Research Statement

Intelligent Tutoring Systems have been shown to substantially improve students' learning outcomes. For example, studies have shown that students double their learning gains when using tutoring systems rather than traditional instruction (Pane, 2013), and, in some cases, students achieve more learning in half the time (Lovett et al., 2013). If tutoring systems are so effective, then why are they not more commonly used in learning environments? One key problem is that *effective* intelligent tutoring systems are extremely time consuming to design and build, making them hard to scale. Creating one hour of tutoring system instruction often takes 200-300 hours of development, and the developers need expertise in multiple disciplines, including programming, cognitive modeling, and the target domain. Furthermore, designing tutoring systems that are effective (e.g., that yield twice the learning) is challenging. Researchers need to conduct cognitive task analyses of domain experts, A/B tests to explore instructional design variations, and pre-posttest studies to ensure that the designed tutors are achieving the desired learning outcomes. All of these steps require time, expertise, and resources that ultimately make tutor development a costly endeavour.

To overcome these barriers, my research explores the development of computational models that simulate students learning from tutoring. Instructional designers can use these models to author tutoring systems as if teaching a human, rather than by programming, and my research shows that these models can reduce authoring time by half when compared to another state-of-the-art authoring-by-demonstration approach. Once a tutor has been built, the same models can be used to simulate students interacting with the tutor in order to determine if the tutor is achieving the desired pedagogical outcome. Additionally, these tutors, once operationalized, generate large amounts of human-learning data, so my research also explores the application of educational data mining techniques to tutor data to test and improve computational models of learning and the theories underlying them. My research shows that simulated agents can successfully predict the results of tutor A/B experiments and that data from these experiments can be used to discover more human-like models. The goal of my research program is to develop a Model Human Learner—similar to Card, Moran, and Newell's (1986) Model Human Processor—that encapsulates psychological and learning science findings in a format that researchers and instructional designers can use to reliably create effective tutoring systems. Ultimately, I envision a platform where teachers and domain experts can author tutors by interactively teaching simulated agents and where classroom studies can be simulated to evaluate tutoring system designs.

A Computational Theory of Apprentice Learning

At the core of my research program is a theory of the computations that any agent (human or otherwise) needs to perform in order to learn from worked examples and feedback, which, in the spirit of Marr (1983), I refer to as a computational theory of apprentice learning. This theory unifies multiple lines of research in the areas of expert systems, explanation-based learning, inductive logic programming, inverse reinforcement learning, and traditional reinforcement learning to support the induction and use of mixed symbolic and probabilistic structures. Since tutoring systems are designed to support apprentice learning (VanLehn, 2006), having a detailed understanding of the mechanisms underlying this format of learning is fundamental to building effective tutors. My preliminary theory posits three separate mechanisms that agents use to perform apprentice learning: how-learning, where-learning, and when-learning. According to this theory, when an agent is faced with a problem to solve (e.g., what is 2+3?), a match is made between learned skills and the current problem state. If any skills match, the agent executes one with high expected reward. If no skills apply (a typical initial response), then the agent requests a demonstration, or bottom-out hint, from the tutor (e.g. the tutor might enter a 5 in the answer field). The agent then performs how-learning 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 where-learning to induce a schema, or pattern, for extracting information from the environment necessary to execute the learned procedure and for recognizing when the procedure should be considered; for example, an agent would learn patterns for extracting the first and second numbers from the tutor interface when there is an operator sign between them. Finally, the agent uses when-learning to estimate a reward function for the skill, which enables it to prioritize matching skills and determine which should be applied in any given situation; for example, the agent might learn that applying the add skill to two numbers produces a high reward when the operator between them is a plus sign. The output of these steps (a procedure, a schema, and a reward function) constitute a new skill. On subsequent problems, the agent attempts to apply learned skills that it estimates will produce high reward and receives correctness feedback. This feedback is then used in where- and when-learning to refine the skill's schema and reward function.

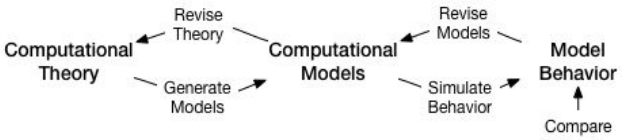

Figure 1: My Theory-Driven Research Approach

Based on this theory, I created the *Apprentice Learner Architecture* [1], which affords the comparison of alternative computational models of apprentice learning. Within the architecture, a model presents as a set of algorithms to solve the computational tasks outlined by the theory. For example, *SimStudent* (Matsuda et al., 2015; Li et al., 2015), my initial model of learning, uses an iterative deepening search for how-learning, the version space algorithm for where-learning, and a decision tree algorithm for when-learning. Unlike statistical learner models that fit functions (e.g., a power or logistic function) to performance data, apprentice learner models actually learn to perform the task being modeled—they induce an expert model from the demonstrations and feedback. Thus, apprentice learner models do not require student data to learn from; instead, they leverage the underlying learning theory to make specific predictions about how students will behave given different instruction. In general, the architecture, and the underlying theory, defines a space of models that can be searched using different model evaluation criteria, and my overarching research approach generates and tests models within this space, and, where necessary refines the theories underlying the models (see Figure 1). In order to support tutor development, my current research explores two evaluation criteria: fit to human data and learning efficiency.

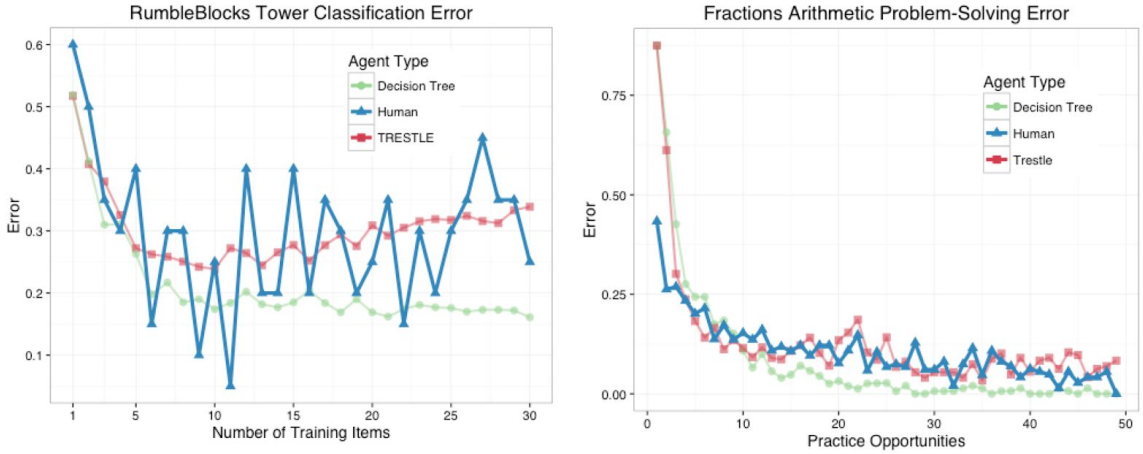

Figure 2: Learning curves for RumbleBlocks (left) and Fraction Arithmetic (right) generated by models that use either Trestle or the Decision Tree algorithm for when-learning.

Discovering Models that Fit Human Data

In order to support the design and testing phases of tutor development, one line of inquiry searches for models of learning that exhibit human-like behavior. The goal is to generate a learning agent that can be used as a "crash test dummy" for testing out different instructional designs to find out which designs are most effective for learning prior to more expensive classroom deployment. Similar to how civil engineers use Computer-Aided Design software to simulate and test bridge designs, instructional designers and researchers can use an apprentice learner model to simulate either a single student or an entire classroom interacting with a tutoring system. The data generated from these simulations has a format identical to the data generated from real students using tutors (i.e., tutor transaction log files), so it can be used with existing analysis tools and techniques as a substitute for classroom study data. In this way, apprentice learner models are cost effective tools for testing multiple tutoring system designs.

To evaluate SimStudent as an initial model of human learning and to begin the search for more human-like models, I created a SimStudent agent for every student in an existing fractions tutoring system dataset and used these agents to simulate behavior within the tutoring systems. Next, I compared the simulated behavior with the human behavior to assess agreement. I found that SimStudent is capable of predicting the main experimental effects of the classroom study that generated the human data. In particular, both SimStudent and human agents had better performance in the tutors that presented problems in a blocked, rather than interleaved order. After training in the tutors, I also gave SimStudent agents the same posttest that the humans took and found that SimStudent agents are able to predict a reversal effect—both SimStudent agents and humans had better posttest performance when they used a tutor that presented problems in an interleaved, rather than blocked order. These promising results demonstrate how simulated data might be used as a substitute for actual classroom data; however, when further investigating the step-level data, I found an important qualitative differences between the SimStudent agents and humans. In particular, SimStudent agents achieved greater asymptotic performance.

I hypothesized that this difference was because the decision tree algorithm, used by SimStudent for when-learning, has a perfect memory of all prior examples and trains on all these examples in batches after each new example is presented. To further investigate, I created *Trestle* [2], an incremental categorization algorithm that models forgetting via interference. As a targeted test of this new when-learning approach, I had humans categorize the stability of towers of blocks generated in RumbleBlocks, an educational game that teaches K-3 students engineering concepts. Next, I used two apprentice learner models (one using Trestle and one using Decision Trees) to perform the same categorization task, and I found that Trestle better approximated human asymptotic performance than the Decision Tree algorithm (see the left graph in Figure 2). I then returned to the Fractions Tutor and replicated my earlier simulations to compare two variations of the SimStudent model that used either Trestle or the Decision Tree algorithm for when-learning. My results replicated the RumbleBlocks findings—the models using Trestle better fit the human data than the models using the Decision Tree approach (see the right graph in Figure 2). These results suggest agents that take key characteristics of human learning into account, such as its incremental and stochastic nature, better model behavior than agents that ignore them. These initial studies also show how my theory-driven approach can be used to structure the search for more human-like models, and I aim to apply this approach to test alternative approaches for the other learning mechanisms (where and how) and to test alternative theories (e.g., merging where-learning and when-learning). Ultimately, I would like to

discover apprentice learner models that are consistent with the large amounts of educational data available in public repositories, such as DataShop (see https://pslcdatashop.web.cmu.edu/).

Discovering Efficient Models of Learning

Building tutors takes hundreds of hours per each hour of instruction and requires multiple kinds of expertise, so my second line of work searches for models that learn as efficiently as possible, and I explore how these models can be used by non-programmers to support the design and build phases of tutor development. Like humans, apprentice learner models can be taught by domain experts or instructional designers how to perform domain tasks through worked examples and feedback. Apprentice learner models translate this instruction into expert models that can be used to power a tutoring system or, more generally, to emulate expert behavior on a task. Thus, they provide a means for non-programmers to build expert systems and intelligent tutors. Additionally, this interactive authoring process serves as a form of cognitive task analysis. If apprentice learner models are unable to learn the target skills, then it suggest that the author is leaving out key steps that are necessary for learning. Further, a learned expert model is, in fact, a decomposition of the domain into its target skills, which is the target output of a cognitive task analysis. For the purposes of authoring, efficient models of learning are desirable because they require less training from an instructional designer to learn the target skills. To discover efficient models of learning, I use a similar approach to the one used to find human-like models: I simulate the behavior of alternative models in multiple tutors and identify which models have better performance. For example, in the RumbleBlocks and Fractions Tutor domains shown in Figure 2, the model that uses the Decision Tree algorithm has better performance than the Trestle model, so it would be preferable for authoring purposes.

Having identified a more efficient model, I then explored how this model can be used to facilitate the tutor building process [3, 4]. I used a keystroke-level model to estimate the time it would take a trained author to build an expert model for a twenty-problem Algebra tutor using either SimStudent or Example Tracing, a widely used authoring-by-demonstration approach. My analysis shows that SimStudent cuts the overall authoring time by half, and that the difference between the two approaches grows as the number of problems authored increases (see Figure 3). SimStudent is faster than Example Tracing because it generalizes from previous examples and requires less demonstrations over time, requiring only correctness feedback, which is faster for an author to provide. Also, the learned SimStudent model is general, so if we wanted the tutor to support 20 additional problems then little, or even no, additional authoring would be required. In contrast, Example Tracing models are specific to the problems they are authored with, so the author would need to duplicate the authoring process with the 20 new problems. These results demonstrate the use of apprentice learner models for supporting tutor design by making the authoring process more efficient. It is my hope that by lowering the barrier to tutor authoring, more teachers and domain experts will be able to build tutors—ultimately improving student learning outcomes at an unprecedented scale.

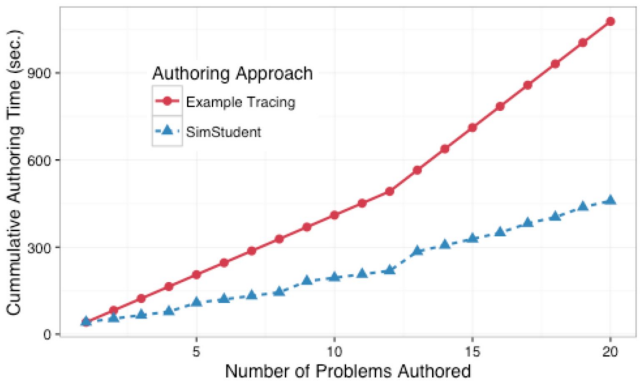

Figure 3: Cumulative authoring time for a twenty-problem Algebra Tutor, as estimated by a keystroke-level model.

Future Work

My initial results have been encouraging, but more work is needed to explore the use of apprentice learner models for scaling-up expert system and intelligent tutor development. If tutoring systems are easier to develop, then they should become more widely adopted in K12 classrooms, MOOCs, and college classes, and as adoption increases these tutors will generate human-learning data that can be used to guide the search for better models of human learning. In support of this goal, my recent work at Soar Technology has explored how to make teaching of apprentice learner models a more seamless and well-supported process for end users, such as teachers. For example, my recent MineApprentice project (ONR) applies apprentice learning to acquire mine countermeasures planning models from examples and feedback provided directly by naval operators in their planning interfaces. A major outcome of this work has been the development of the *Natural Training Interactions (NTI) framework* [5], which maps out the space of interaction design patterns for non-technical users to teach interactive machine learning systems, and the *NTI testbed* [6], which provides a methodology of rapidly testing learning system designs to identify which are more natural and efficient for users.

Moving forward, I aim to extend this work to a wider range of complex tasks, such as those found in educational games [7], experimental design [8], conceptual design [9], and computer programming [10], leveraging the insights and data gained in each task to guide development of the underlying apprentice learning theory as well as extend human-computer interaction theory to support the design of interactive learning systems that are natural for people to teach and use. By investigating how a single computational theory might account for learning across all these tasks, I hope to contribute a unified theory of learning—motivated by Newell's (1990) unified theories of cognition—to the learning sciences, and more generally the cognitive and computational sciences.

References

1. **MacLellan, C.J.**, Harpstead, E., Patel, R., Koedinger, K.R. (2016). The Apprentice Learner Architecture: Closing the loop between learning theory and educational data. In *Proceedings of the 9th International Conference on Educational Data Mining* (pp. 151-158). Raleigh: International Educational Data Mining Society.
2. **MacLellan, C.J.**, Harpstead, E., Aleven, V., Koedinger K.R. (2016) TRESTLE: A Model of Concept Formation in Structured Domains. *Advances in Cognitive Systems*, 4, 131-150.
3. **MacLellan, C.J.**, Koedinger, K.R., Matsuda, N. (2014) Authoring Tutors with SimStudent: An Evaluation of Efficiency and Model Quality. In S. Trausan-Matu, K. E. Boyer, M. Crosby, & K. Panourgia (Eds.), *Proceedings of the 12th International Conference on Intelligent Tutoring Systems* (pp. 551-560). Switzerland: Springer International. doi: 10.1007/978-3-319-07221-0
4. **MacLellan, C.J.**, Wiese, E.S., Matsuda, N., & Koedinger, K.R. (2014). SimStudent: Authoring Expert Models by Tutoring. In R. Sottilare (Ed.), *Proceedings of the Second Annual GIFT Users Symposium* (pp. 25-32). Orlando: US Army Research Laboratory.
5. **MacLellan, C.J.**, Harpstead, E., Marinier III, R. P., Koedinger, K.R. (2018). A Framework for Natural Cognitive System Training Interactions. *Advances in Cognitive Systems*, 6, 177-192.
6. Sheline, R., **MacLellan., C.J.** (2018). Investigating Machine-Learning Interaction with Wizard-of-Oz Experiments. In *Proceedings of the NeurIPS 2018 Workshop on Learning by Instruction.*
7. Harpstead, E., **MacLellan, C.J.**, Koedinger, K.R., Aleven, V., Dow, S.P., Myers, B.A. (2013) Investigating the Solution Space of an Open-Ended Educational Game Using Conceptual Feature Extraction. In S.K. D'Mello, R.A. Calvo, & A. Olney (Eds.), *Proceedings of the 6th International Conference on Educational Data Mining* (pp. 51-58). Memphis, TN: International Educational Data Mining Society.
8. **MacLellan, C.J.**, Harpstead, E., Wiese, E.S., Zou, M., Matsuda, N., Aleven, V., & Koedinger, K.R. (2015). Authoring Tutors with Complex Solutions: A Comparative Analysis of Example Tracing and SimStudent. In J. Boticario & K. Muldner (Eds.), *Proceedings of the Workshops at the 17th International Conference on Artificial Intelligence in Education AIED 2015* (Vol. 5, pp. 35-44). Aachen: CEUR-WS.org.
9. **MacLellan, C.J.**, Langley, P., Shah, J., Dinar, M. (2013) A Computational Aid for Problem Formulation in Early Conceptual Design. *Journal of Computing and Information Science in Engineering*, 13(3). doi: 10.1115/1.4024714
10. Wiederrecht, M., **MacLellan, C.J.**, Gamboa, R. (2010) Reasoning about DrScheme Programs in ACL2. In R. Page, Z. Horvath, & V. Zsók (Eds.), *Trends in Functional Programming* (pp. 276-283). Berlin: Springer-Verlag. doi: 10.1007/978-3-642-22941-1