Toward Stable Asymptotic Learning with Simulated Learners

No Author Given

No Institute Given

Abstract. Simulations of human learning have the potential to open a vast number of opportunities for learning technologies including AI supported authoring and automated testing of intelligent tutoring systems, enhanced student knowledge modeling, and simulated learning-by-teaching companions for both student and teacher learning. To date, simulated learner technologies have demonstrated considerable potential for many of these use cases, but the scope and reliability of simulated learner applications has been limited by the robustness of existing systems. In this work we identify several impediments to producing perfect asymptotic learning performance in simulated learners, introduce one significant improvement to the Apprentice Learner Framework to this end, discuss persisting causes of asymptotic error in our system and broadly discuss possible reframings of inductive task learning that address the need for new types of working memory.

Keywords: Simulated Learners · Cognitive Modeling · Authoring Tools

1 Introduction

Simulated learners are simulations of human learning that learn to perform tasks through an interactive process of demonstrations and feedback provided either by a human tutor or an Intelligent Tutoring System (ITS). Simulated learners have the potential to revolutionize learning technologies on a number of fronts. VanLehn suggested that simulated students could be used as teacher training tools, proxies for student collaboration, and as a means of testing and refining learning technologies in advance of classroom deployment [24]. Matsuda demonstrated that students can learn by teaching a simulated student called SimStudent [18], and Li showed the potential of simulations of human learning for cognitive model discovery [12]. Additionally, Matsuda, Maclellan, and Weitekamp [17][14][25] have demonstrated the use of simulated learners as a potential means of authoring ITSs [23], such as cognitive tutors [7], more efficiently than comparable methods [2] that do not employ simulated learners.

Across each of these use cases, a key consideration arises in how to evaluate different simulated learner models. Koedinger identified a number of standards by which competing simulated learner systems may be evaluated [8]. These include the generation of cognitive models that show close fit to student data when used with hierarchical mixed effect methods such as AFM[6], the generation of

erroneous actions reflective of human student behavior [26], and an agreement between human and simulated learners in error reduction per learning opportunity.

In this work we are concerned with producing simulated learners that exhibit perfect asymptotic performance, the performance of the simulated learners after considerable practice on a particular type of problem. This goal is important for purposes of quickly and robustly authoring ITSs with simulated learners, so that the resulting ITSs do not mark correct student responses as incorrect or incorrect responses as correct.

Toward the goal of modeling human learning, one might note that the performance of human learners often does not reach asymptotic perfection [4]. Nevertheless, identifying challenges in achieving asymptotic perfection in simulated learners may produce novel predictions for why humans continue to produce mistakes after achieving mastery on various tasks.

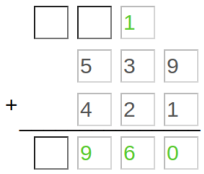
We explore the asymptotic performance of simulated learners using the Apprentice Learner (AL) Framework, a modular software library for creating simulated learners instantiating different mechanistic theories of learning [13]. In recent work, Weitekamp et al. presented the first demonstration of an interface and simulated learner implementation, using the AL Framework, whereby human users could train a simulated learner to near model-tracing completeness [25]. Authors could employ methods for addressing incompleteness or inaccuracy in the learned model, but found them frustrating in some cases. If AL could reach perfect performance more reliably and with less training, it would provide a better authoring experience.

In this work, we better identify sources of learning failure that prevent AL agents from achieving full accuracy. Further, we demonstrate new learning algorithms and instructional methods that help AL avoid these failure modes and approach perfect accuracy. Finally, we analyze sources of the few remaining errors and discuss potential remedies. In particular, we suggest extending AL's display-based learning and reasoning capabilities with a working memory that replicates observed displays and facilitates subgoal-based learning and reasoning. These contributions advance understanding of inductive task learning and point the way to improved asymptotic performance in systems such as ours.

2 Training Test Domain: Multi-column Addition ITS

To demonstrate issues in aysmptotic training behavior, we use mutli-column addition as a simple prototypical example. We train our agents on an ITS implemented with CTAT's [1] nools [16] model tracer [5] that supports practice on what is often called the "standard" or "traditional" algorithm for adding large numbers, whereby the digits of the numbers to be summed are aligned in columns, summed column by column, with an extra "carry" row that is used for carrying the tens' digit from one column to the next.

One of the challenges for simulated learners in this particular type of problem is that they must induce skills that condition on information not immediately



present in the observed state of the problem. Assume, for a moment, that the agent has taken the first two steps of the above problem and placed the 0 in the ones digit and the 1 in the carry field of the tens column. To take these two actions, instead of any other actions, the agent must have learned appropriate conditions on each of its induced skills that suppress any actions other than these two. Similarly, the next two actions must be taken only after the first two actions are taken, and the skill for placing the 9 must be aware that it should not wait for the carry field in its column to be filled in (since $1+3+2 < 10$). In other words some skills' conditions must test for the fact that the agent has mentally suppressed certain actions. Finally, the agent must press the done button to move on to the next problem. The rules for when the done button should be pressed must invoke similar purely mental information. It would not suffice for the agent to only press the done button when all of the fields were filled. In the majority of cases this rule would fail. Instead, the agent must mentally account for the fact that it has reached a point where there is nothing left to be done.

Thus, multi-column addition is a type of problem that cannot be solved purely by "display-based reasoning"—reasoning on information directly in the display[11]. In contrast, the rules for multi-column addition need to condition on a domain specific accounting of which actions still need to be taken. To overcome this issue our AL agents augment the problem state with every plausible application of skills that they have learned, allowing them to reason about actions that could have been taken but were not. This effectively means that as our agents learn new skills the richness of their perception of the problem grows.

It should be noted that the notation for multi-column addition can be altered slightly to make it a display based reasoning problem. Whenever a student would normally leave a box blank, they could instead be required to place a zero in that box. Then the matter of conditioning on domain specific mental information is no longer an issue, since no steps are mentally skipped and there is always a set of conditions on visible features of the interface that can indicate which skill should be applied next. We hypothesize that this alternative pedagogical strategy is likely less confusing to simulated learners, and human learners as well, especially in the early phases of learning. We focus our investigation on the standard strategy because it is more generally representative of ITS domains.

3 A Brief Overview of the Apprentice Learner Framework

As previously stated Apprentice Learner agents learn in an interactive process with an ITS or human author. An AL agent's knowledge is distributed across

a number of skills (in the form of production rules) each capable of performing a particular subtask. Initially, when the AL agent has no skills, or whenever it does not know what to do for an observed problem state, it will ask the ITS for a hint in the form of an example action that is correct for the next step of the problem. The AL agent will use these examples of correct behavior to induce a new Right-Hand-Side (RHS)—the "then" part of the if-then structure of a skill.

A RHS is a composition of domain general functions that produces an action. For example, the RHS for the skill that carries the 1 in the previous problem might be Mod10(Add(?.v, ?.v)) which takes two interface elements (the ?s) as arguments, sums their values (the .v's) and takes their one's digit (i.e. the modulus of 10). In AL agents, RHSs are produced by a brute force forward chaining process over a set of domain-general operations and the values currently present in an interface. This RHS inference process is termed *how-learning*.

A found RHS can work for a particular example but fail to work in general, for example another explanation an AL agent may come up with from the previous example is Copy(?.v) or just copy the value of the second value in a column into the carry slot. For 539+421 this would work for the first carry step, but would fail in general. In the AL agents used in this study a skill is identified by its RHS, so if the RHS happens to be wrong a new one must be induced, and the old one must be overridden or discarded.

The other part of a skill, the left-hand-side (LHS), holds the preconditions for firing a skill. In AL agents LHSs have three parts *where*, *when*, and *which*, and each of these parts are learned and refined from positive and negative examples via their own learning mechanisms. *Where-learning* learns a set of rules that pick out a set of arguments for a RHS, for example all of the numbers above the line in a column, and a "selection", the field into which the evaluation of the RHS will be placed. *Where-learning* is implemented by maintaining the specific-to-general boundary set of a version space [19]. Roughly speaking, this *where-learning* mechanism starts with a specific set of rules, that for example, bind just to the particular interface elements that the RHS was induced from, and apply generalizations of those rule sets by dropping conditions to accommodate new examples as they are seen, to for example bind to the interface elements in any column. In this work we'll discuss situations where *where-learning* can overgeneralize and bind to selections and arguments that it should not bind to.

When-learning learns the conditions, over the whole state, under which a skill should fire given a proposed *where-part* binding. For example, a *where-part* binding to elements in the third column of a multi-column addition problem might be proposed before the first or second columns have been handled. *When-learning* would learn a set of conditions on the whole state relative to the where-part's proposed selection that prevents the rule from firing when it is not supposed to. The AL agent implementation used in this work conditions the *when-part* on the *where-part* by relabelling the state relative to the *where-part*'s proposed selection (the field the action will be taken on). For example if there are three interface elements in a row named "A", "B", and "C" respectively, and "C" is the *where-part* proposed selection then the interface element identifiers "A", "B", "C",

would be remapped to "sel.left_of.left_of", "sel.left_of", "sel". Remapping the state in this way allows the *when-part* conditions to generalize across *where-part* bindings. For example, in a multi-column addition problem the same conditions could apply to a skill applied to any column and for problems with any number of columns. In general *when-learning* mechanisms can be any binary classifier, since they simply must predict whether or not each possible *where-part* binding ought to fire given the current problem state. In this study our agents use a custom decision tree implementation similar to C4.5 [21]. The input to this tree for training is simply a one-hot encoding of the triples ("object-id", "attribute", "value") present across each state (with "object-id" remapped relative to the selection as previously noted), and a positive or negative label produced during training feedback.

For a single problem state multiple *where-part* bindings can pass the learned *when-part* conditions, leading to multiple possible applications of learned skills to the same state. *Which-learning* learns a utility function over the passing skill applications that is greatest for those skill applications that are most likely to be correct. In the baseline AL agent implementation this utility value is simply the proportion of times that a particular skill has received positive feedback.

4 Experiment 1: Addressing Lingering Weak and Overgeneral Skills

AL agents may need to observe several examples of taking particular problem steps to induce the correct RHS for the true skill associated with that kind of step. In the process, skills are induced that have incorrect RHSs that do not consistently produce correct actions. Skills with incorrect RHSs are weak because they fail to produce the correct action in general. Weak skills will tend to be buried by building up a low *which-part* utility through repeated negative feedback, whereas correct skills may accumulate some negative feedback as their *when-part* conditions are refined and fewer later on. Weak skills tend to accumulate more negative feedback since they can make both *when-type* errors and errors of *how* an action is taken. Consequently correct skills tend to override weak skills by accumulating a higher *which* utility. Prior work with simulated learners has shown that overriding via *which* utility works well in many domains [15], but we have identified some circumstances that necessitate a revaluation of this method.

For instance, it is possible for RHSs to be misattributed to the wrong skill. Consider for example, the case of an untrained simulated learner seeing 215+846, and asking for examples of how to do the first few steps. Adding the 5 and 6 produces 11, so the first two actions will be to place a 1 below and carry a 1 to the next column. Regardless of which of the two actions is taken first, any RHS induced to explain the first action also explains the second since both actions have the same value. If a RHS is induced that is correct for either action then both actions will be attributed to the same skill (one of them is misattributed) even though in reality there should be two skills. Since the interface elements

on which the two actions act are different, the *where-learning* mechanism for that skill will over-generalize the conditions constraining legal bindings of the selection field causing the agent to apply the skill in a number of absurd ways. Another possible source of *where-part* overgeneralization is user error if an agent is being trained interactively. Prior to this work AL agents had no way to redress these sorts of overgeneralization issues.

4.1 Two Methods for Addressing Overgeneralization Errors

To address overgeneralization issues in AL agents we present two possible implementation changes and evaluate each independently. First, we implement a means for faulty skills to be removed including those that have overgeneralized. Second, we implement a *where-learning* mechanism that is capable of undoing generalization errors. A key observation in both proposed implementation changes is that faulty skills, either those with incorrect RHSs or overgeneralized where-part rules, will tend to make more errors than non-faulty ones, especially late in the training process. From a cognitive standpoint these can be thought of as persistent weak hypotheses of the true procedure. But these weak hypotheses should not persist indefinitely in the face of negative reinforcement, and should eventually be given up on.

Our first implementation change is to add a new removal utility, a number between 0 and 1 that when lowered below a threshold of .2 signals that a skill should be removed from an agent. We try three different functions of "p" and "n" (the numbers of instances of positive and negative feedback) for this utility: 1) the proportion correct $p/(p+n)$ (same as the *which* utility) 2) double counted negatives $p/(p+2n)$, and 3) nonlinearly counted negatives $p/p+n+1/4n^2$). Nonlinearly counted negatives implements the intuition that skills that persistently produce errors after considerable training are more likely to be faulty than skills that only produce errors initially while a skills' *when-part* rules are still being refined.

Our second implementation change introduces a fourth condition called "recovering where" that enables overgeneralized *where-part* conditions to return to a more specific state. Each newly generalized set of *where-part* conditions has its own removal utility that is updated, when applicable, with positive or negative feedback, and is removed when the utility calculated on the counts of positive and negative feedback falls below a threshold of .5.

4.2 Results of Implementation Changes

For all tested variations of the two proposed implementation changes we ran 100 agents on 100 3x3 multi-column addition problems. The first problem is always fixed to 215+846 to ensure that a large number of the agents exhibit the *where-part* overgeneralization issue, and the remaining 99 problems are sampled randomly.

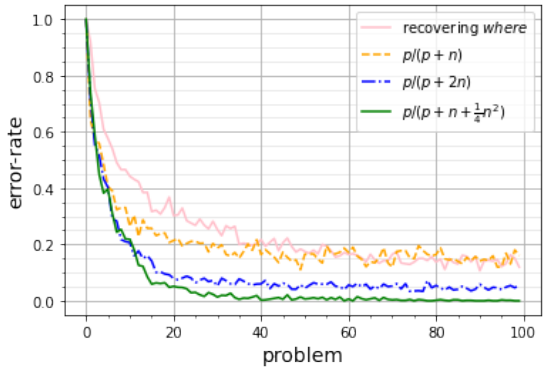


Fig. 1: Comparison of four recovery methods from *where-part* overgeneralization

Among the implementations of skill removal utility, nonlinearly counted negatives (i.e. "$p/(p+n+1/4n^2)$") was the only removal utility function that reliably removed overgeneralized and persistent weak skills. Our implementation of *where-part* generalization removal was essentially ineffective. We suspect that a large contributing factor to this is that each competing *where-part* generalization shares a *when-learning* mechanism, meaning that even if bad generalizations are eliminated, eventually a considerable number of unusual training instances centered around irrelevant selection fields will remain in the *when* training history, making it far more challenging to establish a set of consistent *when-part* conditions. Whole skill removal is likely more effective than generalization removal since removing an entire skill allows for a new skill to be induced in its place, giving *when-learning* a clean slate to work with.

5 Experiment 2 : An Analysis of Remaining Sources of Error

In the previous section we demonstrated that *where-part* overgeneralizations can be more quickly detected and removed by using a utility function that puts greater (non-linear) weight on higher failure counts. However, this technique does not achieve perfect performance as the average error after 40 problems remains at about 1%. After 40 multi-column addition problems we have found that agents have consistently arrived at a set of skills with correct RHSs and appropriately generalized *where-part* rules, however, some acquired *when-part* conditions are fully not accurately scoped. The decision tree *when-learning* mechanism sometimes produces incorrect behavior whether run with both a well established CART [9] based scikit-learn implementation [20] and our own implementation built with numba [10], a just-in-time compiler for python.

Small amounts of error are generally expected as a matter of course when using any kind of machine learning. However, to achieve authoring of a correct ITS,

we seek perfect performance. In our case the problem posed to *when-learning* has at least one correct answer and no sources of noise. Since we implemented the production rules for the multi-column ITS that the AL agents train on, we know that the true *when-part* conditions are within the scope of what a decision tree can learn. Additionally we know that the decision trees produced by our implementation always completely split the training examples, so there is not any underfitting of the data. The trees do however tend to contain more conditions than we know to be necessary, meaning that the remaining error is likely due to overspecialization of the trees. The main goal of the experiment that follows is to uncover the nature of this overspecialization.

One hypothesis is that even after training on dozens of problems our agents are not exposed to a diversity of edge cases, and errors persist asymptotically as edge cases are randomly encountered. For example, one class of edge case arises from the fact that our agents typically learn two separate skills for taking carry steps, one for when summing three numbers ("carry3") and another for when summing two numbers ("carry2"). Any skill that must condition on a carry step being mentally skipped must generate conditions functionally equivalent to checking that "carry2"!=1 and "carry3"!=1 in the previous column. However, for most problems the values of applying "carry2" and "carry3" are the same. So conditioning on only one of them would suffice. An edge case arises in a problem like 347+258 where applying "carry2" in the second column (tens(4+5) = 0) has a different value than applying "carry3" (tens(3+6+1) = 1). In this case an agent that had learned a *when-part* conditioned on either "carry2"!=1 or "carry3"!=1 but not both would produce an error.

5.1 Results of Training with Edge Cases

To test this hypothesis, we created 20 edge case problems that we expected to cause an agent using under- or over-specialized *when-part* conditions to produce an incorrect action in multi-column addition problems. We shuffled these edge case problems together with 20 random problems, and injected these 40 problems at the start of training in the "initial edge cases" condition, and after 20 random problems in the "delayed edge cases" condition. We compare these two new training sequences to a purely random sequence of 100 problems. Additionally, we ran these three training sequences with a display-based reasoning friendly implementation of a multi-column addition where the tutoring system requires the learner to explicitly enter zero when there is no carry.

Results from this simulation can be seen in Figure 2. These results confirm our intuition that a display-based reasoning version of the multi-column addition ITS is easier for AL agents to learn. All conditions achieve essentially perfect asymptotic behavior on this version of the ITS. However, in the normal implementation of multi-column addition all training sequences produce roughly the same .004 error in the last 20 problems.

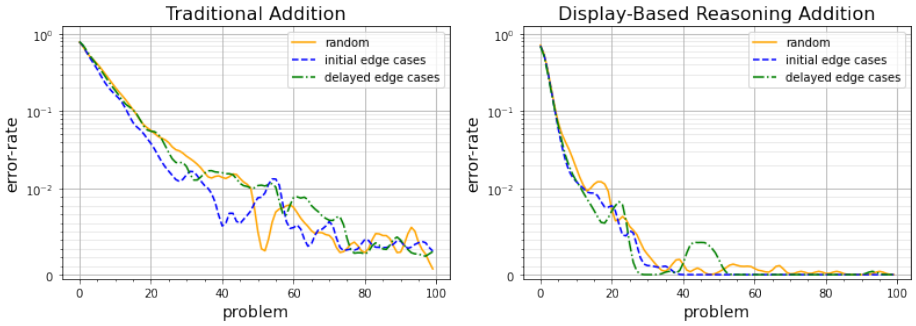


Fig. 2: Average error rate per training problem. Traditional multi-column addition (left) and display based reasoning friendly version (right). Trained with random (orange), initial edge case (blue), and delayed edge case (green) training sequences. The vertical axis is in log scale in the interval $[10^0, 10^{-2}]$ and linear thereafter. Values are smoothed horizontally with a sigma=1.3 gaussian filter.

5.2 Discussion of Remaining Asymptomatic Error

Since injecting edge cases does not appear to produce any significant effect on the asymptotic performance of our simulated learners, we can conclude that the remaining errors do not arise from an initial lack of exposure to edge cases in the random condition. Rather, the remaining asymptomatic errors can feasibly be explained by inaccurate specializations in *when-part* rules. Decision tree methods employ a divide-and-conquer strategy [21], meaning conditions introduced higher in the tree will tend to be supported by more data relative to deeper nodes that are more likely to handle infrequent edge cases and are more prone to spurious associations. By the end of training the *when-part* conditions learned by our agents recover several of the conditions embedded in the ITS, but the conditions that come later in each term only approximate more general conditions. Consider the following set of conditions learned for the Add2 skill.

	Raw Feature	Description
A	('contentEditable', .r)==False	field to right is filled in
B	(Div10(Add(?.v,?.v)), .a.a.a, .r.a.a, .r.a)!=1	carry2 does not produce 1 in this column
C	∃(Div10(Add(?.v,?.v)), .r.a.a.a, .r.r.a.a, .r.r.a)	carry2 is applicable in column to right
D	∃(Div10(Add(?.v,?.v)), .a.a.a, .r.a.a, .r.a)	carry2 is applicable in this column

Rule: ABC'+ABCD (i.e. AB(C'+CD))

The learned conditions A and B are exactly the same as two of four conditions in the equivalent production rule in the ITS, however, the disjunction C'+CD, which is learned deeper in the tree, is only a weak approximation of the other two conditions. This disjunction produces the correct behavior in the vast majority of cases. However, the cases in which it would fail are not specific

to any of the identified edge case problems, but may arise if skills are applied in a novel order because the relative which utility changes between them. A human author using the interactive training methods presented by Weitekamp [25] would likely quickly eliminate any incorrect specializations of this sort since authors can give feedback to all conflicting next steps proposed by an agent. Thus, these particular sorts of incorrect specializations are likely less of an issue in an authoring environment. In future work we plan to explore alternative methods of inductive rule learning that may produce more parsimonious rules. For example, some brute force search methods [22] have been shown to produce more succinct rules than existing tree methods.

The remaining .004 average asymptotic error in the traditional multi-column addition tutor is likely small enough to be admissible for the purposes of human modeling. However, the issues of incorrect specializations in the when-part may be exacerbated in more complex environments. The current design of these agents makes an assumption common to the framing of autonomous AI systems, where an agent's next action is purely a function of its current state. However, there may be alternative framings that additionally incorporate some variation of the idea of a persisting mental state or working memory. To this end we offer three possibilities which may operate in combination:

1. *Introduce a theory of memory activation* such as those outlined in ACT-R [3]. This could simplify condition learning problems by operating as a form of attention mechanism. Features, whether visible or induced, are probably more likely to be relevant if they have been considered recently and frequently.
2. *Add a form of subgoal induction* that could simplify the learning of procedure conditions, by spreading condition learning across skills. For example, the multi-column addition procedure has a recursive structure of subgoals for adding and carrying each column. If the system can acquire rules with subgoals, going beyond display-based reasoning, it may additionally make induced rules more understandable to authors.
3. *Reframe problems as display based reasoning problems*, making them easier to learn. An agent's learning may be scaffolded with these simpler framings of problems and then graduated to non-display based versions during which an agent may rely on a mental display to account for skipped steps, and other mental phenomena.

6 Conclusion

In this work we have identified challenges to achieving asymptotic performance with simulated learners and remediated sources of persistent asymptotic error in simulated learners implemented with the Apprentice Learner Framework. Furthermore, we have analyzed small remaining sources of asymptotic errors in this system. We generate the novel prediction that human multi-column addition learning will be enhanced by encouraging explicit entry of a 0 carry. Finally, we suggest that AL can be further improved by adding a working memory that facilitates mental display-based and subgoal-based learning and reasoning.

References

1. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In: International Conference on Intelligent Tutoring Systems. pp. 61–70. Springer (2006)
2. Aleven, V., Sewall, J., McLaren, B.M., Koedinger, K.R.: Rapid authoring of intelligent tutors for real-world and experimental use. In: Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06). pp. 847–851. IEEE (2006)
3. Anderson, J.R.: Act: A simple theory of complex cognition. American Psychologist **51**(4), 355 (1996)
4. Anderson, J.R.: Cognitive skills and their acquisition. Psychology Press (1981)
5. Blessing, S.B., Gilbert, S.B., Ourada, S., Ritter, S.: Authoring model-tracing cognitive tutors. International Journal of Artificial Intelligence in Education **19**(2), 189–210 (2009)
6. Cen, H., Koedinger, K., Junker, B.: Learning factors analysis–a general method for cognitive model evaluation and improvement. In: International Conference on Intelligent Tutoring Systems. pp. 164–175. Springer (2006)
7. Koedinger, K.R., Anderson, J.R., Hadley, W.H., Mark, M.A.: Intelligent tutoring goes to school in the big city (1997)
8. Koedinger, K.R., Matsuda, N., MacLellan, C.J., McLaughlin, E.A.: Methods for evaluating simulated learners: Examples from simstudent. In: AIED Workshops (2015)
9. L. Breiman, J. Friedman, R.O., Stone, C.: Classification and Regression Trees. Wadsworth, Belmont, CA (1984)
10. Lam, S.K., Pitrou, A., Seibert, S.: Numba: A llvm-based python jit compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. pp. 1–6 (2015)
11. Larkin, J.H.: Display-based problem solving. Complex information processing: The impact of Herbert A. Simon pp. 319–341 (1989)
12. Li, N., Cohen, W.W., Koedinger, K.R., Matsuda, N.: A machine learning approach for automatic student model discovery. In: Edm. pp. 31–40. ERIC (2011)
13. Maclellan, C.J., Harpstead, E., Patel, R., Koedinger, K.R.: The apprentice learner architecture: Closing the loop between learning theory and educational data. International Educational Data Mining Society (2016)
14. MacLellan, C.J., Harpstead, E., Wiese, E.S., Zou, M., Matsuda, N., Aleven, V., Koedinger, K.R.: Authoring Tutors with Complex Solutions: A Comparative Analysis of Example Tracing and SimStudent. In: The 2nd AIED Workshop on Simulated Learners. CEUR-WS.org, Madrid, Spain (2015)
15. MacLellan, C.J., Koedinger, K.R.: Domain-general tutor authoring with apprentice learner models. International Journal of Artificial Intelligence in Education pp. 1–42 (2020)
16. Martin, D.: Nools. https://github.com/noolsjs/nools
17. Matsuda, N., Cohen, W.W., Koedinger, K.R.: Teaching the teacher: Tutoring simstudent leads to more effective cognitive tutor authoring. International Journal of Artificial Intelligence in Education **25**(1), 1–34 (2015)
18. Matsuda, N., Keiser, V., Raizada, R., Stylianides, G., Cohen, W.W., Koedinger, K.R.: Learning by teaching simstudent–interactive event. In: International Conference on Artificial Intelligence in Education. pp. 623–623. Springer (2011)

19. Mitchell, T.M.: Generalization as search. Artificial Intelligence **18**(2), 203–226 (1982)
20. 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.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
21. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers San Mateo, CA (1993)
22. Rijnbeek, P.R., Kors, J.A.: Finding a short and accurate decision rule in disjunctive normal form by exhaustive search. Machine learning **80**(1), 33–62 (2010)
23. VanLehn, K.: The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. Educational Psychologist **46**(4), 197–221 (2011)
24. VanLehn, K., Ohlsson, S., Nason, R.: Applications of Simulated Students: An Exploration. Journal of Artificial Intelligence in Education **5**(2), 1–42 (1994)
25. Weitekamp, D., Harpstead, E., Koedinger, K.R.: 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 (2020)
26. Weitekamp, D., Ye, Z., Rachatasumrit, N., Harpstead, E., Koedinger, K.: Investigating differential error types between human and simulated learners. In: International Conference on Artificial Intelligence in Education. pp. 586–597. Springer (2020)