Noname manuscript No.
(will be inserted by the editor)

Domain-General Tutor Authoring with Apprentice Learner Models

Christopher J. MacLellan ·
Kenneth R. Koedinger

Received: date / Accepted: date

Abstract Intelligent tutoring systems are effective for improving students' learning outcomes (Bowen et al., 2013; Koedinger and 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 paper, we explore the use of computational models of apprentice learning, or computer models that learn interactively from examples and feedback, for authoring expert-models via demonstrations and feedback (Matsuda et al., 2014) across a wide range of domains.

To support these investigations, we present the Apprentice Learner Architecture, which posits the types of knowledge, performance, and learning components needed for apprentice learning. We use this architecture to create two models: the Decision Tree model, which non-incrementally learns skills, and the Trestle model, which instead learns incrementally. Both models draw on the same small set of prior knowledge (six operators and three types of rela-

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), two National Science Foundation Awards (#DRL-0910176 and #DRL-1252440), and by a DARPA award (#HR00111990055). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. We would also like to thank Carnegie Learning, Inc. for providing the Cognitive Tutor data that supported this work.

Christopher J. MacLellan
Information Science Department
Drexel University
Philadelphia, PA 19104
E-mail: christopher.maclellan@drexel.edu

Kenneth R. Koedinger
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15214
E-mail: koedinger@cmu.edu

tional knowledge) to support expert model authoring. Despite their limited prior knowledge, we demonstrate their use for efficiently authoring a novel experimental design tutor and show that they are capable of learning an expert model for seven additional tutoring systems that teach a wide range of knowledge types (associations, categories, and skills) across multiple domains (language, math, engineering, and science).

This work shows that apprentice learner models are efficient for authoring tutors that would be difficult to build with existing non-programmer authoring approaches (e.g., experimental design or stoichiometry tutors). Further, we show that these models can be applied to author tutors across eight tutor domains even though they only have a small, fixed set of prior knowledge. This work lays the foundation for new interactive machine-learning based authoring paradigms that empower teachers and other non-programmers to build pedagogically effective educational technologies at scale.

Keywords Authoring Tools, Intelligent Tutoring Systems, Interactive Machine Learning, Simulated Agents

1 Introduction

Intelligent tutoring systems have been shown to improve student learning across multiple domains (Beal et al., 2007; Graesser et al., 2001; Koedinger and 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 et al., 2006; Murray, 1999, 2003; Sottilare and Holden, 2013). 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 four 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 tutor authoring remains an open research question.

To develop technology to support tutor authoring, we 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 et al., 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.

Given these current practices, what is needed is a tool that leverages learning science theory to guide the initial design phase and to support designers in the build process. In this paper, we explore the use of computational models of human learning from examples and feedback, what we 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 learn like humans do.

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 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; Matsuda et al., 2014), 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 and 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

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

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

et al. (2013) showed that skill models discovered by training SimStudent (a particular apprentice learner model) align with student data as well as, or better than, expert-constructed skill models across three domains. Finally, unlike prior automated expert model authoring approaches (e.g., Barnes et al., 2008; Kumar et al., 2014; McLaren et al., 2004), apprentice learner models do not require a deployable system with existing users or previously available data to support the tutor authoring process.

In the current work, we aim to evaluate the potential for apprentice learner models to support tutor authoring by assessing their applicability and efficiency across a broad range of tutors and domains. While prior work has started to quantify the efficiency gains of these models, particularly the SimStudent model (Jarvis et al., 2004; Li et al., 2014; Matsuda et al., 2014), the past work only focuses on evaluating efficiency gains for a small subset of tutors (primarily an equation solving tutor) and domains (primarily math). Additionally, the models from these prior studies utilize domain-specific prior knowledge to support authoring in these domains, suggesting that a would-be user needs to author additional content for their domain. Li (2013) has investigated how to automatically discover domain-specific prior knowledge using unsupervised learning. However, these approaches require access to training data in one batch upfront, which may not be available for novel domains. Further, it seems unlikely that teachers, or other non-technical domain experts, will be able to collect/clean-up domain-specific training data and apply unsupervised learning approaches to them. Taken together, the narrow focus of previous evaluations and the need for specialized domain knowledge minimizes claims that these tutor authoring models will be able to support efficient tutor authoring across domains.

The current study aims to build on this prior work by providing evidence to support the claims that apprentice learner models: (1) support efficient tutor authoring and (2) are domain general. Towards this end, we first describes the Apprentice Learner Architecture, which facilitates the construction of apprentice learner models, and describes two initial models built using this framework. Next, we provide a case study of using one of these models (the Decision Tree model) to author a novel experimental design tutors. We analyze the authoring efficiency of this model and show that it is more efficient than Example-Tracing, the current state-of-the-art in tutor authoring for non-programmers. This analysis provides evidence to support our first claim. We then extend this initial case study to evaluate both apprentice learner models (the Trestle and Decision Tree models) across seven additional tutoring domains. Our cross domain analysis compares these models to Example-Tracing as well as human learners and shows that the models are capable of learning across a wide range of tutors and domains, even without additional specialized domain knowledge. This second analysis provides evidence to support our second claim that apprentice learner models are domain-general tutor authoring tools. Finally, we conclude with discussions and directions for future work.

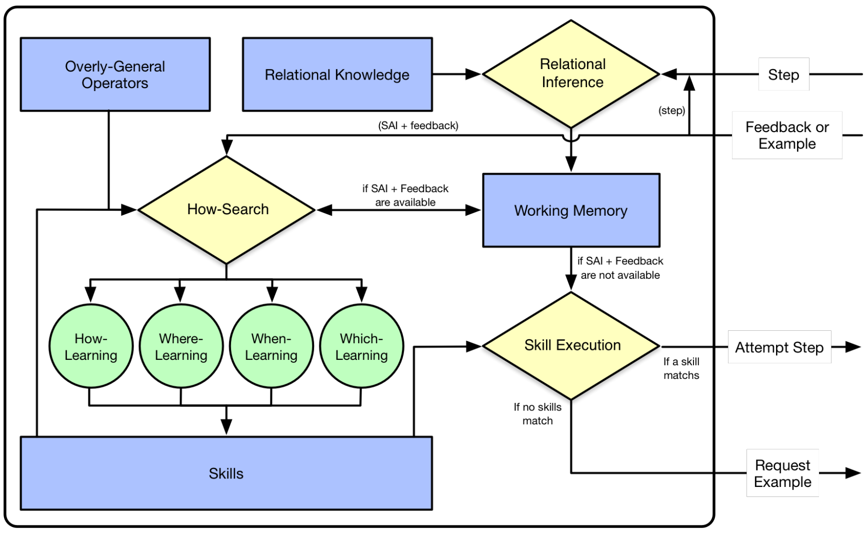


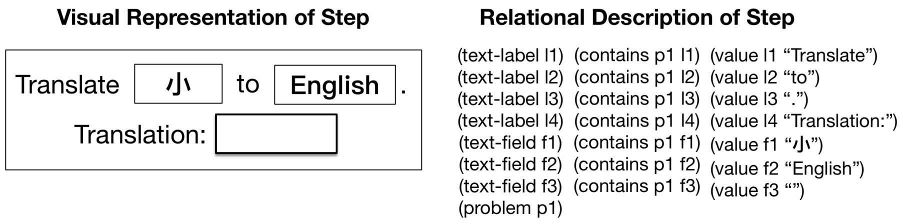
Fig. 1 The Apprentice Learner Architecture and its interactions with a tutor. Blue boxes represent knowledge structures, yellow diamonds represent performance components, and green circles represent learning components. The selection, action, and input (SAI) represent the contents of a step in the tutor.

2 The Apprentice Learner Architecture

In this section, we present a modular architecture that builds on prior systems, such as ACM (Langley and Ohlsson, 1984), CASCADE (VanLehn et al., 1991), STEPS (Ur and VanLehn, 1995), and SimStudent (Li et al., 2014), and unifies their mechanisms and theories. This architecture was designed to interact with tutoring systems. In particular, agents within the architecture can receive steps from a tutor, attempt these steps or request examples of how to perform these steps, and receive feedback or examples. Given these interactions, the architecture (see Figure 1) embodies a theory about the knowledge structures needed to support them and about the performance and learning components that operate over these structures.[3]

To support learning from tutor interactions, our architecture posits four knowledge structures (working memory, relational knowledge, overly-general operators, and skills), three performance components (relational inference, how search, and skill execution), and four learning mechanisms (how-, where-, when-, and which-learning). When an agent from this architecture is faced with a problem to solve (e.g., what is 2+3?), a match is made between previously learned skills and the current problem state (represented in working memory). If any skills match, the agent executes the one with highest utility. If no skills match (a typical initial response), then the agent requests a demonstration, or bottom out hint, from the tutor (e.g., the tutor might en-

[3] The architecture is open source and available at `https://github.com/apprenticelearner`.



```
(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)
```

Fig. 2 A visual representation of a step in the Chinese tutor and its accompanying relational description. The correct English translation for this problem is "small".

ter a 5 in the answer field). The agent then performs how-search to generate a sequence of mental operations (i.e., a procedure) that explains the tutor's demonstration, for example, the agent might explain the demonstration as the addition of the first and second numbers. Next, the agent uses its how-learning mechanism to compile this procedure into a new skill structure. After how-learning, the agent uses where-learning to induce relational heuristic conditions that constrain when the learned skill is considered for later use. Next, the agent uses when-learning to acquire a classifier that further constrains when the skill matches once its other conditions are met. Finally, the system uses which-learning to update the utility of the learned skill. The output of these steps (a procedure, relational conditions, a classifier, and utility) constitute a new skill. On subsequent problems, the agent attempts to apply learned skills that it thinks are applicable and receives correctness feedback from the tutor. This feedback is used in where-, when-, and which-learning to refine the skills relational conditions, classifier, and utility.

The architecture 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 human learning (e.g., if they fail to produce human-like mistakes), which is another research focus this architecture aims to support. 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, we discuss the implementations currently supported by the architecture and two specific models within this architecture that are the focus of the remainder of this paper.

2.1 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

Table 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))

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 shows a simple example of a step's encoding in working memory.[4] 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 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 can contain 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—each knowledge structure is evaluated once per a given binding of its head arguments. 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. Finally, for each successful match, the effects determine how working memory is augmented in response. Like conditions, effects can contain relations and functions.

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

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

Table 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 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)))

apply during explanation, when domain skills fail. Table 2 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 and 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.

The final long-term memory contains domain-specific *skills*, which are acquired from examples and feedback. Table 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).

2.2 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 paper utilize a forward-chaining approach that applies relational knowledge until quiescence or until a user-specified depth limit is reached. 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.

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 an 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 as well as the bindings of the pattern-matching variables to working memory elements. 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).

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 provide 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. 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. 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 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 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.

2.3 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.

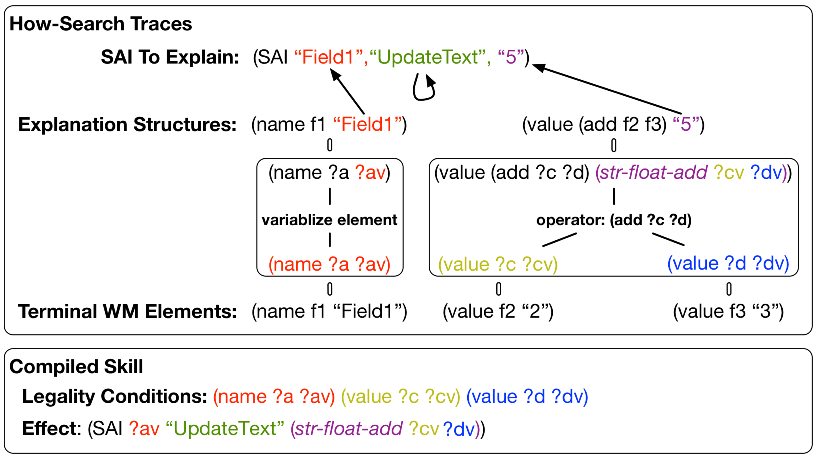

How-Search Traces
SAI To Explain: (SAI "Field1","UpdateText", "5")
Explanation Structures: (name f1 "Field1") (value (add f2 f3) "5")
(name ?a ?av)
variablize element
(name ?a ?av)
(value (add ?c ?d) (str-float-add ?cv ?dv))
operator: (add ?c ?d)
(value ?c ?cv)
(value ?d ?dv)
Terminal WM Elements: (name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

Fig. 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.

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 and 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 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. 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 and 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 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 applications, but not their negative ones. There are many possible approaches to relational condition learning, but all the models in this paper 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, the learner might acquire disjunctive 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 we did not explore them further.

Once where-learning completes, the system activates its third learning component, *when-learning*, which learns a classifier that further constrains when skills are applicable. This component takes the tutor feedback, working memory contents, and each skill application as input. The current working memory structure is augmented with additional information about how pattern-matching variables are bound to the working memory elements. These augmented structures are combined with the tutor feedback to produce labeled classification data, which is passed to a concept learning algorithm. Currently, the system is designed to support incremental algorithms, such as COBWEB (Fisher, 1987) or TRESTLE (MacLellan et al., 2016). These algorithms, which are based on psychological studies of how humans learn concepts, incrementally build probabilistic, multi-attribute classification trees. These trees are similar to decision trees, but branch on probabilistic combinations of all attributes rather than single attributes like decision trees. This probabilistic approach is akin to using a naive Bayes classifier to determine which branch to follow at each point in the classification tree. These incremental algorithms are also able to efficiently update this classification tree in response to each new datum without having to reprocess all of the previous training examples.

Our system also has an interface to the Scikit-learn library (Pedregosa et al., 2011), which implements many non-incremental concept learning approaches. In contrast to the incremental algorithms, these non-incremental approaches reprocess all of the experienced training examples each time they update their learned classifier. Our system receives the training examples one at a time (from the tutor); to support 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 learns an entirely new classifier from scratch using all available data in a single batch. Thus, the non-incremental approaches are less efficient to train (they reprocess all examples after each new example) than incremental approaches, but they learn higher accuracy classifiers because they reconsider all of the data during each update. Additionally, for any approaches that do not support relational representations (e.g., COBWEB or any of the Scikit-learn approaches), the relations are flattened into a boolean attribute-values representation using Scikit-learn's DictVectorizer transformer.

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.

2.4 The Decision Tree and Trestle models

The remainder of this paper 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 section, 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 et al., 2016). Across all of the studies in this paper, both models are given identical initial knowledge configurations that consists of three types of relational knowledge and six overly-general operators. In particular, they have relational knowledge for inferring equality of values (see value-equality in Table 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 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 et al. (2012). In addition to this relational knowledge, both models have overly-general operators for adding, subtracting, multiplying, dividing, rounding, and concatenating values. Table 2 shows the add and concatenate operators; the other operators are almost identical, but perform their respective operations.

3 An Initial Case Study in Expert Models Authoring

Now that we have described the Apprentice Learner Architecture, and the models set within it, we next turn to showcasing how one of the latter, the Decision Tree model, supports efficient tutor authoring. For clarity, the current section only focuses on one model, but the Trestle model could also be used for this purpose. 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 solution structures. 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 section investigates two questions: (1) is authoring with simulated students a viable approach when domain-specific knowledge is not available, and (2) how does the approach compare to Example-Tracing with mass production? To investigate these questions, we describe how to author a novel tutor for experimental design 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—as the model does not have any specialized experimental design knowledge. Additionally, when evaluating the efficiency of authoring with Example-Tracing, we 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 baseline for assessing the Decision Tree model's efficiency. After presenting our evaluation of these two expert-model authoring approaches, we conclude the section by discussing the limitations of each approach.

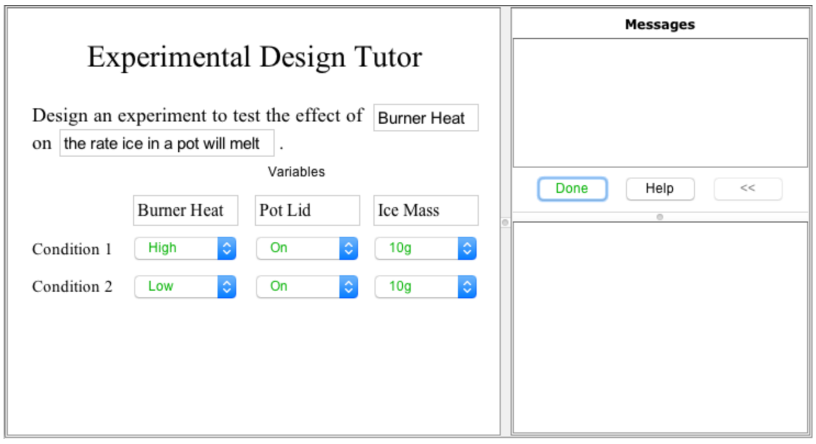
3.1 Experimental Design Task

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 and 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, we decided to use them to author a novel tutor for experimental design.

To coach students in designing good experiments, we created the tutor interface shown in Figure 4, 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 of "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 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 $(721 * 8)$ 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 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


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

Fig. 4 The experimental design tutor interface.

tutor prompts for the interface in Figure 4 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 of 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 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 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, it may be the case that students will learn more in our experimental design tutor if they have a larger number of solutions and solution paths, but the developer may be forced to prompts that are practical to author, even if they 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. We argue that, for authoring tools with low thresholds (i.e., for non-programmers), the ceiling is not well understood.

To investigate the capabilities and efficiency of the Decision Tree model and Example-Tracing with mass production, we built an experimental design tutor using each approach. To create the tutor interface and author the expert models, we used the Cognitive Tutor Authoring Tools (CTAT) (Aleven et al., 2009). This toolkit provides a drag-and-drop interface builder, which we used to create the interface shown in Figure 4. The toolkit also supports two modes for authoring expert models without programming, Example-Tracing and Simulated Student. We modified 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. Next we will describe how to author the experimental design expert model using each mode.

3.2 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 5 shows a behavior graph we authored for the experimental design tutor. The particular prompt chosen (many solutions with many paths) has eight unique configurations, so we demonstrated each unique configuration in the interface. Each unique configuration corresponds to one of the paths shown in the figure. Along each path, the variable values can be chosen 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

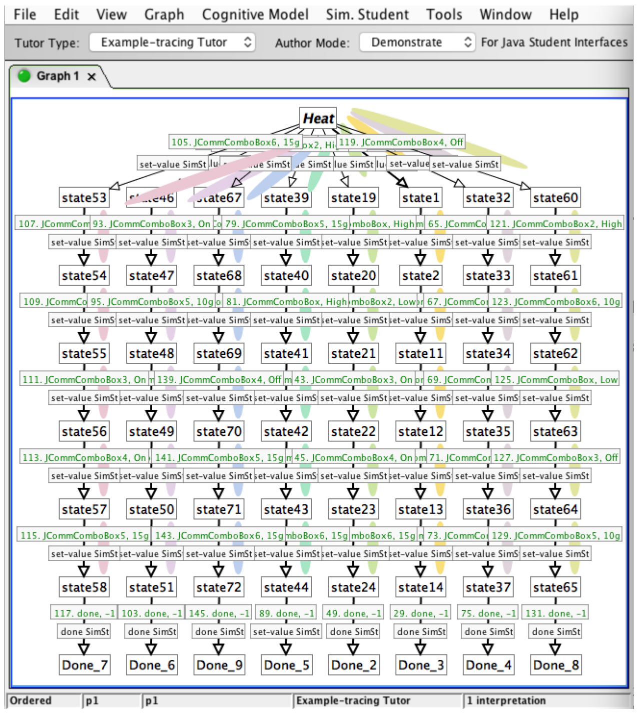


Fig. 5 A behavior graph for the experimental design tutor. The colored ellipsoids represent groups of actions that are unordered.

be executed in any order—drastically reducing the number of demonstrations necessary. Using this approach, we specified that the actions to set variable values along each path are unordered (denoted in the behavior graph by colored ellipsoids).

Once we had successfully authored a behavior graph for the first problem, we 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 and Klahr, 1999). To create a template for mass production, we first authored the same problem as before, but instead of entering specific values in the interface, we entered variables, such as "%(variable1)%" instead of "Burner Heat." Then we 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 (CTAT generates an empty Excel file in this format automatically). We then filled out the rows and columns in this spreadsheet with the new values for each variable and problem and used CTAT's mass production capability to combine the variablized behavior graph with the spreadsheet to create separate grounded behavior graphs for each problem like the one shown in Figure 5.

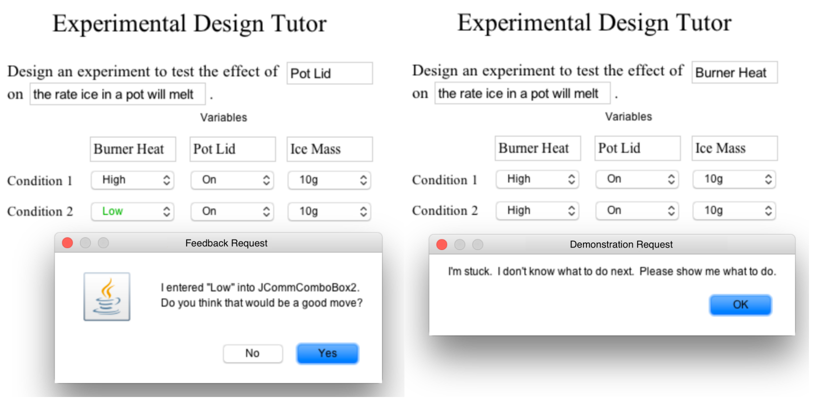
This approach supports different problems that have an identical behavior graph structure, such as replacing all instances of “Burner Heat” with another variable, “Ramp Slope”. However, if a problem varies in the structure of its behavior graph, such as asking the student to manipulate a variable in 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, we ultimately had to author three separate mass production templates, one for each variable column being targeted.

Next, we 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, we estimate that the behavior graphs for the experimental design tutor would take 26.96 minutes to build using Example-Tracing.[5] 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 by updating the contents of the mass production spreadsheet and then generating the new behavior graphs. It is worth noting that the number of variables or values cannot be changed as this would require new behavior graph templates.

3.3 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 directly in the experimental design tutor interface. 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

[5] 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.


Experimental Design Tutor

Design an experiment to test the effect of Pot Lid
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

Feedback Request

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

No Yes

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	High	On	10g

Demonstration Request

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

OK

Fig. 6 The Decision Tree model asking for feedback (left) and for a demonstration (right).

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 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. 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. When authoring the experimental design tutor for the current work, we 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 6 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 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. An interesting side effect of this interactive training is that it produces behavior graphs with both correct as well as incorrect (or buggy) links, which are often more difficult for instructional designers to anticipate and author.

To author an expert model using the Decision Tree model, we tutored it on a sequence of 20 experimental design problems presented in the tutor interface. Unlike Example-Tracing, we did not explicitly demonstrate every correct solution for each problem. Instead, the agent solved each problem a single way, and we provided it with demonstrations and feedback when requested. One 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, we use a solution that is common to both scenarios—testing the agent on previously unseen problems. In particular, we incrementally evaluate its performance on each subsequent training problem (before providing feedback). The top graph in Figure 7 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 we 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, we 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, we 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

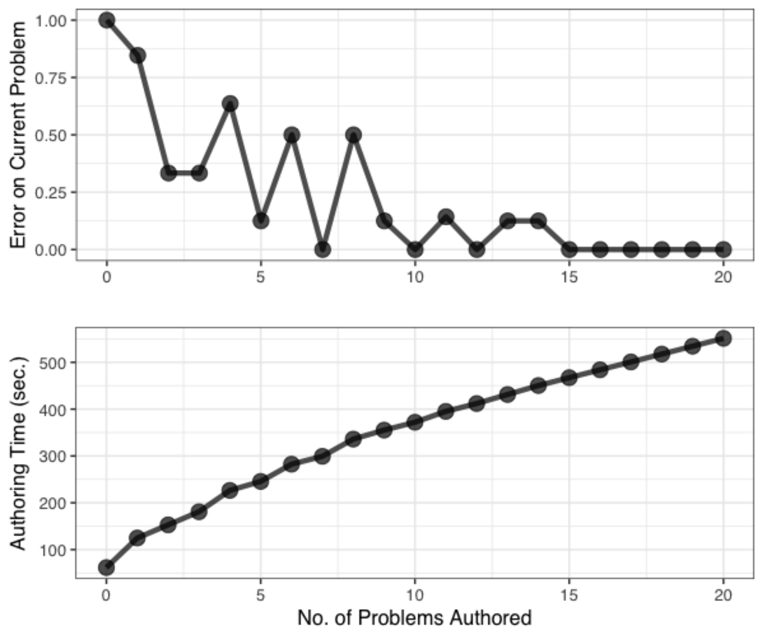


Fig. 7 The average error (top) and cumulative authoring time (bottom) of the DECISION TREE model given the number of experimental design problems authored.

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, we next turned to evaluating its efficiency. While authoring, we tabulated the number of demonstrations and feedback actions that we performed. Using these tabulations and the keystroke-level model from our prior work (MacLellan et al., 2014), we 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,[6] 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 7 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),

[6] 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).

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.

3.4 Discussion

Our 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 provides evidence for the claim that 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. We 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 we will present evidence for this claim in the next section. 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. Although these constant actions are less general than parameterized actions (e.g., copying a value from one field to another), they still support generalization in the conditions. Thus, in the worst case the Decision Tree is at least as general as Example-Tracing, with learned models typically being more general.

Our 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 we 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 we 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 we estimated would take approximately 9.2 minutes for the average trained expert. This learned expert model could also be applied to any novel problem within this interface. Overall this finding supports our claim that expert models can be efficiently authored by training simulated students. Further, this work suggests that authoring the experimental design tutor is more efficient with the Decision Tree model than with Example-Tracing, even when taking mass production into account. This finding extends previous comparisons of apprentice learner models and Example-Tracing (MacLellan et al., 2014) that ignored the mass production capability.

Despite these promising initial results, we 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. This is in contrast to 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, we 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, we 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 and 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 training 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 training 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 larger, requiring more time to author. 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 and Koedinger, 2011), where students could bring their own variables and values.

3.5 Key Findings of the Experimental Design Case Study

The results of our case study suggest that Example-Tracing and tutoring the Decision Tree model are both viable approaches for non-programmers to create tutors. More specifically, we 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.

Our 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—or solution structures, when using mass production—increases because each 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 and Myers, 1999). Another approach would give the agent problems that favor different strategies to encourage variety, similar to tutoring real students (Tenison and MacLellan, 2014).

Our 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. However, for problems that require many complex mass-production templates, our 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 higher ceiling because of its ability to generalize.

This work demonstrates the use of Example-Tracing and the Decision Tree model for authoring a novel experimental design tutor. Our evaluation of these two approaches for this authoring task extends the prior work on authoring tutors with simulated students (MacLellan et al., 2014; Matsuda et al., 2014). In particular, our results show 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, which prior work failed to do. 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 as powerful as hand authoring (high ceiling).

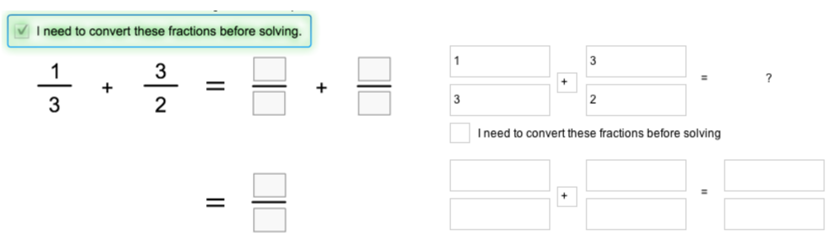
4 Authoring Expert Models Across Domains

The previous section provided a single example of how apprentice learner models can support tutor authoring. In this section, we endeavour to convince the reader that our two preliminary models (the Decision Tree and Trestle models) are general enough to support tutor development across a wide range of tutoring domains, even though they draw on a small, fixed set of prior knowledge. It is impossible to prove that these models will be able to support development of *any* tutor, so we 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. We 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 expands the current literature, which emphasizes results in just one or a few domains.

To support a claim of generality, we 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.[7] 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, we have chosen to author tutoring systems that teach *associations*, *categories*, *skills*, or combinations of these types. Associations involve knowledge that has a constant stimulus and constant response

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


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

(a) (b)

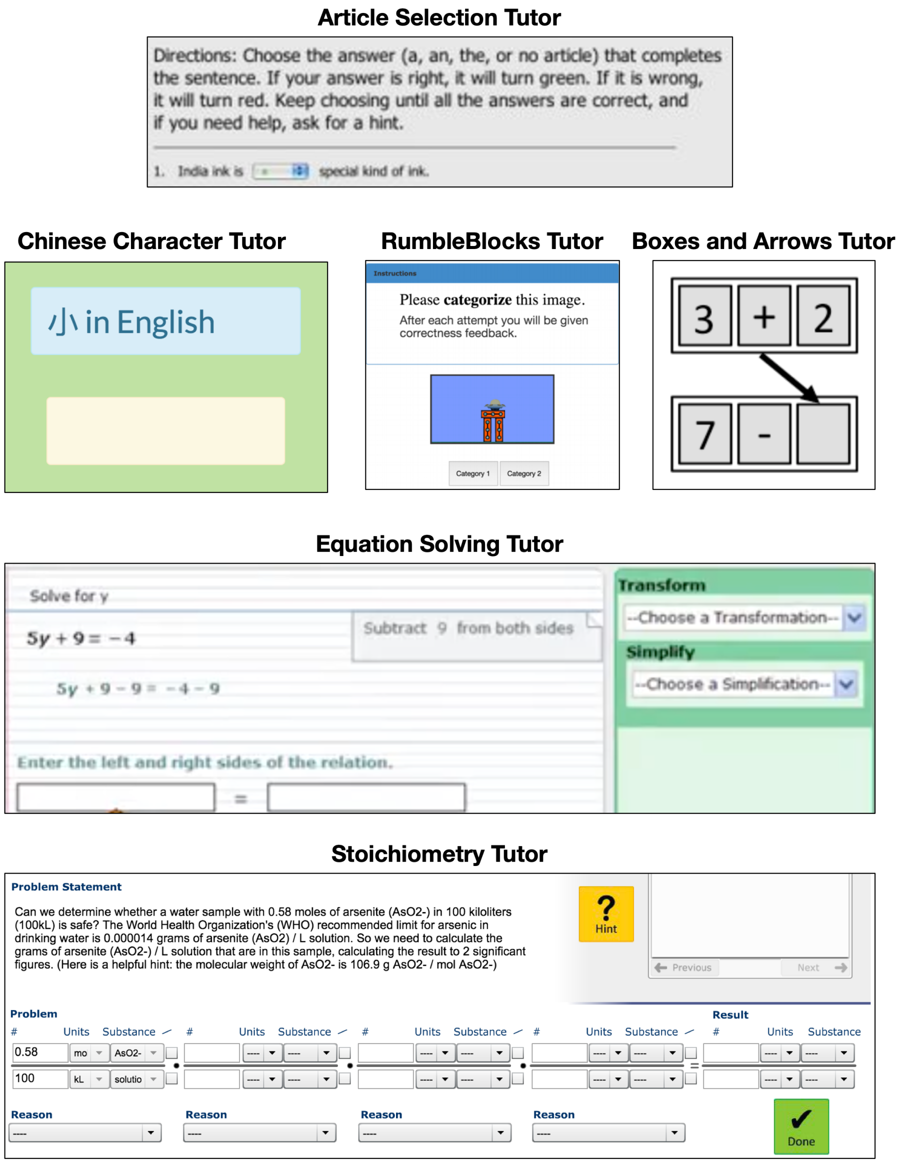
Fig. 8 The fraction arithmetic tutor interface used by the human students (left) and the isomorphic tutor that was recreated using simulated agents (right). If the current fractions need to be converted to a common denominator, then students must check the "I need to convert these fractions before solving" 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).

(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" whenever 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).

4.1 The Seven Tutoring Systems

To demonstrate the generality of our apprentice learner models, we applied them to replicate seven tutoring systems that have human data available in DataShop (Koedinger et al., 2010): the fraction arithmetic tutor (Patel et al., 2016), the Chinese character tutor (Pavlik et al., 2008), the English article selection tutor (Wylie et al., 2009), the RumbleBlocks stability tutor (MacLellan 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. They are also good examples of tutoring systems that have been fielded with real students. Thus, a finding that apprentice learning models support tutor development across all seven tutors is strong evidence for the generality of the models. Towards this end, we start by reviewing each tutoring system in turn.

First, we created a fraction arithmetic tutor, see Figure 8. This tutor presents students with either fraction addition problems with same denominator, fraction addition problems with different denominators, or fraction multiplication problems. In the case of fraction addition with different denominators, it teaches students to convert the fractions into common denominators using cross multiplication (multiplying the two denominators to create a common denominator). Patel et al. (2016) designed this tutor for a classroom experiment to determine whether blocking or interleaving these different types of fraction arithmetic problems produced better learning. This tutor teaches categories for labeling whether conversion is needed and skills for performing the arithmetic



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 special kind of ink.

Please **categorize** this image.
After each attempt you will be given correctness feedback.

Solve for y

$5y + 9 = -4$

Subtract 9 from both sides

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

Enter the left and right sides of the relation.

Problem Statement

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-)

Problem

#	Units	Substance
0.58	mo	AsO2-
100	kL	solutio

Reason ----

Fig. 9 The original interfaces for six additional tutoring systems that have been previously used with real students.

Figure 9 shows the six additional tutoring systems that we replicated. The first of these is the Chinese character tutor (Pavlik et al., 2008), which 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.

The next tutor is the English article selection tutor Wylie et al. (2009), 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).

The third system is the RumbleBlocks stability tutor (MacLellan et al., 2016), which asks students to categorize images of block towers produced in the RumbleBlocks game (Christel et al., 2012). This tutor teaches students the engineering concept of stability. Like the article selection tutor, it supports students' acquisition of category knowledge; e.g., multiple towers (variable stimulus) map to "stable" (constant response).

The fourth tutor is for boxes and arrows problems Lee et al. (2015), which each contains three numbers, two operators ($+$, $-$, $\times$ or $/$), and one arrow that points to the empty field (e.g., $3 + 2 \rightarrow 7 - \_\_$). These rules for solving these problems were designed to be intentionally arbitrary and the primary challenge students face in this tutor is discovering the correct rules given the feedback. There are two types of problems, easy and hard problems, distinguished by the color of the arrow (our descriptions use $\rightarrow$ to denote easy problems and $\Rightarrow$ 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 our 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 $3 + 2 \Rightarrow 7 - \_\_$ is 4 because $7 - 4 = 3$. This tutoring system teaches skills (i.e., variable to variable mappings).

Next is the stoichiometry tutor McLaren et al. (2006), which coaches students on how to do 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).

The last system is the equation solving tutor (Ritter et al., 2007), 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 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. One complication was that the original tutor had problems where students select a transformation 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. There was no straightforward way to replicate this functionality, so we instead just created versions of these problems where the student performed all the actions. This tutor teaches students a combination of both categories (e.g., picking the correct transformation to perform) and skills (e.g., performing the transformations).

4.2 General Authoring Approach

To test if the Decision Tree and Trestle models could support tutor development across these seven tutors and evaluate their efficiency at doing so, we created isomorphic versions of each tutor that our apprentice learning agents could interface with, see Figure 10. We authored each of our tutors using the Cognitive Tutor Authoring Tools (CTAT) (Aleven et al., 2006). We created the interfaces using CTAT's drag-and-drop interface builder and authored behavior graphs for each tutor using Example-Tracing (Aleven et al., 2009). When authoring these interfaces and behavior graphs, we did our best to maintain a close alignment to the original tutor designs, but this was not always possible and a number of discrepancies arose. Some differences were due to a lack of access to the original tutors or screenshots of them. We often had to reverse engineer the tutors' behavior using only the available DataShop log data and textual descriptions of the original tutors as a guide. This was the case for the Chinese character tutor, where we can only guess 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 9.

Authoring these tutors using Example-Tracing provided a baseline that we could evaluate our two apprentice learning models against. Additionally, we were able to use the behavior graphs produced by Example-Tracing to automate the process of interactively training each apprentice learning agent. Thus, rather than manually training each apprentice learning agent for our efficiency evaluation, we used the Example-Tracing tutor to interactively provide demonstrations and feedback to our agents. This approach enabled us to evaluate the efficiency of both the Decision Tree and Trestle models and to compare their authoring efficiency to that of Example-Tracing. Despite this

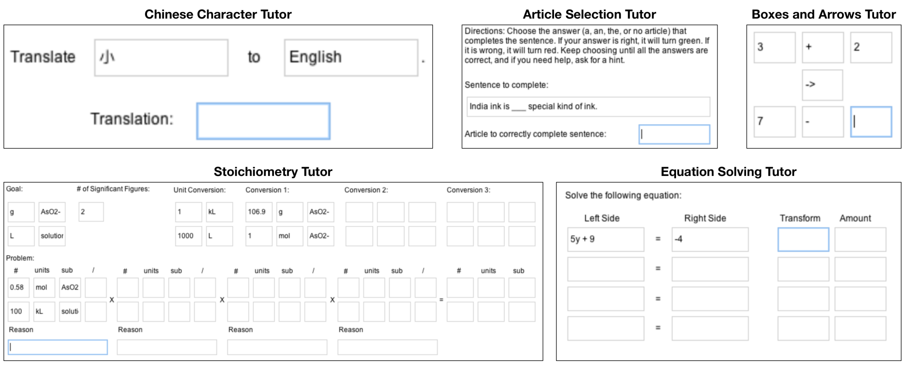

Chinese Character Tutor
Translate
小
to
English
Translation:
Article Selection Tutor
Directions: Choose the answer (a, an, the, or no article) that completes the sentence. If your answer is right, it will turn green. If it is wrong, it will turn red. Keep choosing until all the answers are correct, and if you need help, ask for a hint.
Sentence to complete:
India ink is ___ special kind of ink.
Article to correctly complete sentence:
Boxes and Arrows Tutor
3
+
2
->
7
-
Stoichiometry Tutor
Goal:
g
AsO2-
L
solution
# of Significant Figures:
2
Unit Conversion:
1
kL
1000
L
Conversion 1:
106.9
g
AsO2-
1
mol
AsO2-
Conversion 2:
Conversion 3:
Problem:
# units sub /
0.58 mol AsO2
100 kL soluti
x
=
Reason
Equation Solving Tutor
Solve the following equation:
Left Side
Right Side
Transform
Amount
5y + 9
=
-4

Fig. 10 Isomorphic CTAT Interfaces for five of the seven tutors. The fraction arithmetic tutor is shown in Figure 8. Additionally, the RumbleBlocks tutor was very simple, consisting of a single input (the block configuration) and a single binary output (stable or not), so we created a script that simulated the tutor interface without actually authoring one using CTAT.

automated approach, our models still support interactive training with human authors and we other research efforts that are currently investigating real end-users' experiences with authoring tutors using apprentice learner models (e.g., Weitekamp et al., 2020).

Given that every interactive authoring session could take a non-deterministic trajectory,[8] we authored each tutoring system multiple times to get a better sense of the agents' overall performance. For these multiple runs, we conducted a separate authoring simulation for each sequence that one of the human student who used the original version of the tutor received. This approach produced reasonable variety in the training sequences and had the added benefit that it let us directly compare the learning performance of our agents to that of the human students that used the tutor.

4.3 Model Evaluation

4.3.1 Overall and Asymptotic Performance

To assess the tutor development capabilities of these models across the seven tutor domains, we first looked at the overall and asymptotic performance of the two models. Additionally, we computed the overall and asymptotic 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 11.

To compute the overall accuracy for each model and for the humans, we took the average performance across all tutored steps. For this calculation, a correct step was counted as an accuracy of 1 and an incorrect step or a

[8] Given the same sequence of instruction problems, the agents might take different actions and learn different knowledge as a result

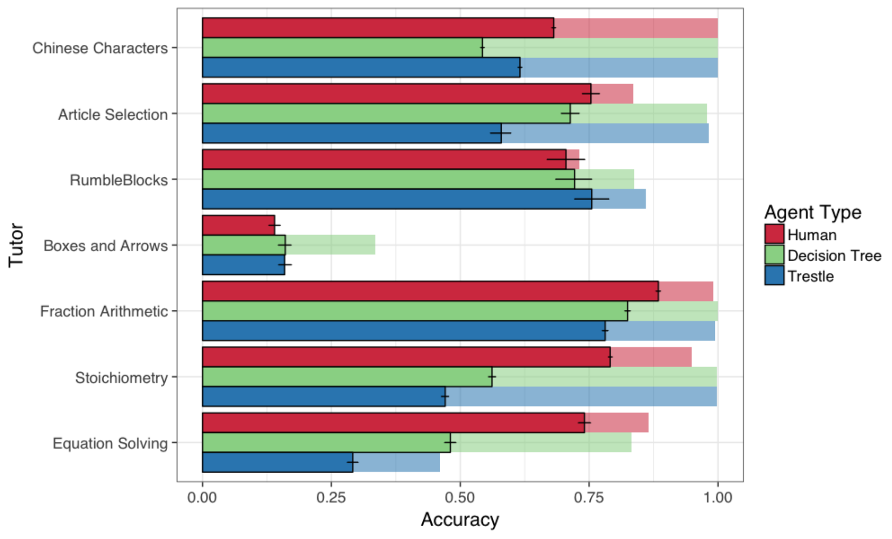


Fig. 11 The overall (opaque) and asymptotic (semi-transparent) accuracy for each type of agent in each tutor. The 95% confidence intervals are shown for the overall accuracy. The asymptotic accuracy was computed by fitting a mixed-effect regression model to each data set and using it to predict the performance 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 agents and skills).

hint request was counted as a 0. In all cases, the step was evaluated prior to providing the demonstrations/feedback to the model or human, so all evaluations are on previously unseen data. One limitation of using overall accuracy as a metric, is it does not provide a picture of the models accuracy towards the end of training; it averages over earlier practice, which typically has lower accuracy, and later practice, which typically has higher accuracy.

To overcome this limitation we computed the asymptotic performance for each model (i.e., what is the average performance of the model at the end of training). To compute asymptotic performance, we 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), we computed the prior number of opportunities a student had to exercise the same skill. Using these values, we 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. This model, which is represented using the R lme4 formula correctness $\sim$ opportunity + (1 + opportunity|skill) + (1|student), is similar to a repeated measures ANOVA and analogous to the Additive Factors Model (Cen, 2009) used for learning curve analysis.

For each data set, we used the fixed-effects estimates for the intercept and opportunity to predict performance for the average agent 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 11 as semi-transparent bars, represent the performance for the average agent and skill at the end of practice in each tutor for each agent type. It is worth noting that we chose a threshold of 95% because some skills were practiced by some 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 human students probably bring more prior knowledge in these tutors. For example, the human students that used the equation solving tutor (which teaches two-step linear equation solving) had all previously completed a unit on one-step linear equations. Additionally, these tutors have the biggest differences between the original and replicated versions of the tutors. 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. It is worth noting that in many cases these prior models were trained with an easier subset of the problems than we explore here.

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, we find that the Trestle model does best . 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. 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 in terms of overall error over the course of training, by the end of training they are able to achieve human-level performance or better in all but the equation solving tutor.

4.3.2 Expert-Model Authoring Efficiency

We next analyzed the efficiency of the two models for the purposes of authoring, similar to the efficiency analysis used in our case study of an experimental design tutor. For this analysis, we tabulated 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, we were 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 tutoring systems was built using Example-Tracing, so it was straightforward to evaluate the efficiency of this alternative approach for comparison purposes by tabulating the number of links in the authored behavior graphs.

When analyzing the efficiency of these authoring techniques, we used the approach from the previous section. In particular, we used the Keystroke-Level Model (KLM) from our 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.[9] 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, we assume that mass production steps take *zero* seconds.

When building the seven tutors using Example-Tracing, we authored the following tutor content:

- **Chinese character:** 586 behavior graphs (one mass production template with 2 demonstration links);
- **Article selection:** 84 graphs (one template with two demo links);
- **Boxes and arrows:** 64 graphs (two templates, each with two demo links);
- **Fraction arithmetic:** 84 graphs (three templates with 22 demo links);
- **Stoichiometry:** 16 graphs (eight templates containing 897 demo links and 30 unordered action groups); and
- **Equation solving:** 1357 graphs (six templates with 53 demo links and 24 unordered action groups).

Additionally, we did not create a new RumbleBlocks tutor, but for the current analysis, we estimated that it would require 139 behavior graphs (two

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

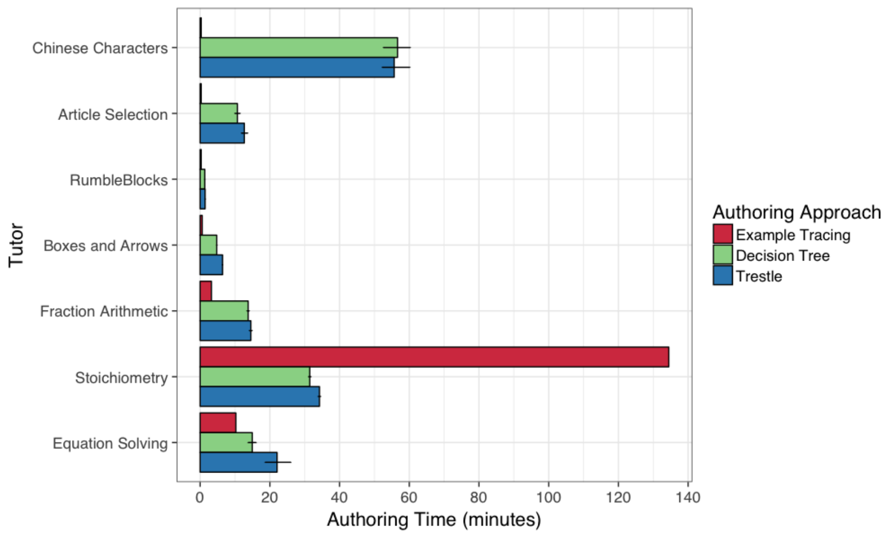


Fig. 12 The estimated amount of time (in minutes) it would take the average trained author to build an expert model for each of the seven tutors using either Example-Tracing or one of the two simulated student models. Both simulated agent approaches are shown with 95% confidence intervals, where the variation is due to differences in the number of interactions for each model across each student training sequence. 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).

templates containing two demo links). We 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.

To estimate the time needed for each simulated student approaches, we tabulated the total amount of examples and feedback that were provided to all of the simulated students across all of the simulations. Using the KLM, we estimated the total amount of time needed to train each agent. We averaged these estimates to generate the results shown in Figure 12.

4.4 Discussion

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 the previous section and previous work (MacLellan et al., 2014) might suggest. A closer inspection of stoichiometry tutor shows that each of the behavior graphs in this tutor are complex, requiring hundreds of demonstrations, and most problems have different behavior graph structure, so they required separate mass production templates. These characteristics make this tutor similar to the experimental design tutor, which we 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., that have 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 apprentice learner models 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 our simulations, we made many mistakes and it took us substantially longer to author these tutors. For example, authoring the behavior graphs using the Example-Tracing approach for the stoichiometry tutor alone took us at least three full eight-hour days of authoring, and this was after a week of designing the interface and planning the appropriate behavior graph structure. When authoring more complicated graphs, we often made mistakes that would not be apparent until the end, and we would have to re-author the entire graph.[10] In contrast, authoring mistakes will harm the simulated agents' learning, but the simulated agents 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). However, this claim remains to be tested.

The KLM also fails to take into account preparation and planning time. In particular, we found it challenging to author behavior graph templates (with variables rather than specific values) directly. Typically, before authoring a template, we would author a complete behavior graph for one or two specific problems, and then think about how we could author a template that would generalize these problems. Ignoring the time spent planning, if we added the demonstrations needed for these preliminary behavior graphs to our estimates, it would double or triple the authoring time. In contrast, training the simulated agents only required us to provide demonstrations and feedback on specific problems, which required us to do 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 steps take zero seconds, an unrealistic assumption. We 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 as-

[10] 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.

sumption 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 cells in an Excel spreadsheet, which we 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 11) 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 apprentice learner models to date, replicating the findings of the previous sections 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.

4.5 Key Findings of the Cross-Domain Study

The results of this section further support our high-level claims. 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.

Second, they indicate that training simulated agents is an efficient authoring approach comparable to Example-Tracing with mass production. 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. In the other cases, authoring with Example-Tracing seems to be more efficient; however, the current analysis ignores the cost of mass production. More work is needed to better quantify the cost of mass productions so it can be more appropriately compared with apprentice learning models.

In conclusion, this section demonstrates the general capabilities of our 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 section support our overarching claim that apprentice learner models are general-purpose tools capable of supporting tutor development.

5 Conclusions and Future Work

In this paper, we explore the use of apprentice learning models, or computer models that learn from examples and feedback, for supporting tutor development. To support these investigations, we presented the Apprentice Learner Architecture, described its application for authoring a novel experimental design tutor, and analyzed its ability to author seven additional tutors across a wide range of domains. Across this work, we have endeavoured to convince the reader of two main claims. In particular, that apprentice learner models (1) support efficient expert-model authoring and (2) are domain-general tools. This section reviews each claim, the evidence to support them, and discusses limitations and directions for future work related to each claim.

Support Efficient Expert-Model Authoring

First, we 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, our case study in authoring an experimental design tutor showed that 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. Our subsequent cross domain analysis, 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 additional 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. Even though we found that Example Tracing are typically more efficient than apprentice learner models, the current work suggests that authoring with apprentice models is generally comparable to that of Example Tracing and shows great promise as a tool for scaling up tutor authoring for non-programmers.

Finally, there were some limitations of the KLM we used to estimate the authoring efficiency of each approach. Specifically, the KLM estimates it should take on the order of minutes to author many of the tutors in our study, but our qualitative experience actually authoring these tutors suggests that it takes much longer (hours to days). Based on our experience, the main reason for the discrepancy is that the KLM assume the human author is trained (knows exactly what steps to do next) and is error free, which are two very strong assumptions. We argue that our KLM is a useful tool for comparing different authoring approaches, as it highlights when one authoring approach might be preferable to another (e.g., example tracing ¿ apprentice learner models for tutors with simple problem spaces). However, future work should explore how to account for errors and error correction during the authoring process within the KLM estimate.

Are Domain-General Tools

Our second high-level claim is that these models are domain general and can support learning of multiple knowledge types across a wide range of domains, even though they draw on a small, fixed set of prior knowledge. Our case study and cross domain evaluation provides the main evidence to support this claim by showing that apprentice learner models support authoring across eight different tutors (experimental design and the seven other 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 what we explicitly demonstrated. One limitation of the current work is that it focuses on modeling learning in tutoring systems, rather than educational technologies more broadly. This broader class of educational technologies comes with additional challenges, such as realtime interaction (as opposed to step-based interaction in tutors), delayed feedback (opposed to immediate feedback in tutors), and a general increase in complexity. Future work should explore how to extend the current models beyond tutors to other learning environments, such as educational games (see Harpstead, 2017), inquiry learning, and programming environments.

In conclusion, this paper makes contributions to the fields of human-computer interaction, artificial intelligence, and intelligent tutoring systems. It demonstrates meaningful progress towards a domain-general, machine-learning powered tutor authoring tool (the Apprentice Learner Architecture), that lets non-programmers author tutors by providing examples and feedback—similar to how they would teach a human student. We show that this tool can support efficient expert model authoring across a wide range of tutors and domains, without the need to hand-author specialized knowledge for these domains. Our hope is that this new tool can lay the foundation for subsequent research to support teachers and instructional designers in creating pedagogically effective educational technologies at scale.

Acknowledgements This work was started while I was a PhD student at Carnegie Mellon University and I continued to develop these ideas while working at Soar Technology, Inc. and at Drexel University (my current institution). This work would not have been possible without the members of my thesis dissertation committee (Ken Koedinger, Pat Langley, Vincent Aleven, and John Anderson) who provided me with guidance and edited much of the text that appears in this paper. Additionally, some excerpts of the text describing the experimental design tutor was pulled from a workshop paper published in collaboration with Erik Harpstead, Eliane Stampfer Wiese, Mengfan Zou, Noboru Matsuda, Vincent Aleven, and Ken Koedinger (MacLellan et al., 2015). Lastly, this work would not have been possible without Mengfan Zou, who built the first version of the experimental design tutor as part of her undergraduate research experience with me. Thank you everyone for all the help you provided.

References

Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the 21st international conference on Machine learning, pp 1–8 2

Aleven V, McLaren BM, Sewall J, Koedinger KR (2006) The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. In: Ikeda M, Ashley KD, Tak-Wai C (eds) Proceedings of the 8th International Conference on Intelligent Tutoring Systems, Springer, pp 61–70 1, 4.2

Aleven V, McLaren BM, Sewall J, Koedinger KR (2009) A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. International Journal of Artificial Intelligence in Education 19:105–154 1, 3.1, 3.5, 4.2

Barnes T, Stamper JC, Lehmann L, Croy MJ (2008) A pilot study on logic proof tutoring using hints generated from historical student data. In: Proceedings of the 1st International Conference on Educational Data Mining, pp 197–201 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 4.3.1

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

Bowen WG, Chingos MM, Lack KA, Nygren TI (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 2.1

Cen H (2009) Generalized Learning Factors Analysis: Improving Cognitive Models with Machine Learning. PhD thesis, Carnegie Mellon University 4.3.1

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 3.1, 3.2

Christel MG, Stevens SM, Maher BS, Brice S, Champer M, Jayapalan L, Chen Q, Jin J, Hausmann D, Bastida N, Zhang X, Aleven V, Koedinger KR, Chase C, Harpstead E, 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 4.1

Clark RE, Feldon DF, van Merriënboer JJG, Yates K, Early S (2008) Cognitive task analysis. In: Spector JM, Sosman MG, van Merriënboer MD, Driscoll MP (eds) Handbook of Research on Educational Communications and Technology, International Journal of Educational Research, Mahwah, NJ, chap 8, pp 578–591 1

Collins A, Brown JS, Newman SE (1987) Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics. Tech. Rep. 403, National Institute of Education, Washington, DC 1

Dejong G, Mooney R (1986) Explanation-based learning: An alternative view. Machine Learning 1:145–176 2.3

Dent L, Boticario J, McDermott JP, Mitchell TM (1992) A personal learning apprentice. In: Proceedings of the 10th National Conference on Artificial Intelligence, pp 96–103 2

Fikes RE, Hart P, Nilsson NJ (1972) Learning and Executing Generalized Robot Plans. Artificial Intelligence 3:251–288 2.1

Fisher DH (1987) Knowledge Acquisition Via Incremental Conceptual Clustering. Machine Learning 2:139–172 2.3

Gobert JD, Koedinger KR (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 3.4

Graesser AC, VanLehn K, Rose C, Jordan PW, 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. PhD thesis, Carnegie Mellon University 5

Jarvis MP, Nuzzo-Jones G, Heffernan NT (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 LP, Littman ML, Moore AW (1996) Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4:237–285 3.3

Koedinger KR, Anderson JR (1997) Intelligent Tutoring Goes To School in the Big City. International Journal of Artificial Intelligence in Education 8:1–14 (document), 1

Koedinger KR, Baker RSJd, Cunningham K, Skogsholm A, Leber B, Stamper J (2010) A Data Repository for the EDM community: The PSLC DataShop. In: Romero C, Ventura S, Pechenizkiy M, Baker RSJd (eds) Handbook of Educational Data Mining, CRC Press, Boca Raton 2.4, 4.1

Koedinger KR, Corbett AT, 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, 4, 4.3.1

Koedinger KR, Booth JL, Klahr D (2013) Instructional Complexity and the Science to Constrain It. Science 342(6161):935–937 1

Kumar R, Roy ME, Roberts RB, Makhoul JI (2014) Towards automatically building tutor models using multiple behavior demonstrations. In: International Conference on Intelligent Tutoring Systems, Springer, pp 535–544 1

Langley P, Ohlsson S (1984) Automated cognitive modeling. In: Proceedings of the 4th National Conference on Artificial Intelligence, pp 193–197 2

Lee HS, Betts S, Anderson JR (2015) Learning Problem-Solving Rules as Search Through a Hypothesis Space. Cognitive Science 40(5):1036–1079 4.1, 4.1

Li N (2013) Integrating Representation Learning and Skill Learning in a Human-Like Intelligent Agent. PhD thesis, Carnegie Mellon University 1, 4.3.1

Li N, Schreiber AJ, Cohen WW, Koedinger KR (2012) Efficient Complex Skill Acquisition Through Representation Learning. Advances in Cognitive Systems 2:149–166 2.4

Li N, Stampfer E, Cohen WW, Koedinger KR (2013) General and Efficient Cognitive Model Discovery Using a Simulated Student. In: Knauff M, Paulen M, Sebanz N, Wachsmuth I (eds) Proceedings of the 35th Annual Meeting of the Cognitive Science Society 1

Li N, Matsuda N, Cohen WW, Koedinger KR (2014) Integrating representation learning and skill learning in a human-like intelligent agent. Artificial Intelligence 219:67–91 1, 2, 2.1

MacLellan CJ, Koedinger KR, Matsuda N (2014) Authoring Tutors with SimStudent: An Evaluation of Efficiency and Model Quality. In: Trausen-Matu S, Boyer K (eds) Proceedings of the 8th International Conference on Intelligent Tutoring Systems 3, 3.2, 3.3, 3.4, 3.5, 4.3.2, 12, 4.4

MacLellan CJ, Harpstead E, Wiese ES, 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 1, 5

MacLellan CJ, Harpstead E, Aleven V, Koedinger KR (2016) TRESTLE: A Model of Concept Formation in Structured Domains. Advances in Cognitive Systems 4:131–150 2.3, 2.4, 4.1, 4.1

Matsuda N, Cohen WW, Koedinger KR (2014) 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, 3, 3.5

McDaniel RG, Myers BA (1999) Getting more out of programming-by-demonstration. In: Proceedings of the human factors in computing systems conference, pp 442–449 3.5

McLaren BM, Koedinger KR, Schneider M, Harrer A, Bollen L (2004) Bootstrapping novice data: Semi-automated tutor authoring using student log files. In: Proceedings of the Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes, Seventh International Conference on Intelligent Tutoring Systems 1

McLaren BM, Lim SJ, Gagnon F, Yaron D, Koedinger KR (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 4.1, 4.1

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

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, Kluwer Academic Publishers, Netherlands, pp 493–546 (document), 1

Murray T (2005) Having It All, Maybe: Design Tradeoffs in ITS Authoring Tools. In: Proceedings of the 3rd International Conference on Intelligent Tutoring Systems, Springer, Berlin, Heidelberg, pp 93–101 1

Myers B, Hudson SE, Pausch R (2000) Past, present, and future of user interface software tools. ACM Transactions on Computer-Human Interaction 7:3–28 3.1, 3.5

Nathan M, Koedinger KR, 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

Ohlsson S (2011) Deep Learning. How the Mind Overrides Experience, Cambridge University Press, New York 2

Pane JF, Griffin BA, McCaffrey DF, Karam R (2013) Effectiveness of Cognitive Tutor Algebra I at Scale. Tech. Rep. WR-984-DEIES, RAND Corporation, Santa Monica (document)

Patel R, Liu R, Koedinger KR (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 4.1, 4.1

Pavlik PI, Bolster T, Wu Sm, 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 4.1, 4.1

Pavlik PI, Kelly C, Maass JK (2016) The Mobile Fact and Concept Training System (MoFaCTS). In: Proceedings of the 12th International Conference on Intelligent Tutoring Systems, pp 247–253 4.2

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12:2825–2830 2.3, 2.4

Quinlan JR (1986) Induction of decision trees. Machine Learning 1:81–106 2.4

Ritter S, Anderson JR, Koedinger KR, Corbett AT (2007) Cognitive Tutor: Applied research in mathematics education. Psychonomic Bulletin & Review 14(2):249–255 1, 2.4, 4.1, 4.1

Sao Pedro MA, Gobert JD, Heffernan NT, Beck JE (2009) Comparing Pedagogical Approaches for Teaching the Control of Variables Strategy. In: Taatgen N, van Rijn H (eds) Proceedings of the 31st Annual Conference of the Cognitive Science Society, pp 1–6 3.1

Schneider M, Rittle-Johnson B, Star JR (2011) Relations among conceptual knowledge, procedural knowledge, and procedural flexibility in two samples differing in prior knowledge. Developmental Psychology 47(6):1525–1538 3.4

Sottilare RA, Holden HK (2013) Motivations for a Generalized Intelligent Framework for Tutoring (GIFT) for Authoring, Instruction, and Analysis. In: Sottilare RA, Holden HK (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

Tenison C, MacLellan CJ (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 3.4, 3.5

Ur S, VanLehn K (1995) Steps: A Simulated, Tutorable Physics Student. Journal of Artificial Intelligence in Education 6:405–437 2, 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 RM, Chi MTH (1991) Modeling the self-explanation effect with Cascade 3. In: Proceedings of the human factors in computing systems conference, pp 132–137 2, 2.1

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

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 3.4

Weitekamp D, Harpstead E, Koedinger KR (2020) An interaction design for machine teaching to develop ai tutors. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp 1–11 4.2

Wylie R, Koedinger KR, 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 4.1, 4.1