to learn and perform.

Given a tutor interface and its relational representation, authoring an expert model with this approach is similar to Example-Tracing in that the simulated agent asks the author for a demonstration when it does not know how to proceed. However, when it already has an applicable skill, it executes it, shows the resulting action in the interface, and asks the author to provide correctness feedback on this action. Given this feedback, it refines its skill knowledge (i.e., its heuristic conditions, classifier, and utility). Figure 4.3 shows the agent asking for feedback and a demonstration. A key feature of this approach is that authors do not need to explicitly specify that actions are unordered—the agent learns general conditions on its skills that implicitly order its actions. One additional feature of the CTAT's simulated student mode is that it produces a behavior graph containing all actions the author has demonstrated or the agent has taken for each problem. Thus, this approach generates both skills and behavior graphs.

To author an expert model using the DECISION TREE model, I tutored it on a sequence of 20 experimental design problems presented in the tutor interface. Unlike Example-Tracing, I did not explicitly demonstrate every correct solution for each problem. Instead, the agent solved each problem a single way, and I provided it with demonstrations and feedback when requested. One

[4]Future work should explore the supplementation of an agent's relational knowledge, so it can compute its own spatial relations, such as left-of, above, and contains.

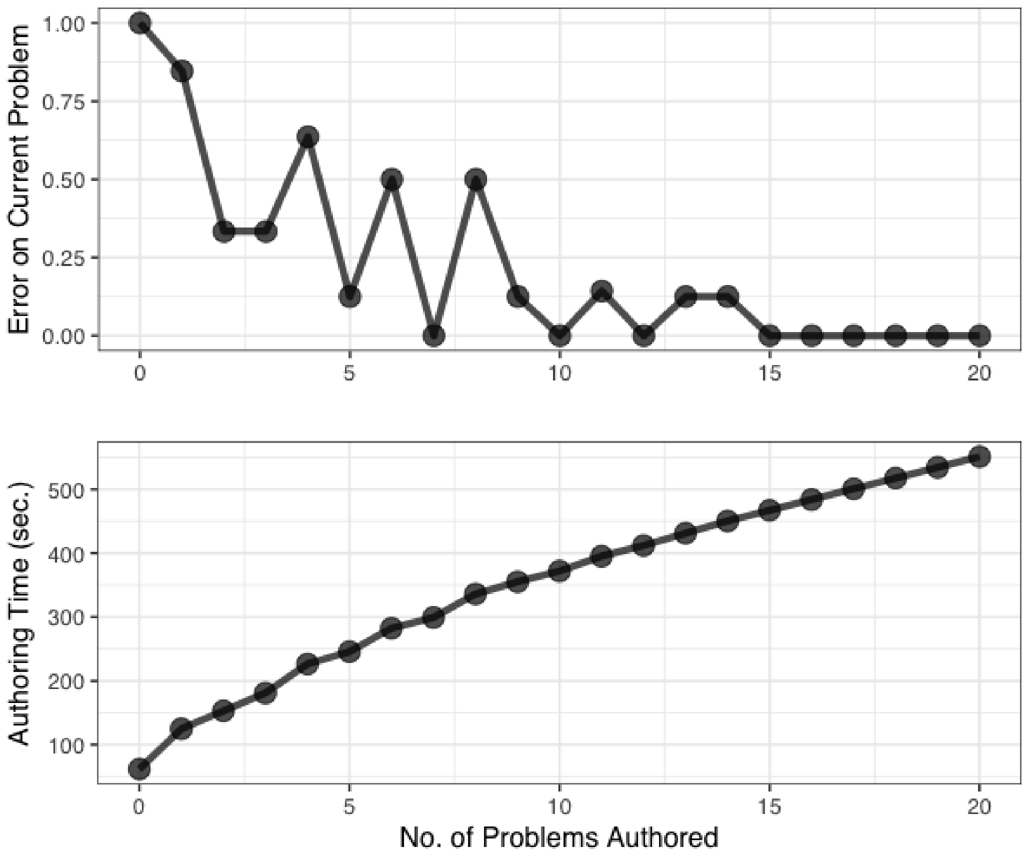


Figure 4.4: The average error (top) and cumulative authoring time (bottom) of the DECISION TREE model given the number of experimental design problems authored.

challenge when authoring with a simulated student is that it is difficult to determine when it has correctly learned the target skills. This is a problem also faced by teachers when they are trying to determine whether a human student has learned something correctly and by developers trying to verify that a program is correctly implemented. To address this problem, I use a solution that is common to both scenarios—testing the agent on previously unseen problems. In particular, I incrementally evaluate its performance on each subsequent training problem. The top graph in Figure 4.4 shows the performance of the agent over the course of the 20 training problems. This graph provides some insight into when the agent has converged to the correct skills. In particular, even though the agent solved the seventh problem without mistakes, it seems unlikely that it has converged because it made mistakes on the sixth and eighth problems. However, it seems reasonable to assume that it has converged by the end (i.e., after completing six problems in a row without errors).

One additional complication I encountered during training was that the agent has a tendency to learn a single correct strategy and then apply it repeatedly (e.g., always setting variables to their first legal value). To discourage this behavior, I provided demonstrations that displayed a range of strategies (e.g., sometimes setting variables to their second legal value). This varied training produced an agent that used multiple strategies. However, I believe this is an authoring problem that should be addressed more fully in future work. In particular, it seems to be an example of the exploration vs. exploitation tradeoff (Kaelbling et al., 1996), where an agent must decide between exploiting the strategies it already knows and exploring alternative strategies, potentially making more mistakes. The DECISION TREE model uses skills' utilities to determine which to try first, always executing higher utility skills. This approach encourages the agent to exploit its knowledge. However, when authoring, it is probably preferable to encourage exploration. One simple approach would be to uniform randomly select matching skills for execution (rather than those with the highest utility), which would likely reduce the agent's initial performance, but would encourage more exploration of the problem space.

Having successfully authored a skill model using this approach, I next turned to evaluating its efficiency. While authoring, I tabulated the number of demonstrations and feedback actions that I performed. Using these tabulations and the keystroke-level model from my prior work (MacLellan et al., 2014), I estimate that it would take the average trained expert 9.19 minutes to author an expert model for experimental design by tutoring the DECISION TREE model,[5] approximately one third of the time it takes to author the same tutor with Example-Tracing (about 27 minutes). Additionally, the bottom graph in Figure 4.4 shows the cumulative authoring time over the course of the 20 training problems. This curve is steeper at the beginning because the agent mainly requests demonstrations then. In contrast, by the end of training the agent only requests feedback, which takes much less time (2.4 vs. 8.8 seconds per request). One limitation of the current model is that it requests something from the author on every step (either a demonstration or feedback), so the cumulative authoring time curve never levels off. Future work should explore when the agent can stop requesting feedback on skill applications it is confident are correct. Such an approach might further reduce the authoring time.

4.4 Discussion

My primary finding is that both approaches support the construction of an experimental design tutor expert model, even though the DECISION TREE model did not have any prior knowledge specific to the domain. This high-level result suggests an affirmative answer to my first research question, whether authoring with simulated students is a viable approach even when domain-specific knowledge is not available. However, the current domain does not require any overly-general operators. For example, all input values (e.g., "High") are directly explained in terms of the available drop-down menu options (e.g., the first option). Thus, the primary challenge in this domain is to determine when the agent should pick the particular menu options (i.e., to discover the correct heuristic conditions and classifier). For tasks where the agent must construct more substantial explanations of demonstrations, it must have sufficient overly-general operators. I would argue that the six overly-general operators (add, subtract, multiply, divide, round, and concatenate) possessed by the current agent are reasonably general and support tutor authoring across a wide range of domains, and I will present evidence for this claim in later chapters.

[5]The original keystroke-level model estimates that, for SIMSTUDENT, it takes 10.4 seconds to provide a demonstration and 2.4 seconds to provide feedback. However, unlike SIMSTUDENT, the DECISION TREE model does not require authors to specify foci of attention, so demonstrating an action takes the same amount of time as Example-Tracing (8.8 seconds).

However, if the agent ever encounters a domain where it does not have sufficient overly-general operator knowledge, then it might not be able learn effectively. In the absence of this prior knowledge, the agent simply memorizes unexplained constants, which enables it to successfully learn problem-specific models similar to those acquired by Example-Tracing (i.e., it does not generalize beyond the demonstrations).

My second key result is that authoring the experimental design tutor using the DECISION TREE model took about one third of the time needed for the Example-Tracing approach, even when I assumed that mass production takes zero time. More specifically, the Example-Tracing approach consisted of authoring three behavior graph templates (one for each variable column being targeted), which I estimated would take the average trained expert about 27 minutes to author. An author could use these templates to mass produce any new problem, as long as they have three variables, each with two values. In contrast, the simulated student approach consisted of tutoring the DECISION TREE model on 20 experimental design problems, which I estimated would take approximately 9.2 minutes for the average trained expert. Overall this finding suggests that expert models can be efficiently authored by training simulated students. Further, the answer to my second research question is that authoring the experimental design tutor is more efficient with the DECISION TREE model than with Example-Tracing, even when using mass production.

Despite these promising initial results, I did encounter a number of issues when authoring with the agent. First, it was difficult to know when it had correctly learned the target skills, which differs from Example-Tracing where the completeness of the behavior graphs is always explicit. However, sometimes the behavior graphs are so complex that it is difficult to keep track of what has and has not been demonstrated. In the current work, I determined when the agent had correctly learned the skill by evaluating its performance during training. However, one complication of this assessment is that an agent can perform well using a single strategy, without knowing other strategies. The goal of training is to author an expert model that can tutor all of the strategies, not just one. To ensure that the agent learned all of the strategies, I had to explicitly demonstrate them to the agent, but future work should explore how to encourage the agent to explore multiple strategies such as having it randomly execute matching skills rather than executing those with the highest utility. This approach would let it discover these alternative strategies on its own, rather than requiring an author to explicitly demonstrate them.

From a pedagogical point of view, it is unclear whether alternative strategies need to be modeled in a tutor. Waalkens et al. (2013) have explored this topic by implementing three versions of an Algebra equation solving tutor, each with progressively more freedom in the number of paths that students can take to a correct solution. They found that the amount of freedom did not have an effect on students' learning outcomes, but argue that tutors should still support multiple strategies. There is some evidence that the ability to use and decide between different strategies is linked with improved learning (Schneider et al., 2011) and subsequent work (Tenison & MacLellan, 2014) has suggested that students only exhibit strategic variety if they are given problems that favor different strategies. Regardless of whether multiple strategies are pedagogically necessary, it is important that available tools support them so that these research questions can be further explored.

Finally, it is important to point out that although the agent-discovered skill model would not immediately support additional variables or values, it could be easily extended to support these cases with further training. In particular, adding a new variable would likely only require a few additional problems, so the where-learner can see that the current skills apply to the new drop-down menus. Similarly, updating an agent's model to support a new value would also only require a few problems, so the agent can learn when the new value should be selected. In contrast, when using Example-Tracing, adding a new variable would require an author to construct four new graphs, one for each column and adding a new variable would require the author to re-create the three graphs. In both cases the behavior graphs would be bigger. It is important to note that training the agent to support these new situations should not require the author to retrain the agent from scratch. One final feature of the skill model is that it is general enough to tutor a student on new variables and values even if they are not known in advance, whereas the behavior graph must be pre-generated when using Example-Tracing. This level of generality could be useful in inquiry-based learning environments (Gobert & Koedinger, 2011), where students could bring their own variables and values.

4.5 Conclusions

The results of my case study suggest that Example-Tracing and tutoring the DECISION TREE model are both viable approaches for non-programmers to create tutors. More specifically, I found that the agent-based approach was more efficient for authoring the experimental design tutor. However, this approach comes with a number of challenges related to ensuring that the authored model are both correct and complete, and it remains to be seen whether non-programming authors are comfortable navigating these challenges. In contrast, Example-Tracing was simple to use and it was clear that the authored models were complete, but it took almost three times longer to use. Overall, this analysis supplements prior work showing that Example-Tracing is good for authoring a wide range of problems for which non-programmers might want to build tutors (Aleven et al., 2009). However, authoring with the DECISION TREE model shows great promise as a more efficient approach—particularly for tutors that require multiple, complex, mass-production templates.

My analysis also identified situations where these approaches encounter difficulties. The Example-Tracing approach has mechanisms for dealing with unordered actions, but it struggles as the overall number of final solutions increases because each unique solution must still be demonstrated. Conversely, the DECISION TREE model has difficulties when there are multiple correct strategies. In these cases, it has a tendency to learn a single strategy and to apply it repetitively. This behavior has been observed in other programming-by-demonstration systems, and there exist techniques for demonstrating alternative strategies (McDaniel & Myers, 1999). Another approach would give the agent problems that favor different strategies to encourage variety, similar to tutoring real students (Tenison & MacLellan, 2014).

My findings also shed light on the thresholds and ceilings (Myers et al., 2000) of existing tutor authoring approaches. For example, hand authoring an expert model has a high threshold (hard to learn), but also a high ceiling (you can model almost anything with enough time and expertise). Example-Tracing, on the other hand, has a low threshold and a comparatively low ceiling (for problems with requiring many complex mass-production templates), though my results suggest the ceiling is higher than one might think. Functionality for specifying that actions in a behavior graph are unordered and mass producing content greatly amplify Example-Tracing. By contrast, the agent-based approach has a threshold similar to Example-Tracing, but it has a much higher ceiling because of its ability to generalize.

In conclusion, this work demonstrated the use of Example-Tracing and the DECISION TREE model for authoring a novel experimental design tutor. Additionally, my evaluation of these two approaches for this authoring task extended the prior work on authoring tutors with simulated students (MacLellan et al., 2014; Matsuda et al., 2014). In particular, my results showed that authoring with simulated students is viable even when domain-specific knowledge is unavailable. Further, they suggest that the approach can be more efficient than Example-Tracing, even when taking into account mass production. This work further advances the goal of developing a tutor authoring approach that is as easy to use as Example-Tracing (low threshold), but that is a powerful as hand authoring (high ceiling).

Chapter 5

Predicting Student Behavior and Testing Learning Theory

5.1 Introduction

In the previous chapter, I showed that apprentice learner models can support tutor authoring. However, it is still difficult for developers to build tutoring systems that are effective for learning. Fortunately, one branch of Educational Data Mining (EDM) research leverages data to improve our theoretical understanding of how people learn (Baker & Inventado, 2014; Romero & Ventura, 2010) and this developers can leverage this understanding to guide tutor design. Analogous to how data from the Large Hadron Collider, which was motivated by a strong theory, can be used to gain insights into physical laws, educational data can provide insights into the mechanisms underlying student learning. Although early EDM work explores this idea (Anderson et al., 1989; Ur & VanLehn, 1995), recent EDM research trends center on using statistical methods to discover the domain skills students learn within tutors (i.e., knowledge-component models) and estimating which skills students know based on their performance (i.e., knowledge tracing) (Baker & Inventado, 2014). While these research directions are important, the availability of educational data makes the EDM community well poised to contribute substantially towards our theoretical understanding of human learning.

Although many widely used predictive models of learning, such as Bayesian Knowledge Tracing (Corbett & Anderson, 1995) and Additive Factors Model (Cen et al., 2006), rely on existing theories of human learning, such as the *chunking theory of learning* (Newell & Rosenbloom, 1981), researchers rarely apply these techniques to educational data with the aim of improving psychological theory. Further, there are a number of barriers to using educational data for this purpose. First, many EDM approaches are only loose approximations of the theories on which they are based. For example, the Additive Factors Model predicts that improvements in human performance will follow a single logistic function, whereas many previous theories states that the improvements should follow a power function (Heathcote et al., 2000). Second, EDM models do not reflect the current state of learning theory. For example, recent studies of skill acquisition actually suggest that improvements should follows three distinct power functions, one for each phase of cognitive skill acquisition (Tenison & Anderson, 2015), rather than a single logis-

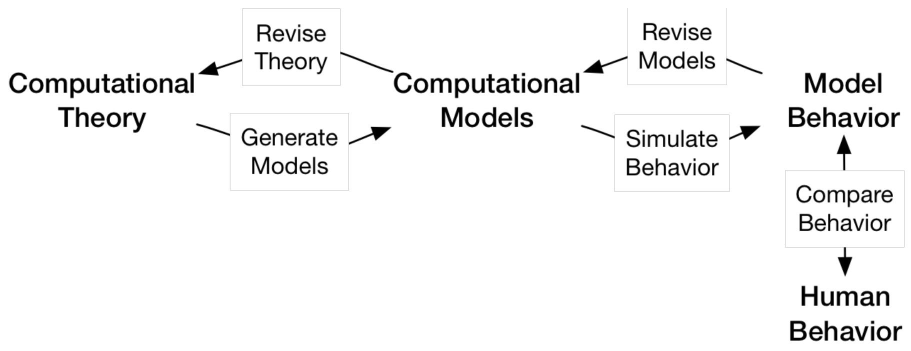


Figure 5.1: A depiction of how psychological theories, models, and behavior relate. Researchers use theories to generate models, which simulate behaviors that researchers compare to human behaviors. Researchers then use behavioral differences to inform future models and theories.

tic function. This disconnect between theories and models makes it difficult to draw inferences about the underlying theories given the fit of models to data. By more tightly connecting EDM models to psychological theory, we can leverage educational data to improve our understanding of the mechanisms behind human learning and, in turn, use these theories to improve predictions of student behavior.

To more tightly link a theory to models, researchers can develop computational theories (Newell, 1973), which describe the mechanisms that produce observed phenomena. Within this paradigm, a model presents as a specific implementation of these mechanisms that can be executed to simulate behavior, which then can be compared with observed behavior to test both the model and the underlying theory. The objective is not only to explain or "fit" relationships to observed data, but also, to predict relationships before data are observed. Figure 5.1 shows the iterative relationship between theories, models, and behaviors. I argue that this approach should be central to the field of EDM.

In the current chapter, I demonstrate the use of apprentice learner models for theory-driven prediction of student behavior. Unlike prior models of student performance, such as Additive Factors Model and its variants, apprentice learner models explain the mechanisms students use to acquire new knowledge from instruction. This mechanistic description lets one simulate learner behavior within an instructional environment and use the results to predict human behavior. Rather than arriving at a general conclusion, say that students learn differently from positive and negative feedback, this approach lets one explore possible explanations for the mechanisms driving these results.

Leveraging this capability, I use the Apprentice Learner Architecture to test two alternative theories of learning, in order to show how the architecture also supports theory development. The first, which corresponds roughly to SIMSTUDENT's theory of learning (Li et al., 2014), posits that humans learn when to apply skills *non-incrementally*; i.e., they relearn to classify the applicability of skills in light of all available training data. Although this theory seems implausible for lifelong learning, there is some evidence that, at least for the duration of a single equation solving tutoring session, it explains some errors that humans make (Matsuda et al., 2009). In contrast, the second theory posits that humans learn the conditions for skill application *incrementally*; i.e., they update how they classify skills given only the most recent training datum. This second theory seems more psychologically plausible and better aligns with other computational theories of learning (Feigenbaum & Simon, 1984; Fisher & Yoo, 1993; Langley, 1985).

To test these theories, I instantiated two models within the architecture: the DECISION TREE model and the TRESTLE model. The key difference is that the first uses a non-incremental decision-tree learner (Pedregosa et al., 2011; Quinlan, 1986) for when-learning, whereas the other uses Trestle (MacLellan, Harpstead, Aleven, & Koedinger, 2016), an incremental categorization tree learner. In all other respects the models are identical—they have the same prior knowledge and implementations for the other components.[1] This variation is reasonably simple and not particularly contentious, but prior models of human learning have adopted the assumption that humans learn when to apply skills non-incrementally (Li et al., 2014), so it seemed like a good initial test for of the general research approach.

To evaluate these models and test the theories underlying them, I use them to predict human behavior in a fraction arithmetic tutor that manipulated the order students received problems (Patel et al., 2016). Given this predicted behavior, I assess which models are capable of predicting the main instructional effects and which have better agreement with the human behavior. Using this approach, I provide evidence to support two claims:

1. Both models predict the main experimental effects of the problem ordering manipulation.
2. The incremental TRESTLE model better explains human behavior than the non-incremental DECISION TREE model.

Given these initial computational models and my data-driven theory development approach (see Figure 5.1), my ultimate goal is to develop a model that is consistent with available educational data sets, such as those found in DataShop (Koedinger et al., 2010) and other similar repositories. The development of such a model is possible under the assumption that differences in students' behavior are primarily due to differences in their experiences and prior knowledge (Matsuda et al., 2009), rather than differences in the mechanisms they use for learning.[2]

To pursue this goal, the general research program consists of generating different models of human learning using the Apprentice Learner Architecture, then connected them to the same intelligent tutoring systems that logged the data found on Datashop, and finally comparing the the behavior of these models in these tutors to human behavior. By analyzing differences between the models and humans, researchers can revise the underlying models and theories to reduce these differences and repeat the process. The following sections describe one iteration of this general research program, focusing on the fraction arithmetic tutor and one data set it collected. They describe how to stimulate student learning in this tutor and use simulation data to predict the human behavior. Additionally, they demonstrate the use of human data for testing which model (and theory) is better supported by the data. Finally, they discuss the implications of my results and potential directions for research.

[1]See chapter 3 for more details on these models.

[2]Under this view, individuals might have slightly different parameters for the same mechanisms; for example, individuals might have different thresholds for how widely or deeply they search when explaining an example.

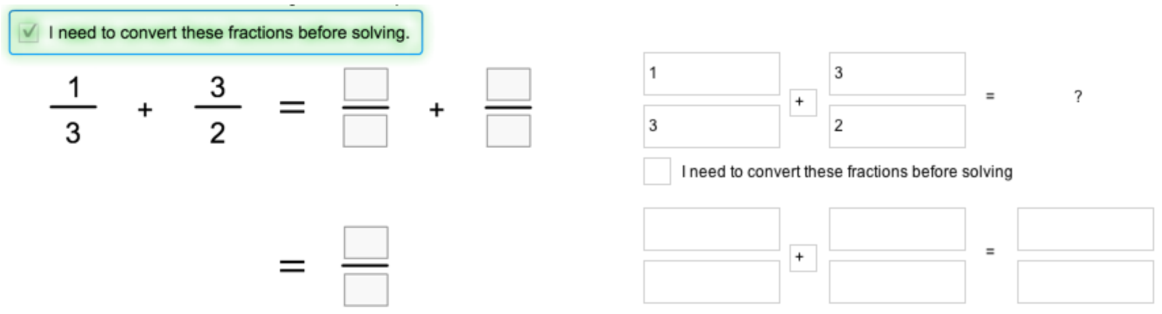

I need to convert these fractions before solving.
1/3 + 3/2 = □/□ + □/□
= □/□
1 3
+ =
3 2
?
I need to convert these fractions before solving
+ =

Figure 5.2: The fraction arithmetic tutor interface used by the human students (left) and the isomorphic, machine-readable interface used by the simulated agents (right). If the current fractions need to be converted to a common denominator, then students must check the “I need to convert these fractions before solving” box before performing the conversion. If they do not need to be converted, then they enter the result directly in the answer fields (without using the conversion fields).

5.2 The Fraction Arithmetic Tutor

For my simulations, I used the “Fraction Addition and Multiplication, Blocked vs. Interleaved” data set accessed via DataShop. Patel et al. (2016) collected these data during a classroom experiment to determine whether blocking or interleaving different types of fraction arithmetic problems produced better learning. These log data were generated by 79 students solving 24 fraction addition problems (ten with same denominators and 14 with different denominators) and 24 fraction multiplication problems in the tutor interface shown on the left in Figure 5.2. Some characteristics of this tutor are worth mentioning because they affect learning. First, it required students to check the “I need to convert these fractions before solving” box before even making the fields necessary for conversion visible. Additionally, for fraction addition problems that required conversion, the only strategy the tutor accepted for computing a common denominator was multiplying the denominators of the provided fractions (other common denominators were labeled by the tutor as incorrect).

Students in this data set were divided randomly into two conditions, blocked and interleaved. The students in the blocked condition received three blocks of problems: fraction addition problems with same denominators, then fraction addition problems with different denominators, and then fraction multiplication problems. The order of the problems within each block was randomized for each student. In contrast, the students in the interleaved condition received a random ordering of all problems with different types intermixed. This experiment was designed to test whether interleaving different types problems are better for learning than the blocking approach suggested by the Common Core State Standards (i.e., fraction addition with same denominators, then different, then fraction multiplication). The main finding of this experiment is that students in the blocked condition have better performance during learning, but students in the interleaved condition have better scores on a posttest taken after training (Patel et al., 2016). The aim of the current chapter is show that both the DECISION TREE and TRESTLE models predict these main effects, but that the TRESTLE model better accounts for human behavior.

5.3 Simulation Approach

In order to simulate human behavior in the fraction arithmetic tutor, I created an instance of each model (i.e., an agent) for each student and connected them to a machine-readable version of the tutor (shown on the right in Figure 5.2). This isomorphic tutor provides each agent with the same order of problems that the respective human student received. It is important to note that agents are not forced to take the same actions as their respective humans, so a tutor is necessary to provide the agents with appropriate hints and feedback during their training as the log data alone is not enough to conduct the simulation.

For my simulations, I created a new machine-readable version of the fraction arithmetic tutor. Creating this tutor consisted of authoring the tutor interface and behavior graphs to power it using the Cognitive Tutor Authoring Tools (CTAT) (Aleven, McLaren, Sewall, & Koedinger, 2006). The primary reason for creating a new tutor, rather than trying to interface with the original one, was to take advantage of software I had developed for interfacing CTAT with the Apprentice Learner Architecture. This software automatically trains each agent using the CTAT tutor and generates the appropriate relational representations for each tutor step.[3] Although authoring a new tutor takes effort, this approach works even when access to the original tutor is not available. Additionally, I have found it to be an invaluable part of understanding the peculiarities of each task. For example, I originally assumed that the fraction arithmetic tutor supported multiple strategies for computing the common denominator, but in understanding the tutor well enough to recreate it (through studying the log data and speaking with the author of the original tutor), I discovered that it only supported a single strategy.

Despite these benefits, there is a key limitation to this approach: it is challenging to create a completely isomorphic interface. In the current simulation, the human tutor hides the fields for converting fractions until the student checks the “I need to convert these fractions before solving” box, but the machine-readable tutor never hides them because the software interface between CTAT and the Apprentice Learner Architecture does not support hidden fields. The implication of this difference is that the simulated students can make errors (i.e., trying to convert fractions before hitting the conversion box) that are impossible in the human tutor. In general, there is a trade off between minimizing such differences and minimizing development time. In the current case, I ran preliminary simulations and found that the simulated agents rarely make these errors, so I proceeded without developing support for hidden fields.

In addition to tutored problems, the human students also took a posttest to assess their fraction arithmetic knowledge after using the tutor. This assessment was administered directly in the tutor, with the correctness feedback disabled. When working on these problems, students could enter any values in the text fields and they could check the “I need to convert these fractions before solving” box, which would reveal the conversion fields regardless of whether conversion was necessary. At any point in the problem, students could press the done button and advance to the next problem. Patel et al. (2016) assessed students in terms of the percentage of problems they solved correctly. Because the posttest was administered within the tutor, the simulated agents could take it after completing all the tutored problems. When agents took the posttest, hints and

[3]The original tutor was authored in CTAT, but it used a newer HTML5 interface and my existing software only supports Java interfaces.

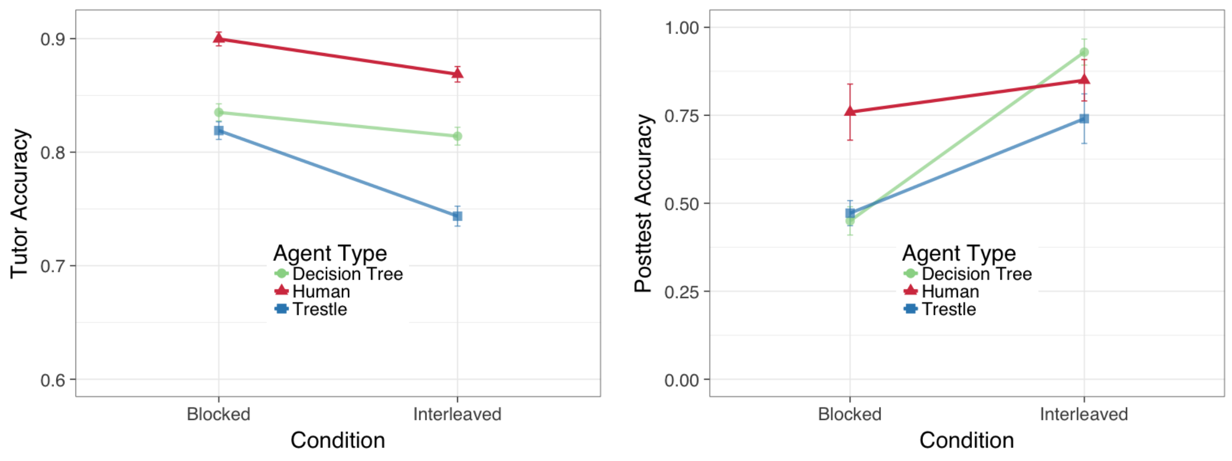


Figure 5.3: The average tutor (left) and posttest (right) accuracy in the fraction arithmetic tutor, separated by condition. The lines and whiskers denote the 95% confidence intervals.

feedback were disabled. If they got all the steps correct, then they were marked as getting the entire problem right and advanced to the next problem. However, if they made any mistakes, then they were immediately marked as getting the problem wrong and moved to the next problem. As with the humans, I assessed the simulated agents in terms of the percentage of problems solved correctly.

5.4 Model Evaluation

5.4.1 Instructional Effect Prediction

After simulating every student in the data set using each model, I conducted two evaluations using the human and simulated data. First, I examined the overall effect of instructional condition (blocked vs. interleaved) on both tutor and posttest performance for the humans and the two models. Figure 5.3 shows the overall tutor and posttest performance by condition for each type of agent (humans and the models). Both models successfully predict the effect of condition on both tutor performance and posttest (via logistic regression on tutor performance and linear regression on posttest performance). In particular, the humans have higher tutor performance in the blocked condition ($z = 6.663$, $p < 0.01$), and both the DECISION TREE ($z = 3.799$, $p < 0.01$) and TRESTLE ($z = 12.53$, $p < 0.01$) models predict this effect. In contrast, humans have higher posttest performance in the interleaved condition ($t = 1.86$, $p < 0.06$) and both the DECISION TREE ($t = 17.73$, $p < 0.01$) and TRESTLE ($t = 6.938$, $p < 0.01$) models predict this reversal.

These results support my claim that both models can predict the effect of a problem ordering manipulation; i.e., that humans in the blocked condition will do better in the tutor, but worse on the posttest. This finding suggests that, even though the models differ in their approach to when-learning, their shared mechanisms for the other performance and learning components are sufficient to predict these main effects. The results also indirectly support the models' underlying

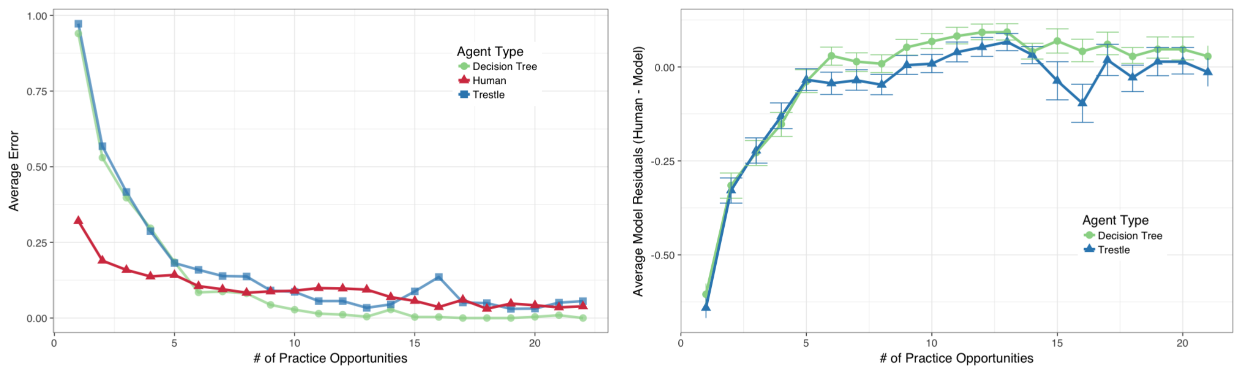


Figure 5.4: Overall learning curves for the humans and the two models in the fraction arithmetic tutor (left) and the model residuals plotted by opportunity (right). Model residuals are computed by subtracting the model prediction from the actual human performance (for a particular student on a particular opportunity of a particular skill). For model residuals, the 95% confidence intervals are also shown.

assumption that student behavior is primarily a factor of their experiences and prior knowledge, rather than difference in their learning and performance mechanisms.

5.4.2 Learning Curve Prediction

For the second analysis, I computed the error on each learner's first attempts of each step and generate overall learning curves using these values. In particular, I labeled each step in both the human and simulated data sets by the skill, or knowledge component (Koedinger et al., 2012), that it exercised. In this case, I labeled them by the field that was being updated (which roughly corresponds to the application of a particular skill) for each problem type.[4] Then, for each first attempt (the first time performing a particular step on a problem), I computed the prior number of opportunities a student has had to exercise the same skill and plotted the average error on first attempts (across all skills) by the number of prior practice opportunities.

The result of this process is the three learning curves shown in the left graph of Figure 5.4: one for the humans and one for each model. Unlike models that generate predictions by fitting a function to the student data, such as the Additive Factors Model, the learning curves generated by these models constitute *parameter-free* predictions of the human students' performance. There are a number of characteristics of these curves that are worth noting. First, there is a large difference between the models and humans on earlier opportunities. This suggests that many human students have prior knowledge of how to perform fraction arithmetic—they get more than 50% of the steps correct on the first practice opportunity. In contrast, the simulated agents start without any fraction arithmetic skills. However, by the fifth practice opportunity the performance of the humans and the models has begun to converge. Further, after the eighth opportunity, the DECISION TREE model consistently outperforms the humans, whereas the TRESTLE model more closely approximates the human performance. This difference in performance between the two

[4]This model, called the "Field" model, is currently the best fitting knowledge-component model on DataShop.

models at the higher practice opportunities is due to the different when-learning implementations and suggests that the TRESTLE model is a better approximation of the human behavior.

To ensure that these results are not simply a side effect of averaging over students and steps, the right graph of Figure 5.4 shows the model residuals plotted by opportunity. Model residuals were generated by subtracting each model's prediction (for a particular student on a particular opportunity of a particular skill) from the corresponding human student's performance. These residuals represent the average agreement between the humans and models on specific items. The key result of this analysis is that it agrees with my initial analysis of the learning curves. Mainly, both models are different from the humans on the first five practice opportunities, but the TRESTLE model better agrees on the higher opportunities (i.e., the residuals are closer to zero).

5.4.3 Theory Testing

To quantitatively evaluate which of the two models better accounts for human behavior, I fit two linear mixed-effects binomial regressions using the lme4 package in R (Bates et al., 2015), one for each model. Each regression has two fixed effects, one for the respective model prediction and one for the number of prior practice opportunities, a random intercept and slope (for prior practice opportunities) for each skill, and a random intercept for each student.[5] These regressions assess the explanatory power of the models, properly accounting for the non-independence of model residuals within each opportunity, skill, and student, similar to a repeated measures ANOVA.

These regression models show that the TRESTLE model (AIC $= 9512.8$, BIC $= 9566.7$) better fits the human data than the DECISION TREE model (AIC $= 9521.3$, BIC $= 9575.2$) in terms of both AIC and BIC.[6] Further, this difference in model fits is significant (via a likelihood ratio test, $\chi^2(0) = 8.49$, $p < 0.01$). These results support the claim that the TRESTLE model has better agreement with the human data than the DECISION TREE model. They also support the theory that humans learn the conditions under which skills apply in an incremental, rather than non-incremental, fashion—even within the span of a single tutor session. It is worth noting, however, that the model's when-learning approaches also differ in other regards (e.g., probabilistic vs. discrete). Even though it is unlikely that these other differences account for the different model fits, future work should explore these possibilities.

5.5 General Discussion

I have argued that the results of my simulation study support my two key claims. In particular, (1) both models predict the main experimental effects of a problem ordering manipulation and (2) the TRESTLE model better explains the human data. To support the first claim, I showed that, like the humans, both models have better tutor performance in the blocked condition, but better posttest performance in the interleaved condition. One caveat is that the models only *qualitatively* predict the experimental effect. They do not predict the absolute tutor and posttest scores for

[5]The corresponding R lme4 regression formula is: human.performance $\sim$ model.prediction $+$ opportunity $+$ (1 $+$ opportunity $|$ skill) $+$ (1$|$student).

[6]Although the absolute AIC and BIC scores have no meaningful interpretation, a lower score means the model fits better.

each student because they do not currently model the students' prior knowledge. Thus, one promising direction for future work would be to explore how to properly account for students' prior knowledge in order to improve the quantitative predictions. For example, the simulated agents might be pre-trained on additional fraction arithmetic problems, where the number of pre-training problems is proportional to the respective human student's pretest score.

In my second analysis, I showed that both models generate reasonable parameter-free learning curve predictions. To my knowledge, these results are the first example in the EDM literature of how student performance can be predicted in a completely theory-driven way without having to fit the models to the student data first. However, for earlier practice opportunities, both models predict that human performance should be much worse than the performance actually observed in the human data. As just mentioned, these differences are likely due to the prior fraction arithmetic skills that the human students bring to the tutoring system, which are not currently being accounted for in my models. In theory, the models are predicting what the human performance would look like if the students did not have any prior fraction arithmetic skills. While these exaggerated error rates might be useful for detecting transitions between skills, such as when using learning curve analysis to develop knowledge-component models (Corbett & Anderson, 1995), they also suggest an opportunity for improvement (e.g., by taking into account each student's prior knowledge).

Given the human and simulated learning curve data, I next turned to evaluating which of the two models better accounts for variation in the human behavior. For this evaluation, I performed a mixed-effects regression analysis, which showed that the TRESTLE model better explains the variation in student performance across students, skills, and opportunities. This latter result supports the theory that humans learn the conditions under which skills apply incrementally, rather than non-incrementally and provides an example of how my general research approach supports theory testing and revision.

5.6 Conclusions and Future Work

The results of this study have been encouraging, but they do not mean that the Apprentice Learner Architecture and the models cast within it offer a complete account of human learning. However, they do suggest the framework is flexible enough to support new hypotheses about learning and provides ways to test these hypotheses. In future work, I plan to explore several variations of the current theory and invite the community to extend it to explain other phenomena.

One affordance of the Apprentice Learner Architecture is that it facilitates a search among alternative theories and models. Not unlike existing techniques for searching the space of domain models (Cen et al., 2006), a search among alternative apprentice learner models would let researchers explore several hypotheses of human learning. For example, it is questionable whether how-, where-, when-, and which-learning are the correct combination of internal components. It may be that the TRESTLE model's when-learning approach is sufficient for both where- and when-learning, which would suggest that the current distinction between where- and when-learning is unnecessary and that a single learning component might produce more human-like simulated data. Alternatively, it could be argued that the architecture is incomplete because it has a fixed set of relational knowledge, rather than updating its relational knowledge given new experiences. This argument implies that a learning component for updating relational knowledge, effectively a what-learner, should be included in the architecture (Li et al., 2014). Beyond merging or adding architectural components, each component could be instantiated using different approaches. For example, the current where-learning approach carries out effectively no generalization, in that it memorizes the field names used in positive examples, but future models might explore more human-like approaches, such as learning heuristic conditions in a general-to-specific fashion (Langley, 1985). Exploring all of these possibilities could be framed as a search task over different models within the architecture to find those that generate the most human-like behavior.

This chapter compared model and human error rates, but the Apprentice Learner Architecture allows for finer-grained evaluation. Rather than comparing simulated and human learners on whether they performed a step correctly, researchers could compare learners in terms of their literal response on a step. This type of comparison would support the evaluation of theories of student misconceptions and lets researchers explore how misconceptions affect the particular responses students make, similar to the prior work on modeling student bugs (Langley & Ohlsson, 1984; VanLehn, 1983). Similarly, in this study I only compared performance on first attempts, because this is a common convention in EDM, but the high-fidelity simulation data would let me examine learner behavior beyond the first attempt. Ultimately, a unified theory of apprentice learning should account for all of the behaviors learners exhibit on their path to mastery.

As I have stated previously, I view the current state of the Apprentice Learner Architecture as incomplete. There are several aspects of learning that the architecture does not try to explain, such as the effects of delayed feedback (Schmidt & Bjork, 1992), the impacts of metacognition (Aleven, McLaren, Roll, & Koedinger, 2006), and the behavior of collaborative learners (Olsen et al., 2015). Crucially, however, the architecture is not fundamentally incompatible with these ideas. For example, which-learning, which updates skills' utilities, might utilize a reinforcement learning scheme to back-propagate correctness from delayed feedback. The role of metacognition might be accounted for with a more nuanced variation of a recognize-act cycle that takes into account metacognitive decisions. Finally, instantiating multiple apprentice learner models within the same environment and allowing them to generate demonstrations for each other could serve as an initial computational model of collaborative learning. These are just a few examples of how the structure of the architecture could be augmented to incorporate and test additional learning theories.

In conclusion, this chapter provided an initial proof of concept for my proposed research program (see Figure 5.1) and showed that the Apprentice Learner Architecture supports both theory-driven prediction of human behavior and data-driven theory development. Additionally, it constituted a first step towards a complete computational theory of apprentice learning. Not only do I believe that EDM can improve our fundamental theories of learning, but that it is uniquely positioned to do so. Using my proposed approach, researchers can use every tutored learning data set in the canon of EDM to test and advance learning theories. I hope that other EDM researchers will also see the potential of the Apprentice Learner Architecture and the proposed research paradigm, and I look forward to working with them to further develop our collective understanding of human learning.

Chapter 6

Cross-Domain Evaluation of Apprentice Learner Models

6.1 Introduction

In the previous two chapters, I showed that apprentice learner models can support tutor authoring, behavior prediction, and theory testing. In this chapter, I endeavour to convince the reader that my two preliminary models (the DECISION TREE and TRESTLE models) are general enough to support these pursuits across a wide range of tutoring domains, even though they draw on a small, fixed set of relational knowledge and overly-general operators. It is impossible to prove that these models will be able to support these applications in *any* tutor, so I instead focus on demonstrating their use in seven tutoring systems that teach different kinds of knowledge across a wide range of domains that include language, math, engineering, and science. I hope that this presentation will make clear the wide applicability of the current models and provide some insight into their current limitations. It is worth mentioning that this focus on generality and demonstrating functionality runs counter to the mainstream literature, which typically emphasizes stronger results in just one or a few domains.

To support a claim of generality, I have chosen to highlight the ability of these models to acquire knowledge across the types outlined in the Knowledge-Learning-Instruction framework (Koedinger et al., 2012), which characterizes knowledge in terms of whether it has constant or variable stimuli and responses.[1] Although almost all stimuli and responses are variable in some respect, this distinction is meant to capture the qualitative distinction between minor variability, such as the use of different fonts for a word, from more substantial variability, such as the use of different words.

Within this framework, I have chosen to simulate humans in tutoring systems that teach *associations*, *categories*, *skills*, or combinations of these types. Associations involve knowledge that has a constant stimulus and constant response (e.g., respond with the constant "small" whenever presented with the constant "小"). Slightly more general are categories, which encode knowledge with a variable stimulus and a constant response (e.g., respond with the constant "stable" when-

[1]This framework also distinguishes between knowledge that is explicit or implicit and that does or does not have a rationale, but I only focus on implicit knowledge without a rationale in the current work.

ever presented with a tower of blocks that is symmetrical, has a wide base, and has a lower center of mass). Finally, skills represent knowledge with a variable stimulus and a variable response (e.g., whenever presented with any two numbers with a plus sign between them respond with their sum).

Demonstrating that my computational models support these different types of knowledge suggests that they will apply more broadly to other tutoring systems that utilize these types. Additionally, Koedinger, Corbett, and Perfetti (2012) go on to hypothesize links between these knowledge types (e.g., associations) and the learning processes (e.g., fluency building) and instructional principles (e.g., spaced practice) that support them. I hope that future work will use my computational models to test these hypothesized links.

In addition to showing that these models are able to learn different kinds of knowledge, the goal of this chapter is to replicate the findings of the previous two chapters across multiple domains. Towards this end, it first investigates whether the two models can predict human behavior across multiple tutoring systems. Next, it evaluates the two models using student data from all seven domains to determine which better aligns with human behavior. The previous chapter found that the TRESTLE model better accounts for human behavior than the DECISION TREE and a replication of this finding will further demonstrate the use of these models for theory testing and provide stronger evidence to support the theory that humans incrementally learn when to apply skills, rather than non-incrementally—as the SIMSTUDENT model of learning assumes (Li, 2013). Finally, this chapter evaluates the authoring efficiency of the models in all seven domain and compares their efficiency with Example-Tracing.

Before presenting the results of these investigations, the chapter describes the seven tutors investigated in this study as well as the details of the human data collected with these tutors. In order to simulate learning across these systems, it was necessary to construct machine-readable versions of each tutors. The chapter reviews each of these isomorphic systems and notes discrepancies between the human and machine versions of the tutors. Next, the chapter reviews the general simulation approach, presents the results of simulations across the seven tutors, and discusses the general implications of these results. Finally, it discusses conclusions and directions for future work.

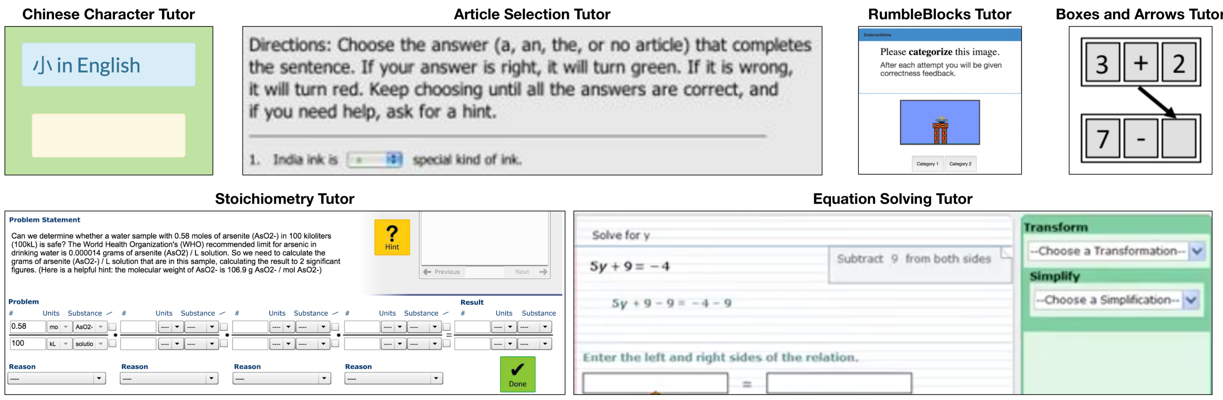
6.2 Tutoring Systems

To demonstrate the generality of my apprentice learner models, I applied them to seven tutoring systems that have human data available in DataShop (Koedinger et al., 2010): the Chinese character tutor (Pavlik et al., 2008), the article selection tutor (Wylie et al., 2009), the RumbleBlocks stability tutor (MacLellan, Harpstead, Aleven, & Koedinger, 2016), the fraction arithmetic tutor (Patel et al., 2016), the boxes and arrows tutor (Lee et al., 2015), the stoichiometry tutor (McLaren et al., 2006), and the equation solving tutor (Ritter et al., 2007). These tutors were selected because they teach multiple types of knowledge and span a wide range of content domains. Thus, a finding that the models support the three applications explored in the previous two chapters (tutor development, behavior prediction, and theory testing) across all seven domains is strong evidence for the generality of the models.

The previous chapter already presents the fraction arithmetic tutor, and the human data collected in it, so other than mentioning that it teaches categories for labeling whether conversion is needed and skills for performing the arithmetic, I will exclude it from my discussion here. This leaves six new tutoring systems (interfaces shown in Figure 6.1) that I will review in turn.

First, the Chinese character tutor is essentially a sophisticated flashcard program that tasks students with translating particular characters into either English or Pinyin. The tutor, which focuses on teaching students associations (i.e., knowledge with a constant stimulus and constant response), was designed to use an optimized spacing model to improve students' retention of knowledge. For my simulations, I used human data from the "Chinese Radical Transfer Fall 2007" data set accessed via DataShop. These data were collected from 94 students who used the tutor as part of their Chinese I university level course, within which they were asked to complete 15 minutes of training within the tutor per unit over the course of the entire semester (Pavlik et al., 2008). When students opened the tutor, they could choose between the "Optimized" version of the tutor, which used the optimized spacing model, and the "Flashcard" version of the tutor, which randomly ordered practice of either all the available characters or just the characters from a user-selected unit. This latter version also removed items that students got right from the practice "deck" for the current session. Over the semester the tutor presented the students with 555 unique character-translation pairs, and logged 61,323 steps. Pavlik et al. (2008) showed students' performance improves as they use the tutor, but they found no effect of tutor version (optimized or flashcard) on learning gains.

The next tutor is the article selection tutor, which scaffolds students in selecting the correct article (a, an, or the) for English sentences (e.g., teaching students when to say *a* book vs. *the* book). This tutor teaches students category knowledge, which encodes a variable stimulus (different English sentences) and constant response (particular English articles). Given this tutor, I used human data from the "IWT Self-Explanation Study 0 (pilot) (Fall 2008) (tutors only)" DataShop data set, which was collected as part of an experiment to test the effect of self-explanation on students' ability to select the correct English articles. For this experiment, 61 adult students, who were learning English as a second language, were randomly assigned to three versions of the tutor: one that simply tutored them on choosing the correct English articles, one that also asked them to self-explain the correct answer to each problem in a free-response



Directions: Choose the answer (a, an, the, or no article) that completes the sentence. If your answer is right, it will turn green. If it is wrong, it will turn red. Keep choosing until all the answers are correct, and if you need help, ask for a hint.

1. India ink is a special kind of ink.

Please **categorize** this image.

After each attempt you will be given correctness feedback.

Can we determine whether a water sample with 0.58 moles of arsenite (AsO2-) in 100 kiloliters (100kL) is safe? The World Health Organization's (WHO) recommended limit for arsenic in drinking water is 0.000014 grams of arsenite (AsO2) / L solution. So we need to calculate the grams of arsenite (AsO2-) / L solution that are in this sample, calculating the result to 2 significant figures. (Here is a helpful hint: the molecular weight of AsO2- is 106.9 g AsO2- / mol AsO2-)

$5y + 9 = -4$

Subtract 9 from both sides

$5y + 9 - 9 = -4 - 9$

Enter the left and right sides of the relation.

Figure 6.1: The interfaces for the six tutoring systems that collected the human data used in the current study in addition to data from the fraction arithmetic tutor from the previous chapter.

box, and one that instead asked them to choose one of multiple tutor-provided explanations for the correct answer to each problem. Wylie et al. (2009) found that all three versions of the tutor produced significant pre-posttest learning gains but found no difference in learning between versions that included self-explanation prompts and those that did not. For my simulations, I only used data from the 27 students assigned to the no self-explanation condition, because my computational models cannot currently respond to self-explanation prompts. Within the tutor, these students solved 84 article selection problems, yielding 3,696 steps.

The third system is the RumbleBlocks stability tutor, which teaches students to categorize the stability of towers of blocks produced in the RumbleBlocks game (Christel et al., 2012). Like the article selection tutor, it supports students' acquisition of category knowledge; e.g., multiple towers (variable stimulus) map to "stable" (constant response). The data for my simulations were collected from Mechanical Turk for the purpose of evaluating Trestle (MacLellan, Harpstead, Aleven, & Koedinger, 2016). A key characteristic of this tutor is that it did not tell the human students they were labeling the stability of towers, so they would be unable to bring their prior knowledge about stability to bear on the task. In particular, students labeled each tower as either "Category 1" or "Category 2", which, unknown to the students, corresponded to either "stable" or "not stable" (to avoid bias due to ordering, the particular category that mapped to "stable" was randomized for each student). Students were never told what the categories meant, but were provided correctness feedback on their categorizations. The 20 students in this data set each classified 30 towers producing a total of 600 steps. MacLellan, Harpstead, Aleven, and Koedinger (2016) reported that students' performance improves as they use the tutor.

The fourth tutor is for boxes and arrows problems, which each contains three numbers, two operators ($+$, $-$, $\times$ or $/$), and one arrow that points to the empty field (e.g., $3 + 2- > 7 - \_\_$). There are two types of problems, easy and hard problems, distinguished by the color of the arrow (my descriptions use $- >$ to denote easy problems and $=>$ to denote hard problems). In both cases, students try to enter the correct number in the empty field and are given correctness feedback. For the easy problems, the correct answer is computed by performing the arithmetic specified in the top boxes (i.e., the arithmetic to the left of the arrow in my example, $3 + 2$) and putting the result in the empty field. In contrast, for the hard problems, the correct answer is produced by entering a value in the empty box that makes the arithmetic in the bottom boxes ($7 - \_\_$) equal to the value in the top boxes that is on the opposite side as the empty box (3). For example, the correct answer to $3 + 2 => 7 - \_\_$ is 4 because $7 - 4 = 3$. To be clear, the rules for solving these problems are intentionally arbitrary and the primary challenge students face in this tutor is discovering the correct rules given the feedback.

For my simulations, I used data from the "Box and Arrow Tutor Data (Turk Study)" data set on DataShop. Lee et al. (2015) collected these data from Mechanical Turk as part of an experiment to investigate how different properties of boxes and arrows problems affect learning when no instructional support is provided. Students were randomly assigned to tutors that provided them either with constrained problems (where the correct rule produces a whole number and the incorrect rule does not) or unconstrained problems (where both the correct and incorrect rules produce whole numbers). Instruction within both tutors was blocked and students were randomly assigned to either receive all of the hard problems first or all of the easy problems first. A majority of the human students solved the easy problems correctly on the first attempt. In contrast, only a few students get their first attempt on the hard problems correct because they are specifically designed to make it difficult for students to use their prior knowledge. Thus, for my analysis, I trained the agents on all the problems, but only analyzed their performance on the hard problems, because my models do not take into account prior knowledge, which seems to plays a major role in their initial performance on easy problems.[2] This subset of the data contains 202 students solving 32 hard problems, for a total of 3,232 steps.

Lee et al. (2015) found that students in the constrained condition had better performance across both easy and hard problems. An analysis of this data shows that this effect holds just for the hard problems as well ($z = -4.601$, $p < 0.01$), via a mixed-effect regression analysis (Bates et al., 2015) with fixed effects for condition and practice opportunity, a random intercept and slope (practice opportunity) for each skill (there was a skill for each type of arithmetic operation—add, subtract, multiply, or divide—that students performed), and a random intercept for each student.[3]

Next is the stoichiometry tutor, which coaches students in how to convert values of one unit into values of another unit. This tutors students on unit, molecular, solution, and composition stoichiometry conversions. As part of these conversions, it also teaches students how to label the types of conversions they are performing, to properly cancel their units, and to round their answers to a specified number of significant digits. Thus, the tutor teaches students both categories (e.g., to label the type of conversion) and skills (e.g., to compute the answer). For my simulation study, I used the 'StoichStudy-WE-ErrEx-PS-TPS' data set from DataShop, which was collected as part of a 2×2 experiment that tested the effect of worked examples (tutored problems only or tutored problems plus worked examples) and personalized language (impersonal instruction or personal instruction) on learning within the stoichiometry tutor. McLaren et al. (2006) showed that all four versions of the tutor produced pre-posttest learning gains, but neither worked examples nor personalized instruction had any effect on these gains. I only simulated the subset of students in the control condition (tutored problems only with impersonal instruction), because simulating students in the other conditions would have required me to develop additional simulation capabilities (e.g., for intermixing examples with tutored practice). This subset contained 71 students solving 11 problems, producing a total of 79,581 steps. Problems in this subset were of six different types: unit conversion; unit and molecular conversion; molecular and composition conversion; unit, molecular, and composition conversion; two molecular and one composition conversion; and unit, composition, and solution concentration conversion. Problems were presented in an order that increased the number of necessary conversions (one conversion, then two conversions, then three conversions) and the types of conversions possible (unit, molecular, composition, and solution).

The last system is the equation solving tutor,[4] which guides students in solving two-step linear equations with a single variable. This tutor presents students with eight different types of problems that vary in which side the variable initially appears (left or right), which term the variable occurs in on a given side (first or second), and the types of operations that need to be performed to solve the problem (subtract then multiply or subtract then divide). When presented

[2]Half of the simulated agents received prior training on easy problems, which require different skills.

[3]The R lmer4 regression formula is correctness $\sim$ condition $+$ opportunity $+$ $(1$ $+$ opportunity$|$skill$)$ $+$ $(1|$student$)$.

[4]Here I will refer to it as the equation solving tutor for clarity, but it is actually a part of Carnegie Learning's Cognitive Tutor.

with a problem, students first choose a transformation to perform (e.g., *subtract* from both sides), then they choose an amount for that transformation (e.g., subtract *9* from both sides). Finally, they apply the transformation—updating both the left and right sides of the equation. This tutor teaches students a combination of both categories (e.g., picking the correct transformation to perform) and skills (e.g., performing the transformations). The human data for my simulations was downloaded from the "Self Explanation sch_a3329ee9 Winter 2008 (CL)" (Ritter et al., 2007) data set in DataShop, which contains data from 71 students solving 2501 unique problems, producing a total of 10,052 steps. One complication of this data set is that approximately half of the tasks (intermixed throughout their practice) are *strategy* problems, where the student only selects the transformations and the tutor performs them. For example, on one of these problems a student might specify the "subtract from both sides" transformation with the amount "9" and the tutor would automatically subtract 9 from both sides and update the left and right sides of the equation. During my simulations, the agents trained on all the problems, but they always performed all their own actions (the tutor did not perform any calculations for them). However, during my analysis, I only used the subset of the human data where they performed their own actions, which contained 4,568 steps.[5]

6.3 Simulation Approach

To test if the DECISION TREE and TRESTLE models support behavior prediction and theory testing across these seven tutors, I employed the simulation approach taken in the previous chapter. For every student I created an instance of both the DECISION TREE and the TRESTLE models[6] and connected these agents to machine-readable versions of the tutors used by the human students. These isomorphic tutors provided each agent with the same problems that the respective human students received, in the same order. Finally, I logged all of the transactions agents generated in these systems.[7]

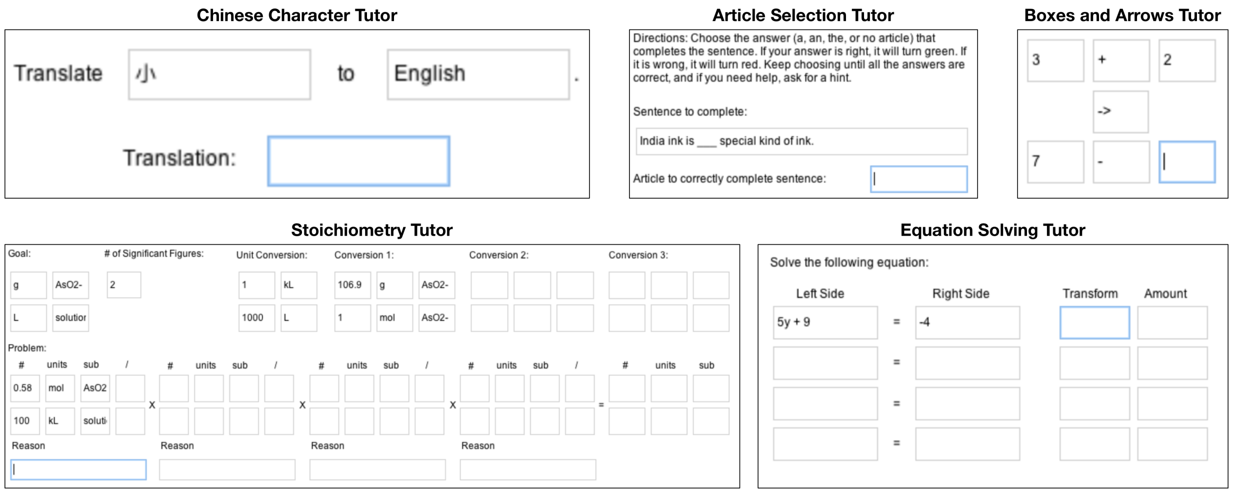
Tutors were needed to train the agents because they were not restricted to taking the same actions as their corresponding students (e.g., the agent might make different mistakes than the human or solve the problem using a different strategy). Additionally, the machine-readable tutors automatically generated relational representations of each step during problem solving. There was one notable exception for the RumbleBlocks stability tutor. Each step in this tutor was a single binary choice ("Category 1" or "Category 2") and the relational representation and correct category for each step were already available in the log data, so it was not necessary to author a new tutor to simulate learning. Thus, for the current simulation study, I had to author the five new tutors shown in Figure 6.2. I reused the fraction arithmetic tutor from the previous chapter.

I authored each of these tutors using the Cognitive Tutor Authoring Tools (CTAT) (Aleven, McLaren, Sewall, & Koedinger, 2006). I created the interfaces using CTAT's drag-and-drop interface builder and authored behavior graphs for each of the problems in the human data sets

[5]This subset *does* includes the steps where students choose the transformations on the strategy problems.

[6]The stoichiometry tutor had so many interface elements that TRESTLE ran very slowly, so I disabled structure mapping in order to improve runtime.

[7]All of the simulated data sets are available in DataShop in the "Apprentice Learner Architecture Simulated Students Datasets" project.



Translate 小 to English .

Directions: Choose the answer (a, an, the, or no article) that completes the sentence. If your answer is right, it will turn green. If it is wrong, it will turn red. Keep choosing until all the answers are correct, and if you need help, ask for a hint.

Sentence to complete:

India ink is ___ special kind of ink.

Article to correctly complete sentence:

3 + 2 -> 7 -

Goal: g AsO2- / L solutior; # of Significant Figures: 2; Unit Conversion: 1 kL / 1000 L; Conversion 1: 106.9 g AsO2- / 1 mol AsO2-

Problem: 0.58 mol AsO2 / 100 kL soluti x x x =

Solve the following equation:

5y + 9 = -4

Figure 6.2: Interfaces for the five new machine-readable tutors used to train simulated agents. The RumbleBlocks stability task did not require a tutor and I reused the machine-readable fraction arithmetic tutor from the previous chapter.

using Example-Tracing (Aleven et al., 2009). When authoring these interfaces and behavior graphs, I did my best to maintain a close alignment between the human and machine tutors, but this was not always possible and a number of discrepancies exist between the human and machine versions of the tutors. Some differences were due to a lack of access to the original tutors or screenshots of them. I often had to reverse engineer the tutors behavior using only the available log data and descriptions of the original tutors as a guide. This was the case for the Chinese character tutor, where I only guessed that the original interface looked similar to the one used in the MoFaCTS system, which was created by the same author (Pavlik et al., 2016) and is shown in Figure 6.1. In general, minor differences, like the additional descriptive text in the machine version of the Chinese character tutor, should have no impact on the simulated agents' learning or performance.

However, some differences exist that could affect learning. For example, in the machine version of the article selection tutor, agents do not have access to menu options, which contain a list of possible English articles from which they could choose. Instead, they only find out about possible English articles once the tutor has demonstrated them. Additionally, the tutor for humans presents the drop-down menu in the training sentence, whereas the machine tutor has a separate text field outside the sentence. This latter difference impacts the simulated agents' ability to parse the sentence; specifically, agents have relational knowledge for parsing an English sentence presented as a single string and could not parse the training sentence if it was broken into separate strings (to the left and right of the drop-down menu). Future work might extend agents to cover these relatively minor differences, so I did not spend the additional development time needed to perfectly align these tutor experiences.

Perhaps more troublesome, though, are the differences between the original and machine-readable versions of the stoichiometry and equation solving tutors. In stoichiometry, the agents do not have prior knowledge sufficient for parsing key information out of textual directions, such as the molecular weight of a particular compound. To overcome this limitation, I replaced the textual problem descriptions with interface elements that contain information that humans can easily extract from the text. Additionally, humans had access to tables containing unit conversion information outside of the tutor, so I included tutor fields that provided this relevant information. Human students were also directly instructed to use the Microsoft Windows calculator in order to perform the actual calculations (e.g., multiplying all numerators and dividing by all denominators). In contrast, the simulated agents received no such instruction and did not have access to a calculator. The result is that many simulated agents struggled to explain the final answer because the correct explanation required them to perform how-search beyond their current depth limit of two; e.g., multiplying four numerators and dividing by four denominators would require a depth limit of three.[8] In contrast, humans are taught (in an instructional video) a procedure for entering numerators and denominators into their calculator (outside the tutor) and copying the result into the tutor interface.

In equation solving, there are also many meaningful differences between the original and machine-readable versions of the tutor. First, the human tutor was highly dynamic—new fields would appear in response to students' actions. Additionally, some of the steps were performed by the tutor, rather than the student. Finally, the students were allowed to take legal algebraic actions that were off-path (they got negative feedback, but the tutor let them continue). These factors made building an equivalent machine version difficult. Ultimately, I modified a version of the SIMSTUDENT algebra tutor interface (MacLellan et al., 2014; Matsuda et al., 2014) to include separate fields for the transformation and amount.[9] This final machine tutor presented all fields at once (i.e., fields were not hidden and then later revealed), made agents perform all the steps (no tutor performed steps), and did not let students take off-path actions. In general, I found minimizing the difference between the simulated and human students' experiences to be one of the biggest challenges of my thesis work. As my models are premised on the assumption that variation in human behavior is due primarily to differences in their prior knowledge and experiences, good alignment is essential. As I show in the next section, the agreement between the human and simulated agents is worse in the domains where more differences exist and where students could use more of their prior knowledge.

6.4 Model Evaluation

6.4.1 Overall and Asymptotic Performance

The data generated from these simulations affords the same analyses taken in the previous two chapters. In order to demonstrate the generality of these models, I attempted to replicate the previous findings across all seven tutor domains. In particular, that apprentice learner models support tutor development, behavior prediction, and theory testing. To assess the tutor development capabilities of these models across the seven tutor domains, I first looked at the overall and asymptotic performance of the two models. Additionally, I computed the overall and asymptotic

[8]Using the current how-search approach, all operators are applied at each depth in parallel, so a depth of three, not eight, is required.

[9]The SIMSTUDENT interface has agents enter both the transformation and amount as a single string, but the human interface has students enter them separately.

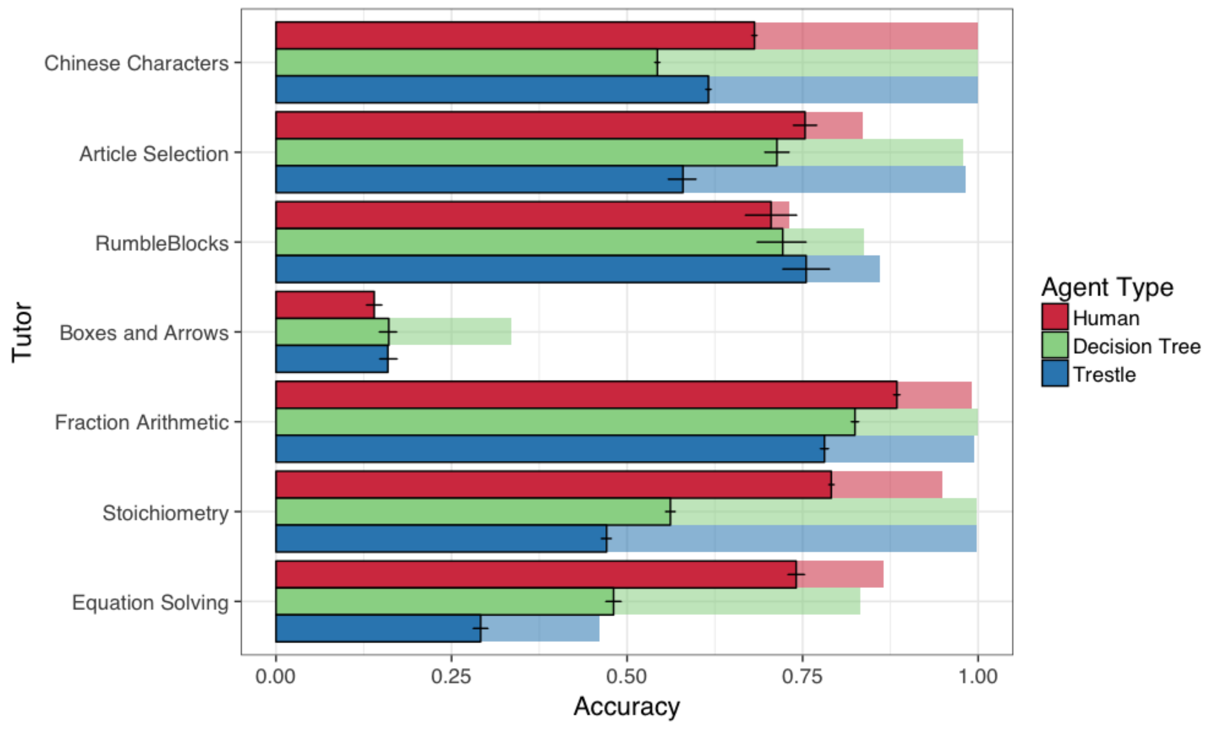


Figure 6.3: The overall (opaque) and asymptotic (semi-transparent) accuracy for each type of agent in each tutor. The 95% confidence intervals are shown for the overall accuracy. The asymptotic accuracy was computed by fitting a mixed-effect regression model to each data set and using it to predict the performance on the practice opportunity where at least 95% of the data had been observed. This approach gives an estimate of the accuracy achieved in each tutor by the end of training (over all students and skills).

performance of humans in each of these tutors as a baseline for evaluating the performance of the two models. These results are shown in Figure 6.3.

To compute asymptotic performance, I labeled each step in both the human and simulated data sets by the skill or knowledge component (Koedinger et al., 2012) it exercised. Then, for each first attempt (the first time performing a particular step on a problem), I computed the prior number of opportunities a student had to exercise the same skill. Using these values, I fit a linear mixed-effects binomial regression model to each data set (Bates et al., 2015), which had a single fixed effect for practice opportunity, a random intercept and slope (practice opportunity) for each skill, and a random intercept for each student.[10] Then, for each data set, I used the fixed-effects estimates for the intercept and opportunity to predict performance for the average student and skill on the opportunity where at least 95% of the data for the respective data set had been observed. These asymptotic accuracies, which are plotted in Figure 6.3 as semi-transparent bars, represent the performance for the average student and skill at the end of practice in each tutor. It is worth noting that I chose a threshold of 95% because some skills were practiced by some

[10]This model is similar to a repeated measures ANOVA and analogous to the Additive Factors Model (Cen, 2009). It is represented using the R lme4 formula correctness $\sim$ opportunity $+ (1 +$ opportunity$|$skill$) + (1|$student$)$.

students much more than others. A threshold of 100% would equate to evaluating the accuracy for all skills at the maximum practice opportunity observed across all students and skills. In contrast, a threshold of 95% covers most of the students and skills, but is more flexible with respect to ignoring students and skills with an unusually high number of practice opportunities.

In terms of overall performance across all tutor steps, humans generally have higher scores than the DECISION TREE model, which generally has higher performance than the TRESTLE model. However, some exceptions exist. For example, TRESTLE outperforms the DECISION TREE model in the Chinese character tutor, and both models have slightly higher performance than humans on RumbleBlocks and boxes and arrows. It is worth noting that these latter tutors were the ones specifically designed to minimize the effect of students' prior knowledge, putting humans on an equal playing field with the models. The stoichiometry and equation solving tutors show the biggest differences in performance between agents and humans, likely because these tutors have the biggest differences between the human and machine versions of the tutors. Additionally, human students probably bring more prior knowledge in these tutors. For example, humans had already completed a unit on one-step linear equations before using the equation solving tutor. In general, these results extend previous work with SIMSTUDENT (Li, 2013), which showed that domain-specific models can learn to solve easier problems in article selection, fraction arithmetic, equation solving.[11]

The situation is different for the asymptotic accuracies. For Chinese characters, article selection, fraction arithmetic, and stoichiometry, both the DECISION TREE and TRESTLE models seem to have learned the target skills by the end of training. However, of these four domains, humans only seem to have mastered two—Chinese character and fraction arithmetic. For RumbleBlocks, I find that the TRESTLE model does best—likely because Trestle leverages the relational structure for this task. For boxes and arrows, the DECISION TREE model outperforms both humans and the TRESTLE model. Apparently the DECISION TREE model's non-incremental approach to when-learning performs better on this task. In contrast, TRESTLE produces asymptotic behavior that is more similar to the humans. Finally, in equation solving, the TRESTLE model performs worse asymptotically than both the DECISION TREE model and the humans. This lower performance may have something to do with the way that Trestle leverages the relational structure of parse tree features for equations. In this case, the relations might be leading it astray. In general, these results support a domain-general version of the claim that these apprentice learner models can support tutor authoring. In particular, they show that even though the simulated agents perform worse than humans overall, by the end of training they are able to achieve human-level performance or better in all but the equation solving tutor.

6.4.2 Instructional Effect Prediction

Given these overall results, I next looked at testing both models' capabilities for predicting the effects of instructional manipulations. In the previous chapter, I showed that both models predict the effect of problem ordering on tutor and posttest performance in fraction arithmetic. Of the new data sets, the only testable instructional manipulation was in boxes and arrows, which varied whether students received constrained or unconstrained training problems. On constrained prob-

[11]These prior models were trained with an easier subset of the problems from the current simulations.

lems, the correct rule produces an integer value, but the incorrect rule produces a non-integer value. For the unconstrained problems, however, both correct and incorrect rules produce integer values. Lee et al. (2015) found that students learn better when presented with constrained problems across both the easy and hard problems. To test if my models predict this effect, I first tested to ensure this effect holds with the humans for just the hard problems as I did not analyze performance on the easy problems. To test for this effect, I used a mixed-effect binomial regression analysis (Bates et al., 2015) with fixed effects for condition (constrained or unconstrained) and practice opportunity, a random intercept and slope (for practice opportunity) for each skill, and a random intercept for each student.[12] This analysis, which is similar to the repeated-measures ANOVA used by Lee et al. (2015), shows that the effect of condition holds just for the hard problems as well ($z = -4.601$, $p < 0.01$). Next, I applied the same statistical test to both of the simulated data sets, which shows that the TRESTLE model successfully predicts this effect $z = -8.278, p < 0.01$, but the DECISION TREE model does not $z = 1.098, p > 0.27$. Although the earlier results suggest that the DECISION TREE model has better performance, this result suggest that the TRESTLE model better predicts human behavior. Interestingly, Lee et al. (2015) hypothesize that students do better with constrained problems because the correct rule is easier to compute. However, correct and incorrect rules are equally easy for both models, suggesting that it is not the difficulty of computation that is driving these results. Instead, my findings suggest that problem ordering and interference, due to different distractor numbers and operators in the unconstrained problems, are responsible. The DECISION TREE model is less affected by problem ordering and interference, so it does not show this effect.

6.4.3 Learning Curve Prediction

Next, I generated learning curves to replicate the behavior prediction results from the previous chapter across all seven domains. Similar to the learning curves from chapter 5, the left graph of Figure 6.4 shows the average error on the first attempts at each step given the practice opportunity for both the humans and the two models. Note, these learning curves averages over all tutors, students, skills, and steps (for the individual learning curves for each tutor see appendix A). Although the overall performance results (Figure 6.3) sometimes suggest that the DECISION TREE model has performance closer to humans, these learning curves show this is an artifact of averaging initially worse performance with subsequently better performance. In contrast, the TRESTLE model starts off worse (like the other model), but converges to the human performance, rather than exceeding it. Further, the learning curves suggest that TRESTLE actually performs slightly better than the DECISION TREE model initially, even though its when-learning approach is non-incremental. What is striking is that the TRESTLE model appears to predict human performance quantitatively at higher practice opportunities. These findings generally support the claim that it better accounts for human behavior than the DECISION TREE model.

To verify that these findings are not an artifact of averaging over multiple tutors, students, and skills, I also plotted residuals of both models for each opportunity, see the right graph of Figure 6.4. I generated these residuals by subtracting each model's prediction, for a particular

[12]The R lmer4 regression formula is correctness $\sim$ condition $+$ opportunity $+$ (1 $+$ opportunity$|$skill) $+$ (1$|$student).

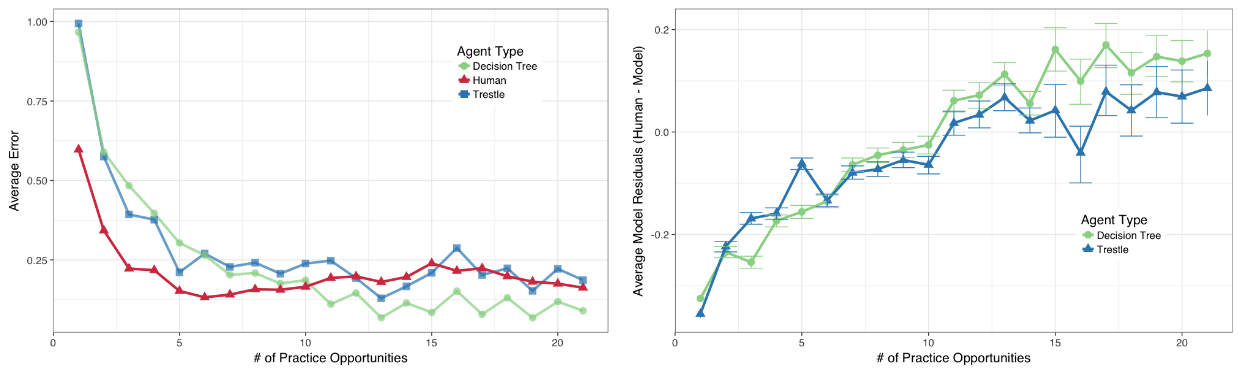


Figure 6.4: Learning curves for humans and the two models across seven tutoring systems (left) and the model residuals plotted by practice opportunity (right). Model residuals are computed by subtracting the model prediction from the actual human performance for a particular student on a particular opportunity of a particular skill. For model residuals, the 95% confidence intervals are also shown.

student on a particular opportunity of a particular skill from the corresponding human student's performance. These residual curves agree with the learning curves and show that, in general, the TRESTLE residuals are closer to zero than the DECISION TREE residuals, suggesting that it better predicts human behavior on specific steps.

6.4.4 Theory Testing

To replicate the prior theory testing results across the seven tutor domains, I fit a linear mixed-effects binomial regressions to the simulated data from each model (Bates et al., 2015), to test which one better better accounts for human behavior. Like my previous analysis of the fraction arithmetic data, each regression has two fixed effects, one for the respective model prediction and one for the number of prior practice opportunities, a random intercept and slope (for prior practice opportunities) for each skill, and a random intercept for each student. However, this analysis also included random intercepts for each tutor to account for the correlation of predictions within the same tutor.[13] This analysis shows that the TRESTLE model (AIC $= 85255$, BIC $= 85331$) better fits the human data than the DECISION TREE model (AIC $= 85638$, BIC $= 85714$). A likelihood ratio test also reveals that the fit of the TRESTLE model is significantly better ($\chi^2(0) = 383.57$, $p < 0.01$). These results extend the findings from the fraction arithmetic tutor, showing that the TRESTLE model has better agreement with the human data across all seven domains. This finding provides more evidence for the theory that humans learn the conditions for skill application incrementally, rather than non-incrementally.

[13]The corresponding R lme4 regression formula is: human.performance $\sim$ model.prediction $+$ opportunity + (1 $+$ opportunity $|$ skill) $+$ (1$|$student) $+$ (1$|$tutor).

6.4.5 Expert-Model Authoring Efficiency

Having demonstrated the use of the models for tutor authoring, behavior prediction, and theory testing. I next analyzed the simulation data to investigate the efficiency of the two models for the purposes of authoring (similar to the efficiency analysis in chapter 4). These data contain counts of the number of examples and feedback that the tutors provided to each agent. By treating each tutoring systems as a stand-in for a human author, I was able to compute the number of demonstration and feedback interactions necessary to train each simulated agents and evaluate the time it would take an average trained developer to train the agents. Additionally, each of the machine-readable tutoring systems was built using Example-Tracing, so it was straightforward to evaluate the efficiency of this alternative approach for comparison purposes.

When analyzing the efficiency of these authoring techniques, I used the approach from chapter 4. In particular, I used the Keystroke-Level Model (KLM) from my previous work (MacLellan et al., 2014) to estimate the time it would take a trained, error-free, author to build expert models for the seven tutors using either simulated students or Example-Tracing. This estimates that demonstrating an action in the tutor interface takes approximately 8.8 seconds using either the simulated student or Example-Tracing approach.[14] The model also estimates that specifying a group of unordered actions (when Example-Tracing) takes 5.8 seconds and providing feedback (when training simulated agents) takes 2.4 seconds. Finally, I assume that mass production takes *zero* seconds.

When building the seven tutors using Example-Tracing, I authored 586 behavior graphs for the Chinese character tutor (one mass production template containing two demonstrations links), 84 graphs for the article selection tutor (one template containing two demo links), 64 graphs for the boxes and arrows tutor (two templates, each containing two demo links), 84 graphs for fraction arithmetic tutor (three templates containing 22 demo links), 16 graphs for the stoichiometry tutor (eight templates containing 897 demo links and 30 unordered action groups), and 1357 graphs for the equation solving tutor (six templates containing 54 demo links and 24 unordered action groups). I did not create a new RumbleBlocks tutor, but for the current analysis, I estimated that it would require 139 behavior graphs (two templates containing two demo links). I multiplied these counts by the appropriate estimate from the KLM, to estimate how long it would take the average trained author to build the behavior graph templates for each tutor. The results of these calculations are shown in Figure 6.5.

To estimate the time needed for each of the simulated student approaches, I tabulated the total amount of examples and feedback that were provided to all of the simulated students across all of the simulations. Then, using the KLM, I estimated the total amount of time needed to train each agent. Finally, I averaged these estimates to generate the results shown in Figure 6.5.

In contrast to the previous results in experimental design, this analysis shows that for all domains except stoichiometry, it is substantially more efficient to author the expert model using Example-Tracing. This unexpected finding suggests that one approach is not always better than the other, as chapter 4 and previous work (MacLellan et al., 2014) would suggest. A closer inspection of stoichiometry and the other domains shows that each of the behavior graphs in this

[14]The original model predicted 10.4 seconds for SIMSTUDENT demonstrations because it required authors to provide foci of attention. However, demonstrating with the the current models does not require these foci, so it is identical to demonstrating in Example-Tracing.

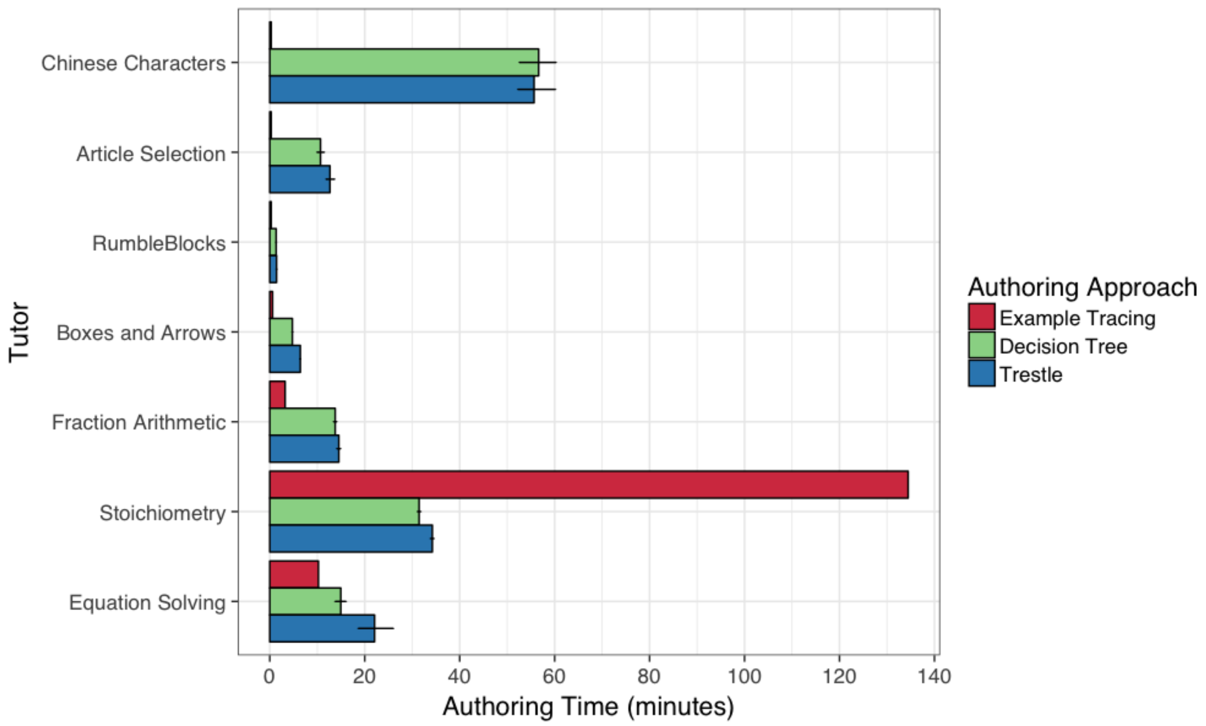


Figure 6.5: The estimated amount of time (in minutes) it would take the average trained author to build an expert model for each of the seven tutors using either Example-Tracing or one of the two simulated student models. Both simulated student approaches are shown with 95% confidence intervals. These estimates were generated by tabulating the number of authoring actions required by each approach and converting these counts into an overall time estimate using a keystroke-level model (MacLellan et al., 2014).

tutor are complex, requiring hundreds of demonstrations, and most problems had different behavior graph structure, so they required separate mass production templates. These characteristics make this tutor similar to the experimental design tutor, which I have already shown is more efficient to author using the simulated student models. These results suggest that for tutors that have simple behavior graph structure (i.e., has a small number of solutions and solution paths) and that have problems with identical behavior graph structure (i.e., benefit from mass production), then it is more efficient to author the expert model using Example-Tracing. However, when authoring tutors with more complex behavior graph structures and that require multiple mass production templates, authoring with simulated students is more efficient.

It is also possible that these results are an artifact of the assumptions adopted in the efficiency analysis. In particular, the KLM assumes that authors are error free in their authoring, which is very optimistic. When authoring the behavior graphs for my simulations, I made many mistakes and it took me substantially longer to author these tutors. For example, authoring the behavior graphs for the article selection tutor alone took me at least three full eight-hour days of authoring, and this was after a week of planning the appropriate behavior graph structure. When authoring more complicated graphs, I often made mistakes that would not be apparent until the end, and

I would have to re-author the entire graph.[15] In contrast, mistakes will decrease the simulated agents' performance, but they should be able to recover from these mistakes given more training (i.e., the author does not have to retrain the agent from the beginning).

The KLM also fails to take into account preparation and planning time. In particular, I found it challenging to author behavior graph templates (with variables rather than specific values) directly. Typically, before authoring a template, I would author a complete behavior graph for one or two specific problems, and then think about how I could author a template that would generalize both problems. Ignoring the time spent planning, if I added the demonstrations needed for these preliminary behavior graphs to my estimates, it would double or triple the authoring time. In contrast, training the simulated agents only required me to provide demonstrations and feedback on specific problems, which seemed to require less preparation and planning.

Additionally, the authoring time results show that expert models with a lot of unique problem content, such as hundreds of Chinese characters and their translations, take substantially longer to build with the simulated agent approaches. This is likely due to the assumption that mass production takes zero seconds, an unrealistic assumption. I aimed to show that, even in the best case, for tutors requiring complicated graph structure like stoichiometry and experimental design, authoring with simulated agent models is still more efficient than Example-Tracing with mass production. However, this assumption has a side effect of hiding how long mass production actually takes. At the very least, it requires the author to input content for each problem into an Excel spreadsheet, which I suspect would be comparable to demonstrating the specific problem content to the simulated agents.

Finally, one key challenge with the simulated agents is determining if they have acquired correct and complete expert models. To overcome this challenge in practice, authors could track a simulated agent's performance during training and stop training only once it has achieved an acceptable level of performance. Applying a similar idea to the simulation data, the asymptotic accuracy results (see Figure 6.3) suggest that agents in the Chinese characters, article selection, fraction arithmetic, and stoichiometry tutors have mostly converged to 100% correct. However, in the RumbleBlocks, boxes and arrows, and equation solving tutors, agents achieve lower asymptotic accuracies, suggesting they might need more training. The towers in the RumbleBlocks tutor are non-deterministically labeled using a free-body physics simulator, so the model will never be able to achieve 100% accuracy. In boxes and arrows, agents only received four examples for each skill, and clearly more training is necessary to learn the correct skills. Similarly, the equation solving tutor had not converged and would require more training before one could consider it finalized.

Overall, these results provide the most rigorous evaluation of authoring tutors with simulated agents to date, replicating the findings of the previous two chapters across seven tutor domains. They provide insight into when authoring with simulated agents is preferable to authoring with Example-Tracing, such as when multiple complex behavior graphs are required. Moreover, they show that the DECISION TREE and TRESTLE models can successfully learn skills across the seven tutor domains, even though they draw on a fixed set of domain-general prior knowledge.

[15]If the errors were minor, then it might be possible to edit the graph rather than re-creating it. However, for more complex graphs, it was often difficult to verify that all errors had been corrected and it was typically easier to re-create the graphs correctly than to try to edit them

6.5 Discussion and Conclusions

The results of this chapter further support the claims I make in the previous two chapters. First, they indicate that building expert models by training simulated agents is viable even when domain-specific knowledge is not available. In this case, both the TRESTLE and DECISION TREE approaches can efficiently learn expert models for the Chinese character, RumbleBlocks, article selection, fraction arithmetic, and stoichiometry tutors as determined by asymptotic accuracy. Further, these approaches appear to be effective in the other tutors (boxes and arrows and equation solving), but more training would be necessary for them to learn correct and complete expert models. In either case, the results demonstrate that domain-specific knowledge is not necessary for successful tutor authoring.

For the second authoring claim, that training simulated agents is an efficient authoring approach comparable to Example-Tracing with mass production, the results were generally positive. However, it appears that neither approach strictly dominates the other. For tutors that require only a few solutions and paths per problem, the Example-Tracing approach appears to be more efficient. In contrast, for tutors that require multiple complex behavior graphs with many solutions and paths per problem, such as the experimental design and stoichiometry tutors, authoring with simulated agents appears to yield better efficiency.

The results from this chapter extend the behavior prediction and theory testing results from chapter 5. More specifically, they show that the TRESTLE model predicts the effect of problem type (constrained or unconstrained) on performance in the boxes and arrows tutor. In contrast, the DECISION TREE model outperform humans in this tutor and did not predict this effect. Additionally, the chapter shows that the models can predict students' learning curves across multiple domains. Both models start initially worse than the humans, but the TRESTLE model converges to human performance, whereas the DECISION TREE model exceeds humans. These results provide evidence that TRESTLE better approximates humans, with a regression analysis showing that the TRESTLE model better explains human behavior across all seven tutors. Collectively, these findings support the theory that humans learn when to apply skills incrementally, rather than non-incrementally, as the SIMSTUDENT model of learning assumes.

In conclusion, this chapter demonstrates the general capabilities of my models. Even though they draw on a small, fixed set of domain-general prior knowledge, they can learn and perform across seven tutoring systems that vary in types of knowledge (associations, categories, and skills) and domain content (language, math, engineering, and science). The findings from this chapter support my overarching claim that apprentice learner models are general-purpose tools capable of supporting the design, building, and testing phases of tutor development. Additionally, this work constitutes a proof of concept for a research program that generates and tests computational models of learning.

Chapter 7

Conclusions and Future Work

In this thesis, I explore the use of apprentice learning models, or computer models that learn from examples and feedback, for supporting tutor development, behavior prediction, and theory testing. To support these investigations, I first developed a computational theory of apprentice learning. In particular, chapter 2 defines the apprentice learning problem and reviews prior work that has attempted to solve it and chapter 3 presents the Apprentice Learner Architecture, a theoretical framework that unifies the theories and mechanisms from prior work and supports the construction of alternative learner models. Additionally, chapter 3 describes two novel models of apprentice learning, the DECISION TREE and TRESTLE models. Given these models, I next turned to exploring their applications. Chapter 4 analyzes the DECISION TREE model's ability to author an expert model for experimental design and chapter 5 explores the use of both models for predicting student behavior in a fraction arithmetic tutor. Additionally, chapter 5 demonstrates the use of students' tutor data for evaluating which of the two models better explain human behavior. Finally, chapter 6 highlights the generality of these models by replicating these capabilities across seven tutors that teach multiple kinds of knowledge across a wide range of domains.

Across this work, I have endeavoured to convince the reader of four main claims. In particular, that apprentice learner models:

- Support efficient expert-model authoring;
- Predict student behavior;
- Facilitate theory testing; and,
- Are domain-general tools.

Additionally, I have tried to provide evidence for the meta-level claim that the Apprentice Learner Architecture unifies prior learning models and supports model development. This chapter reviews each claim and the meta-level claim, the evidence to support them, and discusses limitations and directions for future work related to each claim.

Support Efficient Expert-Model Authoring

First, I set out to show that apprentice learning models can support efficient authoring of tutor expert models, even when they lack domain-specific prior knowledge. To support the efficiency aspect of this claim, chapter 2 shows that authoring an experimental design tutor using the DECISION TREE model takes about one third of time it takes to author an equivalent tutor using Example-Tracing, even when assuming that mass production (a technique for generalizing Example-Tracing content to new problems) takes zero time to use. A similar analysis in chapter 6, however, suggest that one approach is not always better than the other. In particular, it shows that Example-Tracing is more efficient for six out of the seven tutor domains analyzed (stoichiometry was more efficient with the DECISION TREE model). A closer analysis of the differences between the stoichiometry and experimental design tutors and the other tutors suggests that authoring with simulated agents is preferable for tutors with complex problem spaces (i.e., those with many possible solutions and many alternative paths to each solution). In contrast, for simpler domains, Example-Tracing appears to be a more efficient approach. However, the analysis leading to these conclusions assumes that mass production takes zero additional time, which is an unrealistic assumption (in favor of Example-Tracing). Thus, future work should explore how to incorporate the cost of mass producing content into the authoring time evaluation. Additionally, the current evaluation ignores Example-Tracing's formula editing capabilities, which lets developers use formulas to reduce the number of behavior graph demonstrations needed. Although one might argue that use of this capability requires specialized authoring expertise, future work should explore how authoring with simulated agents compares to Example-Tracing when developers can also use this capability.

To support the claim that these models can support authoring even when they lack domain-specific prior knowledge, chapter 2 shows that the DECISION TREE model is able to learn an expert model for experimental design, even though it does not utilize any prior knowledge specific to this domain. Further, chapter 6 shows both models can be applied to seven different tutors (achieving human-level performance in most domains) despite the models lacking prior knowledge specific to many of the domains (e.g., Chinese characters and RumbleBlocks). These results suggest an affirmative to my claim. Despite these positive findings, prior knowledge is still useful for improving learning efficiency and future work should explore the discovery and use of other relational knowledge and overly-general operators that can enhance agents' learning, such as relational knowledge for text-adjacency and overly-general recursive and successor operators.

Predict Student Behavior

My second high-level claim is that apprentice learning models support the testing of alternative tutor designs and can predict which designs will be more effective for learning. To support this claim, I first showed that both the TRESTLE and DECISION TREE models are able to successfully predict which problem orderings (blocked or interleaved) will be yield better tutor and posttest performance in a fraction arithmetic tutor (chapter 5). Additionally, I showed that the TRESTLE model is able to successfully predict that students trained with constrained problems in the boxed and arrows tutor will have better performance than those trained with unconstrained problems (chapter 6). Finally, I showed that both models are able to generate reasonable parameter-free learning curve predictions for seven different tutors (chapter 6). To my knowledge, these results provide the first examples of the use of apprentice learner models for parameter-free prediction of learning behavior. However, one key limitation of the current models is that they do not take into account students' relevant prior experience or knowledge, which may results in systematic discrepancies between human and simulated behavior; e.g., simulated agents always have 100% error on their first opportunity, whereas humans do not. Thus, future work should explore how to initialize prior knowledge for each model. One straightforward possibility is to explore the extent to which student individual differences at the first opportunity can be simply modeled as differences in the number prior unobserved practice opportunities they have experienced.

Facilitate Theory Testing

My third high-level claim is that researchers can use human data, collected by tutoring systems, to evaluate apprentice learner models and guide a search for models and theories that better align with human behavior. Chapter 5 provides the initial evidence to support this claim by showing the use of regression analysis to determine which of the two models better accounts for variation in the human data (the TRESTLE model). Additionally, this chapter shows that the simulated learning curves for the TRESTLE model converge to human performance, whereas those for the DECISION TREE model exceed human performance. Chapter 6 verifies that these findings hold across seven tutor domains and also shows that the TRESTLE model can predict the results of an instructional manipulation that the DECISION TREE model cannot (in boxes and arrows).

These results indicate that student data can distinguish between models that instantiate alternative theories of learning, and in my case, that humans appear to learn when skills apply incrementally (forming generalizations along the way) rather than in batch (storing prior examples and recomputing generalizations). This finding supports my high-level claim and suggests real promise for the general program of research wherein theory is advanced by testing alternatives across multiple human learning data sets. Future work should apply this approach to test other theories of human learning, such as whether where- and when-learning are distinct mechanisms.

Are Domain-General Tools

My fourth high-level claim is that these models are domain general and can model learning of multiple knowledge types across a wide range of domains, even though they draw on a small, fixed set of prior knowledge. Domain generality applies to all three claims above (that these models support tutor development, behavior prediction, and theory testing), but is distinctive in that many efforts to model learning (e.g., Ur & VanLehn, 1995; VanLehn et al., 1991) are evaluated using one or a few domains. Chapter 6 provides the main evidence to support this claim by showing that the two models support authoring, prediction, and theory testing across seven different tutors. Moreover, these tutors teach a wide range of knowledge types and cover multiple content domains, suggesting that the true generality of the models is greater than explicitly demonstrated. One limitation of the current work is that it focuses on modeling learning in tutoring systems, rather than educational technologies more broadly. Future work should explore how to extend the current models beyond tutors to other learning environments, such as educational games (see Harpstead, 2017) and programming environments.

Unify Prior Models and Support Model Development

Collectively, this work supports my final, meta-level claim that the Apprentice Learner Architecture unifies prior learning theory and mechanisms and supports model development. Prior cognitive architectures facilitate cross-domain models of *reasoning and problem solving* while committing to a particular set of learning mechanisms (ICARUS, ACT-R, and Soar), whereas the Apprentice Learner Architecture is arguably more flexible with respect to learning mechanisms and facilitates cross-domain models of *learning*. The primary evidence to support this claim is that the architecture supported the development of two novel models of apprentice learning. One of the models (DECISION TREE) combines components from prior apprentice learning models in novel ways (a STEP-like how-learning approach and a SIMSTUDENT-like when-learning approach). In contrast, the other model (TRESTLE) uses a novel when-learning approach not previously explored in the apprentice learning literature. Analogous to how Mendeleev's periodic table of elements supports the organization and discovery of chemical elements, the Apprentice Learner Architecture supports the organization and discovery of learning models. A major limitation of the current work, however, is its emphasis on learning in tasks where problem-solving subgoals can be retrieved from the external display of progress. Towards the development of a unified theory of learning *and* problem solving, future work should explore the integration of the current architecture with existing cognitive architectures that emphasize reasoning and problem solving. Additionally, the current architecture should be extended to support domains where multiple alternative solution paths and/or subgoals must be mentally maintained.

In conclusion, this thesis makes contributions to the fields of human-computer interaction, educational data mining, artificial intelligence, psychology, and the learning sciences. It explores how early work on the Model Human Processor (Card et al., 1986), which aimed to encapsulate findings from psychology in a format that could be utilized by interface designers, might be extended to learning; in particular, it demonstrates progress towards a Model Human Learner that encapsulates learning science findings in a format that can be utilized by instructional designers. To fulfill this vision, I developed a framework for integrating existing artificial intelligence and machine learning approaches (the Apprentice Learner Architecture) and used it to develop two new models of learning (DECISION TREE and TRESTLE). Finally, I demonstrated the applications of these models for tutor authoring, behavior prediction, and theory testing, and introduced a general research program that can be used to test and improve theoretical models of human learning. This work lays the foundation for the ultimate discovery of a Model Human Learner that can support tutor developers and explain available educational data.

Appendix A

Supplementary Learning Curves

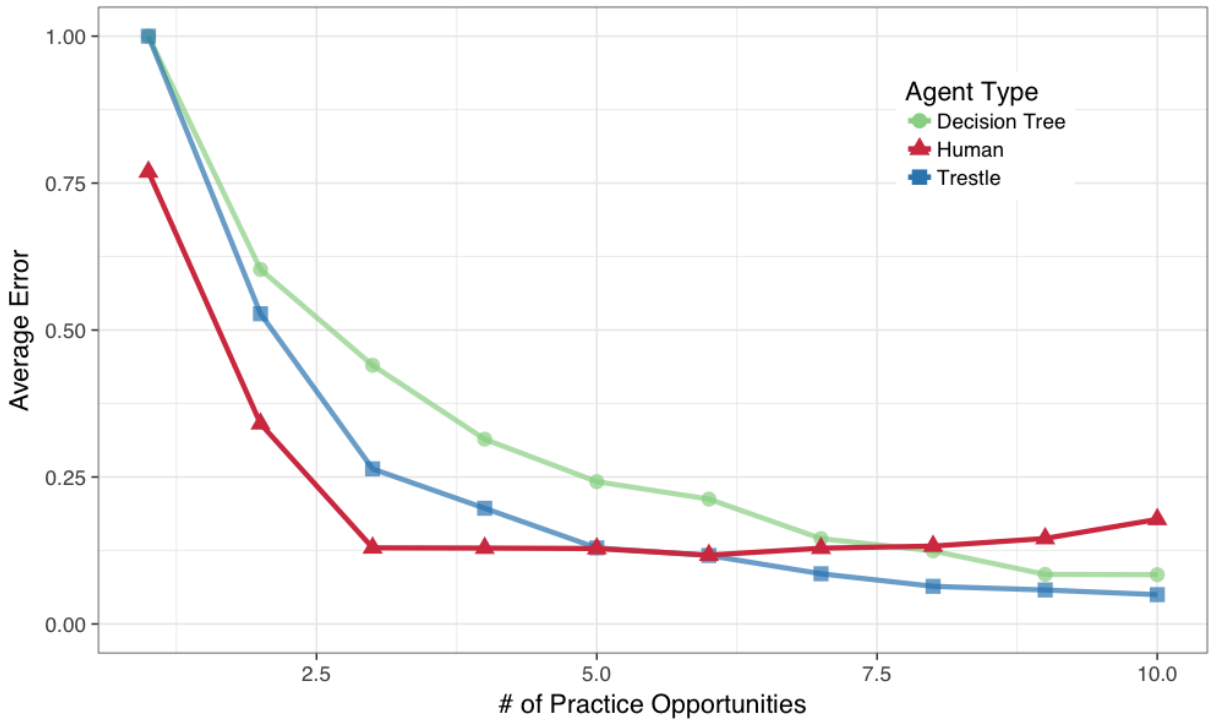


Figure A.1: Overall learning curves for the humans and the two models in the Chinese character tutor.

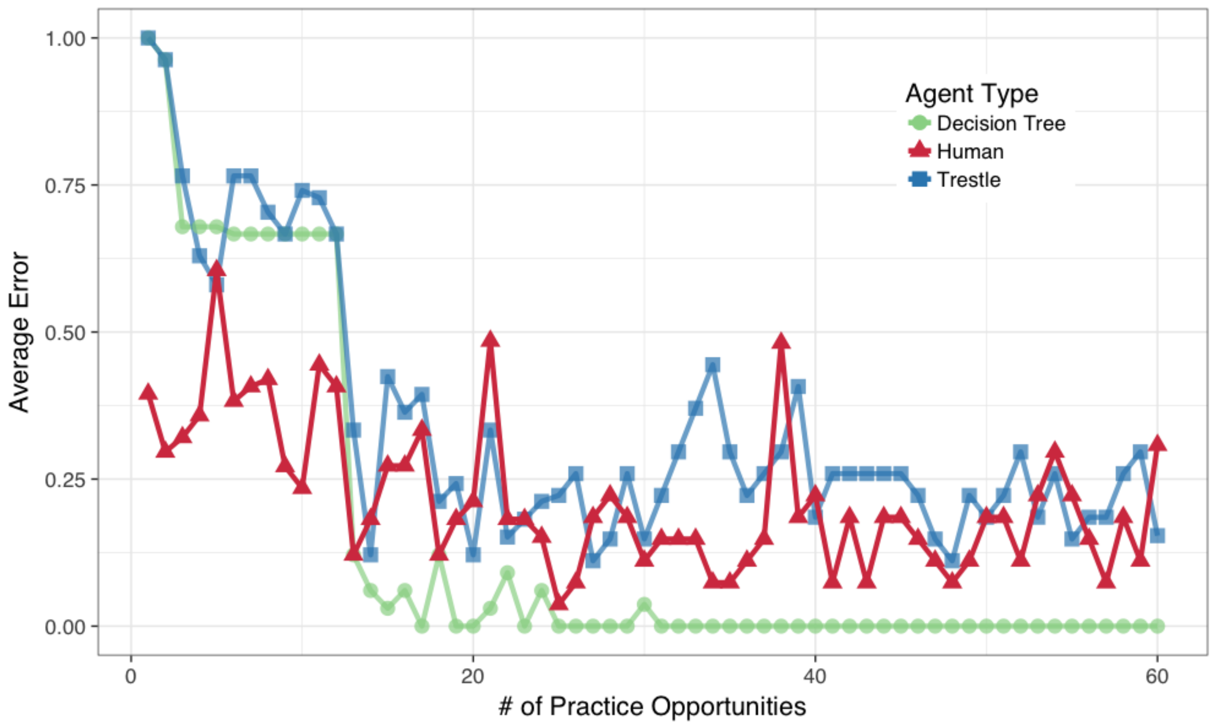


Figure A.2: Overall learning curves for the humans and the two models in the article selection tutor.

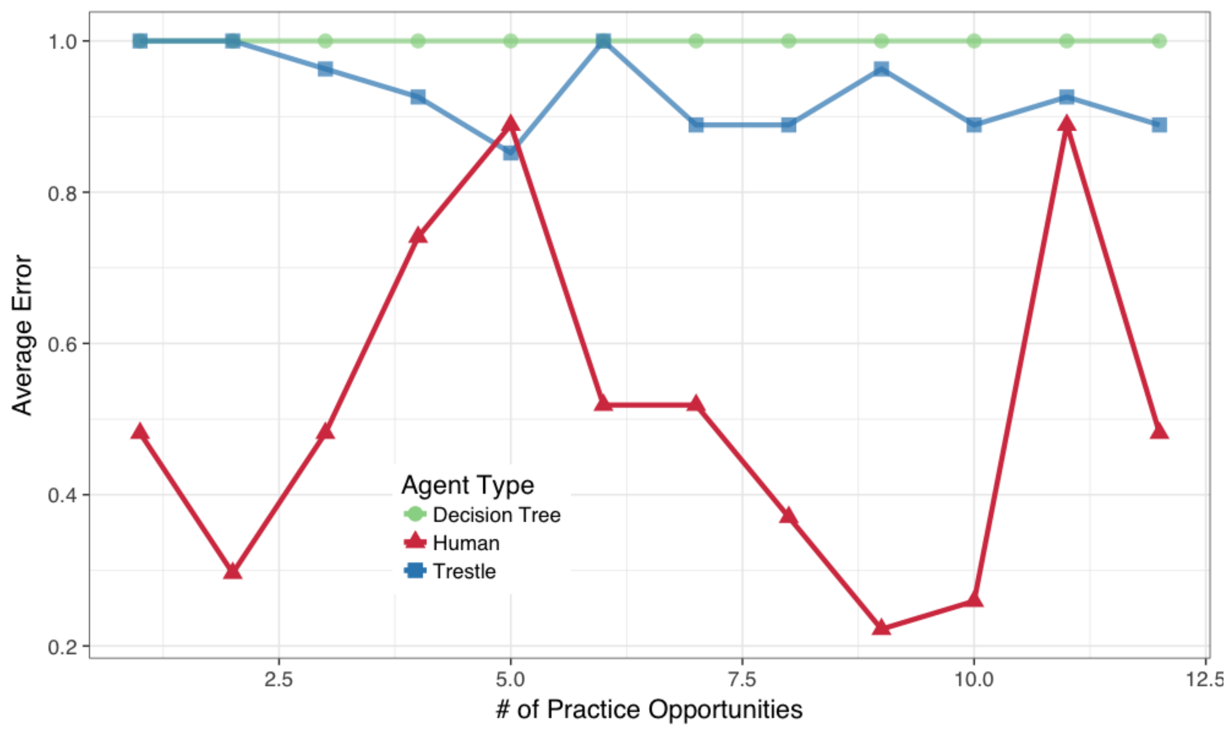


Figure A.3: Learning curves for humans and the two models on the “a” skill in the article selection tutor.

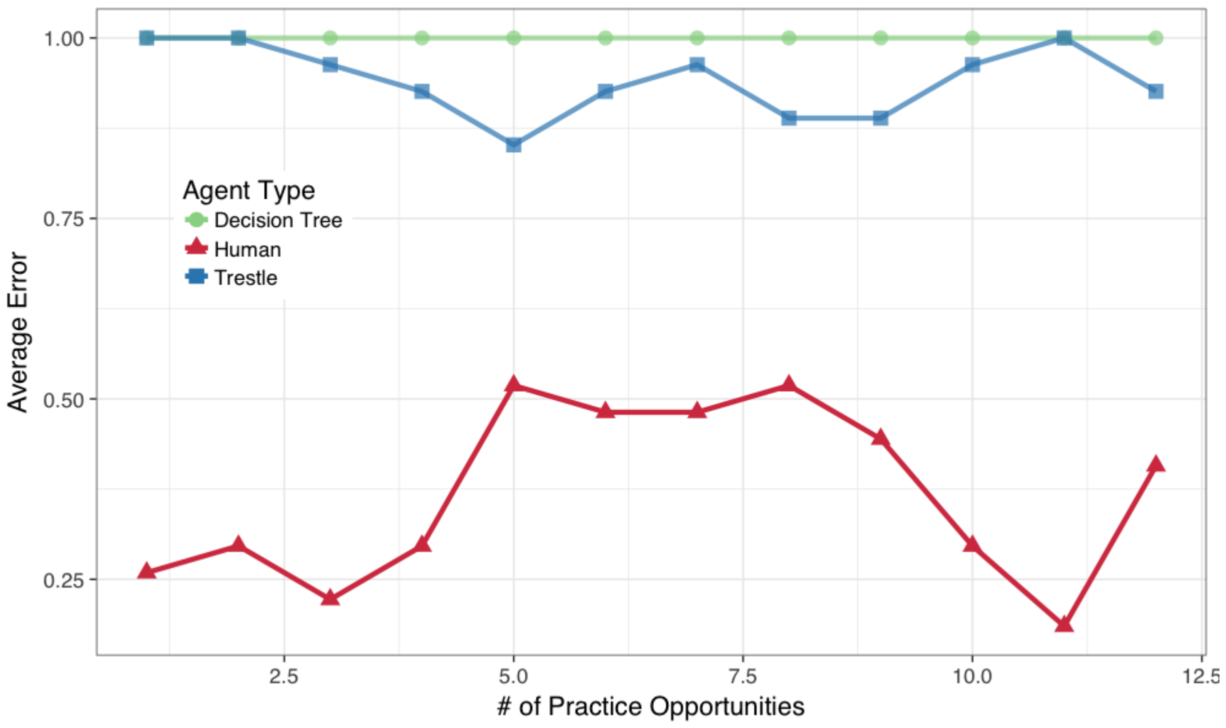


Figure A.4: Learning curves for humans and the two models on the “an” skill in the article selection tutor.

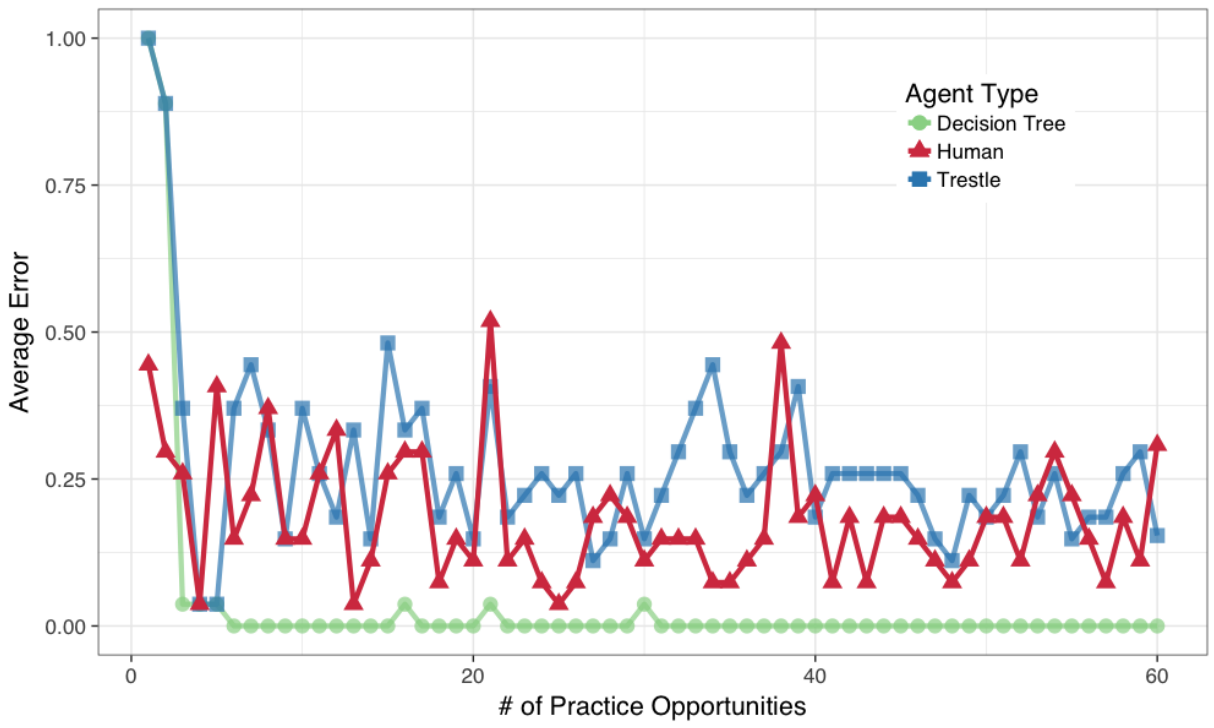


Figure A.5: Learning curves for humans and the two models on the “the” skill in the article selection tutor.

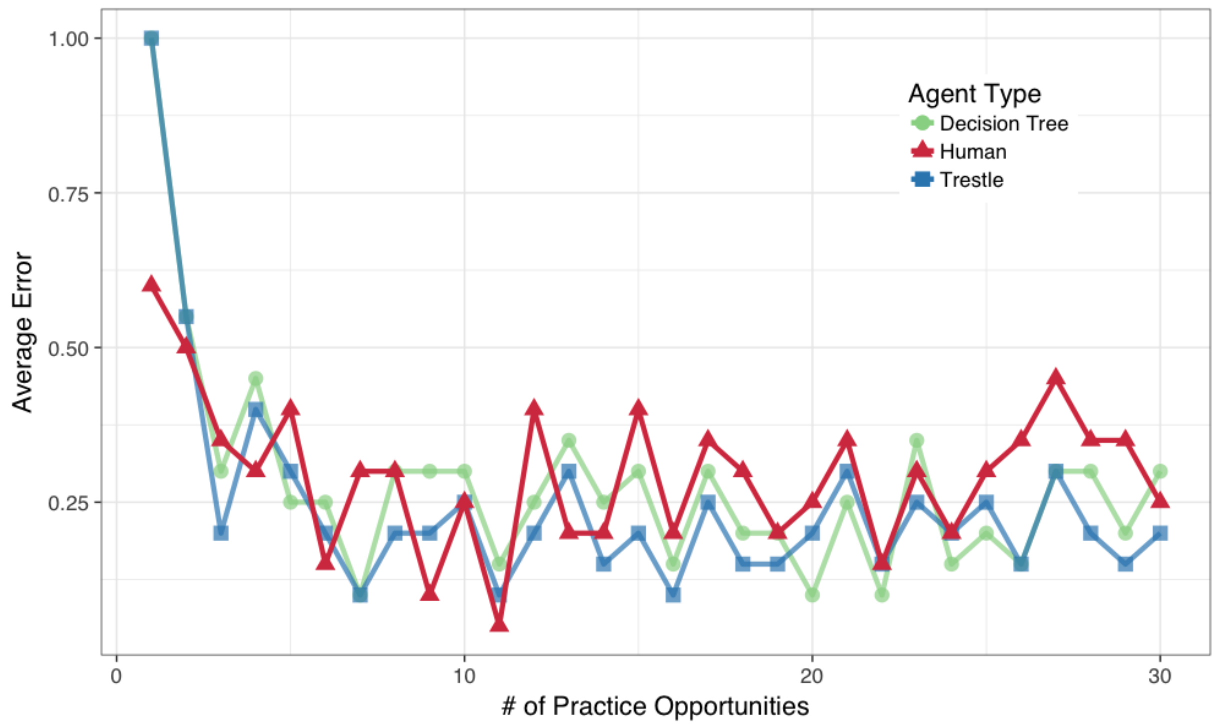


Figure A.6: Overall learning curves for the humans and the two models in the RumbleBlocks tutor.

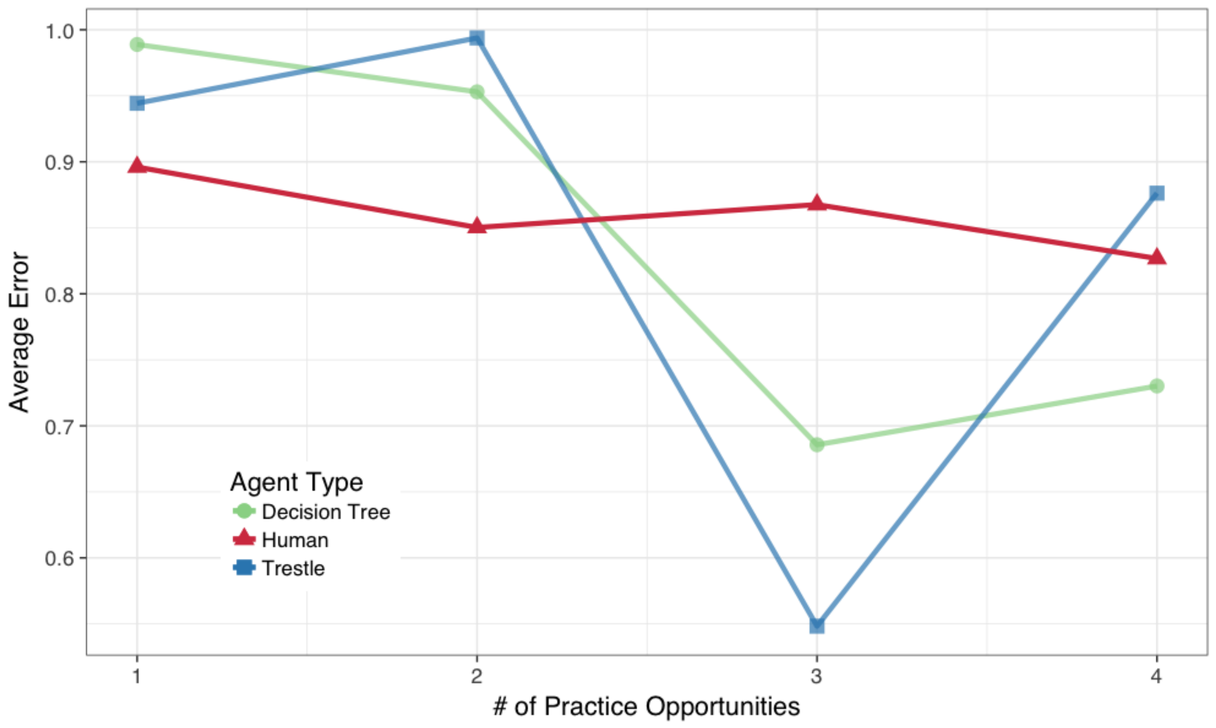


Figure A.7: Overall learning curves for the humans and the two models in the boxes and arrows tutor on the hard problems.

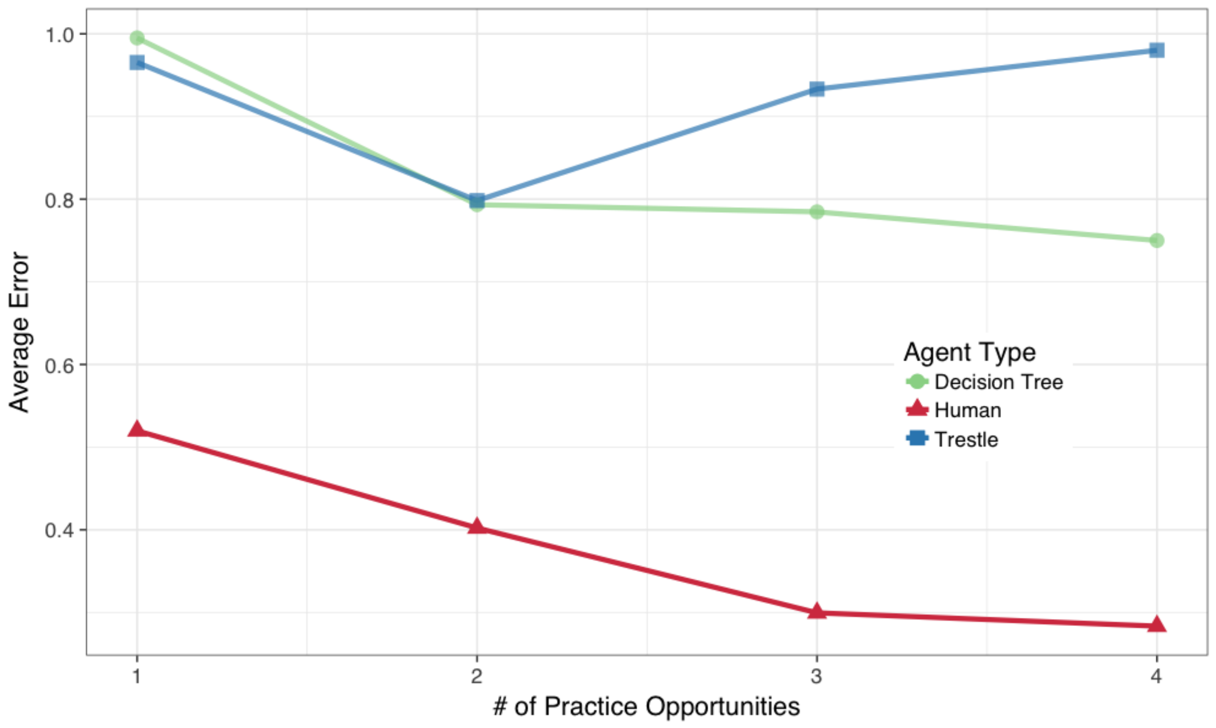


Figure A.8: Overall learning curves for the humans and the two models in the boxes and arrows tutor on the easy problems. These simulation results are excluded from the other analyses because performance on easy problems is highly dependent on prior knowledge, which the models do not take into account.

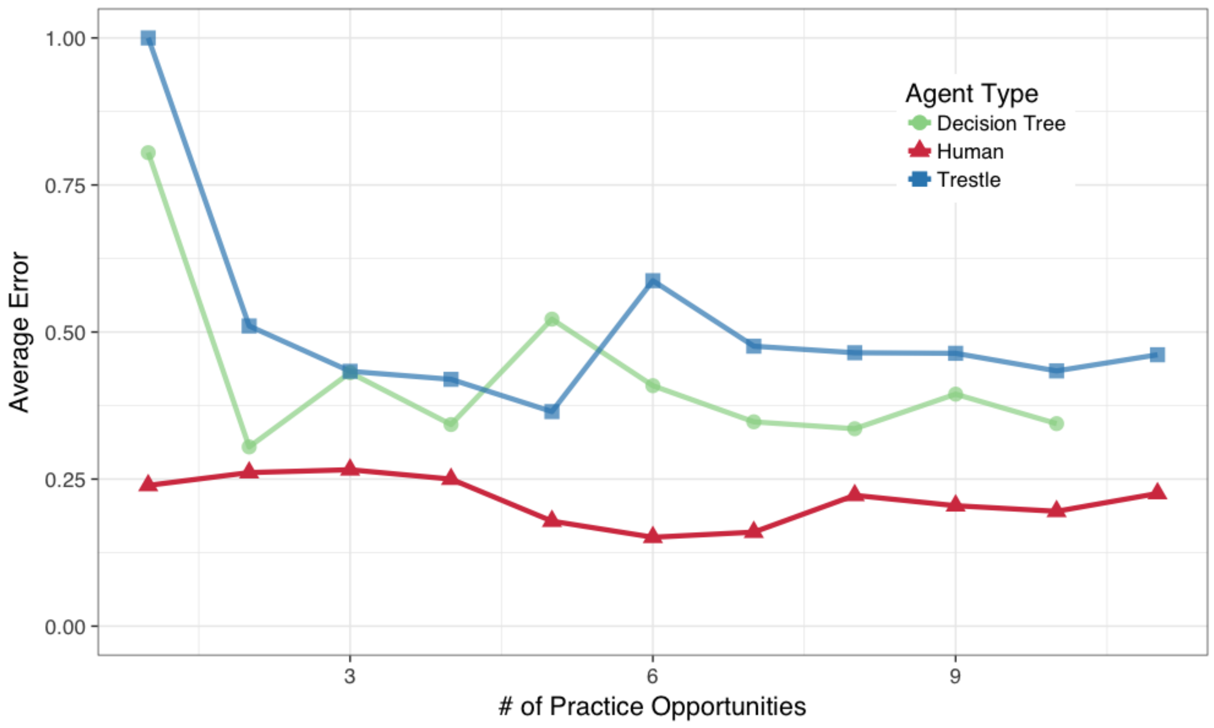


Figure A.9: Overall learning curves for the humans and the two models in the stoichiometry tutor.

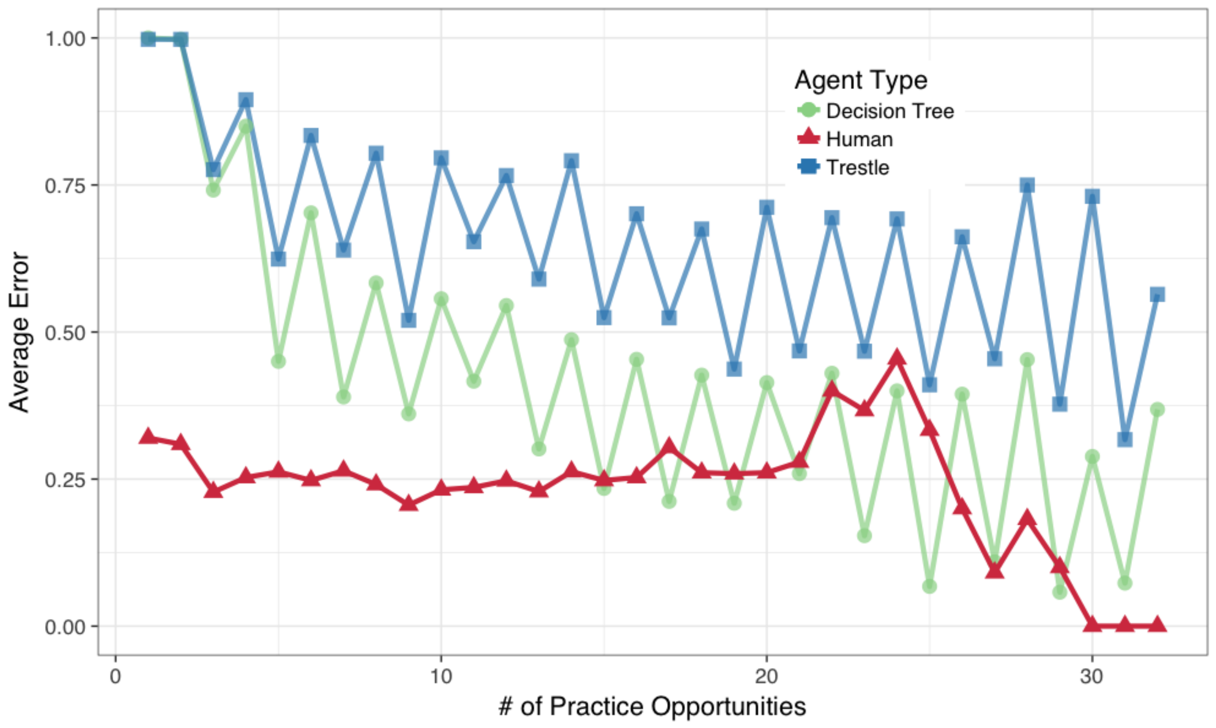


Figure A.10: Overall learning curves for the humans and the two models in the equation solving tutor.

References

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st international conference on machine learning* (pp. 1–8).
2, 2.2

Aleven, V., McLaren, B. M., 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*(2), 101–130.
5.6

Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. R. (2006). The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley, & C. Tak-Wai (Eds.), *Proceedings of the 8th international conference on intelligent tutoring systems* (pp. 61–70).
1, 5.3, 6.3

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.
1, 4.3, 4.5, 6.3

Anderson, J. R. (1982). Acquisition of Cognitive Skill. *Psychological Review*, *89*, 369–406.
2.2

Anderson, J. R. (1993). *Rules of the Mind*. New York: Lawrence Earlbaum Associates.
2.2

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill Acquisition and the LISP Tutor. *Cognitive Science*, *13*, 467–505.
5.1

Baker, R. S., & Inventado, P. S. (2014). Educational Data Mining and Learning Analytics. In J. A. Larusson & B. White (Eds.), *Learning analytics from research to practice* (pp. 61–75). New York: Springer.
5.1

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software*, *67*(1), 1–48.
5.4.3, 6.2, 6.4.1, 6.4.2, 6.4.4

Beal, C. R., Walles, R., Arroyo, I., & Woolf, B. P. (2007). On-line Tutoring for Math Achievement Testing: A Controlled Evaluation. *Journal of Interactive Online Learning*, *6*, 1–13.

1

Bowen, W. G., Chingos, M. M., Lack, K. A., & Nygren, T. I. (2013). Interactive Learning Online at Public Universities: Evidence from a Six-Campus Randomized Trial. *Journal of Policy Analysis and Management*, *33*(1), 94–111.
(document)

Brazdil, P. (1978). Experimental Learning Model. In *Proceedings of the 1978 aisb/gi conference on artificial intelligence* (pp. 46–50).
2.2, 3.1

Carbonell, J. G., Etzioni, O., Gil, Y., Joseph, R., Knoblock, C., Minton, S., & Veloso, M. (1991). PRODIGY: An integrated architecture for planning and learning. *SIGART Bulletin*, *2*, 51–55.
3.1

Card, S. K., Moran, T. P., & Newell, A. (1986). The model human processor: An engineering model of human performance. In *The handbook of human perception* (pp. 45–50).
(document), 1, 7

Cen, H. (2009). *Generalized Learning Factors Analysis: Improving Cognitive Models with Machine Learning* (Unpublished doctoral dissertation). Carnegie Mellon University.
1, 10

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).
5.1, 5.6

Chen, Z., & Klahr, D. (1999). All Other Things Being Equal: Acquisition and Transfer of the Control of Variables Strategy. *Child Development*, *70*(5), 1098–1120.
4.2, 4.3.1

Christel, M. G., Stevens, S. M., Maher, B. S., Brice, S., Champer, M., Jayapalan, L., . . . Lomas, D. (2012). RumbleBlocks: Teaching science concepts to young children through a Unity game. In *Proceedings of the 17th international conference on computer games: Ai, animation, mobile, interactive multimedia, educational & serious games* (pp. 162–166).
(document), 6.2

Clark, R. E., Feldon, D. F., van Merriënboer, J. J. G., Yates, K., & Early, S. (2008). 1, 2
In J. M. Spector, M. G. Sosman, M. D. van Merriënboer, & M. P. Driscoll (Eds.), *Handbook of research on educational communications and technology* (pp. 578–591). Mahwah, NJ: International Journal of Educational Research.

Collins, A., Brown, J. S., & Newman, S. E. (1987). *Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics* (Tech. Rep. No. 403). Washington, DC: National Institute of Education.
1, 2

Corbett, A. T., & Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, *4*(4), 253–278.
5.1, 5.5

Dejong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*, 145–176.
3.3

Dent, L., Boticario, J., McDermott, J. P., & Mitchell, T. M. (1992). A personal learning apprentice. In *Proceedings of the 10th national conference on artificial intelligence* (pp. 96–103).
2

Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like Models of Recognition and Learning. *Cognitive Science*, *8*, 305–336.
5.1

Fikes, R. E., Hart, P., & Nilsson, N. J. (1972). Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, *3*, 251–288.
2.2, 3.1

Fisher, D. H. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, *2*, 139–172.
3.3

Fisher, D. H., & Yoo, J. (1993). Categorization, Concept Learning, and Problem Solving: A Unifying View. In G. Nakamura, R. Taraban, & D. Medin (Eds.), *The psychology of learning and motivation* (pp. 219–255). San Diego: Academic Press.
5.1

Gobert, J. D., & Koedinger, K. R. (2011). Using Model-Tracing to Conduct Performance Assessment of Students' Inquiry Skills within a Microworld. In *Proceedings of the meeting for the society for research on educational effectiveness.*
4.4

Graesser, A. C., VanLehn, K., Rose, C., Jordan, P. W., & Harter, D. (2001). Intelligent Tutoring Systems with Conversational Dialogue. *AI Magazine*, *22*(4), 39.
1

Harpstead, E. (2017). *Projective replay analysis: a reflective approach for aligning educational games to their goals* (Unpublished doctoral dissertation). Carnegie Mellon University.
7

Heathcote, A., Brown, S., & Mewhort, D. J. K. (2000). The power law repealed: The case for an exponential law of practice. *Psychonomic Bulletin & Review*, *7*(2), 185–207.
5.1

Jarvis, M. P., Nuzzo-Jones, G., & Heffernan, N. T. (2004). Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems. In *Proceedings of the 7th international conference on intelligent tutoring systems* (pp. 541–553).
1

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.
4.3.2

Koedinger, K. R., & Anderson, J. R. (1997). Intelligent Tutoring Goes To School in the Big

City. *International Journal of Artificial Intelligence in Education*, *8*, 1–14.
(document), 1

Koedinger, K. R., Baker, R. S. J. d., Cunningham, K., Skogsholm, A., Leber, B., & Stamper, J. (2010). A Data Repository for the EDM community: The PSLC DataShop. In C. Romero, S. Ventura, M. Pechenizkiy, & R. S. J. d. Baker (Eds.), *Handbook of educational data mining.* Boca Raton: CRC Press.
1, 3.4, 5.1, 6.2

Koedinger, K. R., Booth, J. L., & Klahr, D. (2013). Instructional Complexity and the Science to Constrain It. *Science*, *342*(6161), 935–937.
1

Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2012). The Knowledge-Learning-Instruction (KLI) framework: Toward bridging the science-practice chasm to enhance robust student learning . *Cognitive Science*, *36*, 757–798.
1, 5.4.2, 6.1, 6.4.1

Koedinger, K. R., Stamper, J., McLaughlin, E., & Nixon, T. (2013). Using Data-Driven Discovery of Better Student Models to Improve Student Learning. In H. C. Lane, K. Yacef, J. Mostow, & P. I. Pavlik (Eds.), *Proceedings of 16th international conference on artificial intelligence in education* (pp. 421–430).
1

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An Architecture for General Intelligence*. *Artificial Intelligence*, *33*, 1–64.
2.2, 3.1

Langley, P. (1985). Learning to Search: From Weak Methods to Domain-Specific Heuristics*. *Cognitive Science*, *9*, 217–260.
2.2, 5.1, 5.6

Langley, P., Choi, D., & Rogers, S. (2009). Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research*, *10*(4), 316–332.
3.1, 3.2

Langley, P., & Ohlsson, S. (1984). Automated cognitive modeling. In *Proceedings of the 4th national conference on artificial intelligence* (pp. 193–197).
2.2, 3.4, 5.6

Lee, H. S., Betts, S., & Anderson, J. R. (2015). Learning Problem-Solving Rules as Search Through a Hypothesis Space. *Cognitive Science*, *40*(5), 1036–1079.
6.2, 6.2, 6.4.2

Li, N. (2013). *Integrating Representation Learning and Skill Learning in a Human-Like Intelligent Agent* (Unpublished doctoral dissertation). Carnegie Mellon University.
(document), 2.2, 3.4, 6.1, 6.4.1

Li, N., Cohen, W. W., & Koedinger, K. R. (2012). Efficient Cross-Domain Learning of Complex Skills. In *Proceedings of the 11th international conference on intelligent tutoring systems* (pp. 493–498).

1

Li, N., Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2014). Integrating representation learning and skill learning in a human-like intelligent agent. *Artificial Intelligence*, *219*, 67–91.
1, 3.1, 4, 3.1, 5.1, 5.6

Li, N., Schreiber, A. J., Cohen, W. W., & Koedinger, K. R. (2012). Efficient Complex Skill Acquisition Through Representation Learning. *Advances in Cognitive Systems*, *2*, 149–166.
1, 3.4

Li, N., Stampfer, E., Cohen, W. W., & Koedinger, K. R. (2013). General and Efficient Cognitive Model Discovery Using a Simulated Student. In M. Knauff, M. Paulen, N. Sebanz, & I. Wachsmuth (Eds.), *Proceedings of the 35th annual meeting of the cognitive science society.*
1, 2.2

Li, N., Stracuzzi, D. J., Langley, P., & Nejati, N. (2009). Learning hierarchical skills from problem solutions using means-ends analysis. In *Proceedings of the 31st annual meeting of the cognitive science society.*
2.2, 3.1, 3.2

Liu, R., McLaughlin, E. A., & Koedinger, K. R. (n.d.). Closing the loop: Automated data-driven skill model discoveries lead to improved instruction and learning gains. 1

Lomas, D., Patel, K., Forlizzi, J. L., & Koedinger, K. R. (2013). Optimizing challenge in an educational game using large-scale design experiments. In *Proceedings of the 31st annual conference on human factors in computing systems* (pp. 89–98).
1

MacLellan, C. J., Harpstead, E., Aleven, V., & Koedinger, K. R. (2016). TRESTLE: A Model of Concept Formation in Structured Domains. *Advances in Cognitive Systems*, *4*, 131–150.
(document), 3.3, 3.4, 5.1, 6.2, 6.2

MacLellan, C. J., Harpstead, E., Patel, R., & Koedinger, K. R. (2016). The Apprentice Learner Architecture: Closing the loop between learning theory and educational data. In *Proceedings of the 9th international conference on educational data mining.*
(document)

MacLellan, C. J., Harpstead, E., Wiese, E. S., & Zou, M. (2015). Authoring Tutors with Complex Solutions: A Comparative Analysis of Example Tracing and SimStudent. In *Workshops at the 17th international conference on artificial intelligence in education* (pp. 35–44).
(document), 1, 2.2

MacLellan, C. J., Koedinger, K. R., & Matsuda, N. (2014). Authoring Tutors with SimStudent: An Evaluation of Efficiency and Model Quality. In S. Trausen-Matu & K. Boyer (Eds.), *Proceedings of the 8th international conference on intelligent tutoring systems.*
(document), 4.1, 4.3.1, 4.3.2, 4.5, 6.3, 6.4.5, 6.4.5, 6.5

Martínez, D., Alenyà, G., & Torras, C. (2015). Relational reinforcement learning with guided demonstrations. *Artificial Intelligence*, *247*, 295–312.

2.2

Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2014, May). Teaching the Teacher: Tutoring SimStudent Leads to More Effective Cognitive Tutor Authoring. *International Journal of Artificial Intelligence in Education*, *25*(1), 1–34.
(document), 1, 2, 4.1, 4.5, 6.3

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2007). Evaluating a Simulated Student using Real Students Data for Training and Testing. In C. Conati, K. McCoy, & G. Paliouras (Eds.), *Proceedings of the 11th international conference on user modeling* (pp. 107–116).
3.4

Matsuda, N., Lee, A., Cohen, W. W., & Koedinger, K. R. (2009). A Computational Model of How Learner Errors Arise from Weak Prior Knowledge. In N. Taatgen & H. van Rijn (Eds.), *Proceedings of the human factors in computing systems conference* (pp. 1288–1293).
(document), 2.2, 5.1, 5.1

Matsuda, N., Yarzebinski, E., Keiser, V., Cohen, W. W., & Koedinger, K. R. (2011). Learning by Teaching SimStudent – An Initial Classroom Baseline Study Comparing with Cognitive Tutor. In *Proceedings of the 15th international conference on artificial intelligence in education.*
2.2

McDaniel, R. G., & Myers, B. A. (1999). Getting more out of programming-by-demonstration. In *Proceedings of the human factors in computing systems conference* (pp. 442–449).
4.5

McLaren, B. M., Lim, S.-J., Gagnon, F., Yaron, D., & Koedinger, K. R. (2006). Studying the Effects of Personalized Language and Worked Examples in the Context of a Web-Based Intelligent Tutor. In *Proceedings of the 8th international conference on intelligent tutoring systems* (pp. 318–328).
6.2, 6.2

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*(2-3), 363–391.
2.2

Mitchell, T. M., & Banerji, R. (1983). Learning by experimentation: acquiring and refining problem-solving heuristics. In R. S. Michalski (Ed.), *Machine learning* (pp. 163–190). Berlin: Springer.
2.2

Mitchell, T. M., Mahadevan, S., & Steinberg, L. I. (1985). LEAP: A learning Apprentice for VLSI Design. In *The proceedings of the 9th international joint conference on artificial intelligence* (pp. 1–8).
2.2

Mitrovic, A., Martin, B., & Mayo, M. (2002). Using evaluation to shape ITS design: Results and experiences with SQL-Tutor. *User Modeling and User-Adapted Interaction*, *12*(2-3), 243–279.
1

Mooney, R. J. (1987). *A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding* (Unpublished doctoral dissertation). University of Illinois at Urbana-Champaign.
2.2

Murray, T. (1999). Authoring Intelligent Tutoring Systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, *10*, 98–129.
(document), 1

Murray, T. (2003). An Overview of Intelligent Tutoring System Authoring Tools: Updated analysis of the state of the art. In Murray, Ainsworth, & Blessing (Eds.), *Authoring tools for advanced technology learning environments* (pp. 493–546). Netherlands: Kluwer Academic Publishers.
(document), 1

Murray, T. (2005, May). Having It All, Maybe: Design Tradeoffs in ITS Authoring Tools. In *Proceedings of the 3rd international conference on intelligent tutoring systems* (pp. 93–101).
1

Myers, B., Hudson, S. E., & Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, *7*, 3–28.
4.2, 4.5

Nason, S., & Laird, J. E. (2005). Soar-RL: Integrating reinforcement learning with Soar. *Cognitive Systems Research*, *6*, 51–59.
2.2

Nathan, M., Koedinger, K. R., & Alibali, M. (2001). Expert Blind Spot: When Content Knowledge Eclipses Pedagogical Content Knowledge. In *Proceedings of 3rd international conference on cognitive science* (pp. 644–648).
1

Neves, D. M. (1985). Learning procedures from examples and by doing. In *Proceedings of the 9th international joint conference on artificial intelligence* (pp. 624–630).
2.2

Newell, A. (1973). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.), *Visual information processing* (pp. 283–308). New York: Academic Press.
5.1

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* Hillsdale: Earlbaum.
5.1

Newell, A., & Simon, H. A. (1972). *Human Problem Solving*. New York: Prentice-Hall.
2.2

Ohlsson, S. (2011). *Deep Learning*. New York: Cambridge University Press.
3

Olsen, J. K., Aleven, V., & Rummel, N. (2015). Predicting Student Performance In a Collabora-

tive Learning Environment. In *Proceedings of the 8th international conference on educational data mining.*
5.6

Pane, J. F., Griffin, B. A., McCaffrey, D. F., & Karam, R. (2013). *Effectiveness of Cognitive Tutor Algebra I at Scale* (Tech. Rep. No. WR-984-DEIES). Santa Monica: RAND Corporation.
(document)

Patel, R., Liu, R., & Koedinger, K. R. (2016). When to Block versus Interleave Practice? Evidence Against Teaching Fraction Addition before Fraction Multiplication. In *Proceedings of the 38th annual meeting of the cognitive science society.*
5.1, 5.2, 5.3, 6.2

Pavlik, P. I., Bolster, T., Wu, S.-m., Koedinger, K., & MacWhinney, B. (2008). Using Optimally Selected Drill Practice to Train Basic Facts. In *Proceedings of the 12th international conference on intelligent tutoring systems* (pp. 593–602).
6.2, 6.2

Pavlik, P. I., Kelly, C., & Maass, J. K. (2016). The Mobile Fact and Concept Training System (MoFaCTS). In *Proceedings of the 12th international conference on intelligent tutoring systems* (pp. 247–253).
6.3

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.
3.3, 3.4, 5.1

Quinlan, J. R. (1986, March). Induction of decision trees. *Machine Learning*, *1*, 81–106.
3.4, 5.1

Quinlan, J. R., & Cameron-Jones, R. M. (1995). Induction of logic programs: FOIL and related systems. *New Generation Computing*, *13*(3-4), 287–312.
3.4

Ritter, S., Anderson, J. R., Koedinger, K. R., & Corbett, A. T. (2007). Cognitive Tutor: Applied research in mathematics education. *Psychonomic Bulletin & Review*, *14*(2), 249–255.
1, 3.4, 6.2, 6.2

Ritter, S., & Koedinger, K. R. (1996). An Architecture For Plug-In Tutor Agents. *Journal of Interactive Learning Research*, *7*, 315–347.
2.1

Romero, C., & Ventura, S. (2010). Educational Data Mining: A Review of the State of the Art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *40*(6), 601–618.
5.1

Russell, S. J., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson Education.
3.2

Sao Pedro, M. A., Gobert, J. D., Heffernan, N. T., & Beck, J. E. (2009). Comparing Pedagogical Approaches for Teaching the Control of Variables Strategy. In N. Taatgen & H. van Rijn (Eds.), *Proceedings of the 31st annual conference of the cognitive science society* (pp. 1–6).
4.2

Schmidt, R. A., & Bjork, R. A. (1992). New Conceptualizations of Practice: Common Principles in Three Paradigms Suggest New Concepts for Training. *Psychological Science*, *3*(4), 207–217.
5.6

Schneider, M., Rittle-Johnson, B., & Star, J. R. (2011). Relations among conceptual knowledge, procedural knowledge, and procedural flexibility in two samples differing in prior knowledge. *Developmental Psychology*, *47*(6), 1525–1538.
4.4

Sleeman, D., Langley, P., & Mitchell, T. M. (1982). Learning from Solution Paths: An Approach to the Credit Assignment Problem. *AI Magazine*, *3*, 48–52.
2.1, 2.2

Sottilare, R. A., & Holden, H. K. (2013, June). Motivations for a Generalized Intelligent Framework for Tutoring (GIFT) for Authoring, Instruction, and Analysis. In R. A. Sottilare & H. K. Holden (Eds.), *Aied 2013 workshop on recommendations for authoring, instructional strategies and analysis for intelligent tutoring systems (its): Towards the development of a generalized intelligent framework for tutoring (gift)* (pp. 1–150).
1

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational Reinforcement Learning: An Overview. In *Proceedings of the icml workshop on relational reinforcement learning* (pp. 1–9).
2.2

Tenison, C., & Anderson, J. R. (2015). Modeling the Distinct Phases of Skill Acquisition. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *42*, 749–767.
5.1

Tenison, C., & MacLellan, C. J. (2014). Modeling Strategy Use in an Intelligent Tutoring System: Implications for Strategic Flexibility. In *Proceedings of the 12th international conference on intelligent tutoring systems* (pp. 466–475).
4.4, 4.5

Ur, S., & VanLehn, K. (1995). Steps: A Simulated, Tutorable Physics Student. *Journal of Artificial Intelligence in Education*, *6*, 405–437.
2.2, 3.1, 3.4, 5.1, 7

VanLehn, K. (1983). Human Procedural Skill Acquisition: Theory, Model, and Psychological Validation. In *Proceedings of the 4th national conference on artificial intelligence* (pp. 420–423).
2.2, 3.4, 5.6

VanLehn, K. (1999). Rule-Learning Events in the Acquisition of Complex Skill: An Evaluation of Cascade. *The Journal of the Learning Sciences*, *8*, 71–125.

2.2

VanLehn, K. (2006). The Behavior of Tutoring Systems. *International Journal of Artificial Intelligence in Education*, *16*(3), 227–265.
1, 2, 2.1, 2.1, 2.1

VanLehn, K. (2011). The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. *Educational Psychologist*, *46*(4), 197–221.
1

VanLehn, K., Jones, R. M., & Chi, M. T. H. (1991). Modeling the self-explanation effect with Cascade 3. In *Proceedings of the human factors in computing systems conference* (pp. 132–137).
2.2, 3.1, 3.2, 3.4, 7

VanLehn, K., Ohlsson, S., & Nason, R. (1994). Applications of simulated students: An exploration. *Journal of Interactive Learning Research*, *5*, 135–175.
(document), 1

van Lent, M., & Laird, J. E. (2001). Learning procedural knowledge through observation. In *Proceedings of the international conference on knowledge capture* (pp. 179–9).
2.2

Waalkens, M., Aleven, V., & Taatgen, N. (2013). Does supporting multiple student strategies lead to greater learning and motivation? Investigating a source of complexity in the architecture of intelligent tutoring systems. *Computers & Education*, *60*, 159–171.
4.4

Wilkins, D. C., Clancey, W. J., & Buchanan, B. G. (1986). Overview of the Odysseus Learning Apprentice. In T. Mitchell (Ed.), *Machine learning* (pp. 369–373). Boston: Springer.
2.2

Wylie, R., Koedinger, K. R., & Mitamura, T. (2009). Is self-explanation always better? The effects of adding self-explanation prompts to an English grammar tutor. In *Proceedings of the 31st annual conference of cognitive science society* (pp. 1300–1305).
6.2, 6.2