Toward a reflective SimStudent: Using experience to avoid generalization errors

Christopher J. MacLellan, Noboru Matsuda, and Kenneth R. Koedinger

Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh PA 15213, USA
cmaclell@cs.cmu.edu, mazda@cs.cmu.edu, and koedinger@cmu.edu

Abstract. Simulated learner systems are used for many purposes ranging from computational models of learning to teachable agents. To support these varying applications, some simulated learner systems have relied heavily on machine learning to achieve the necessary generality. However, these efforts have resulted in simulated learners that sometimes make generalization errors that the humans they model never make. In this paper, we discuss an approach to reducing these kinds of generalization errors by having the simulated learner system reflect before acting. During these reflections, the system uses background knowledge to recognize implausible actions as incorrect without having to receive external feedback. The result of this metacognitive approach is a system that avoids implausible errors and requires less instruction. We discuss this approach in the context of SimStudent, a computational model of human learning that acquires a production rule model from demonstrations.

Keywords: simulated learners, metacognition, cognitive modeling, representation learning, grammar induction, generalization error

1 Introduction

Simulated learning systems can be used for a wide range of tasks, such as modeling how humans learn, as teachable agents, and as a means to automate the construction of models that can be used in cognitive tutors. In an effort to reduce the amount of developer effort needed to deploy simulated learners for these tasks, researchers have been relying increasingly on the use of machine learning algorithms. However, by increasing the generality of these systems through machine learning approaches, these systems become more susceptible to making unrealistic generalization errors.

When using simulated learners to model human learning, we desire systems that predict student's errors as well as their correct behavior. Unrealistic generalization errors, in the context of these systems, are errors that the system predicts humans will make, but that they never actually make. If a system is prone to making these kinds of errors, then it becomes difficult to draw conclusions from the predictions the simulated learners makes for novel tasks.

These generalization errors also complicate the use of simulated learners as teachable agents because they result in a system that produces non-human behavior. When human students are teaching a simulated learner in a peer-tutoring scenario and it makes errors that humans never make, then it decreases the authenticity of the experience. This inauthenticity might effect the social dynamics of the learning-by-teaching scenario possibly making the teachable agent less effective.

Finally, generalization errors also have negative effects when using simulated learners to automatically build cognitive tutors. For this purpose, simulated learners have been used to author production rule models via interactive demonstrations of the solutions to the problems the system will tutor. This approach may decrease the amount of work required to build a cognitive tutor and allow subject-matter experts to author tutors directly, without an AI developer. In this paradigm, SimStudent's errors are useful to the extent that they correspond with typical student errors; in these cases, the resulting production rules can be added to the tutor's bug library. However, if the errors are unrealistic, the author must waste time identifying and deleting these nonsensical production rules.

In this paper, we propose an approach that uses background knowledge to mitigate unrealistic generalization errors with no changes to the underlying algorithms and which should increase the effectiveness of the underlying learning mechanisms. Before presenting this approach in section 4, we first review SimStudent, the simulated learning system that provides the context for this work (section 2) and introduce a motivating example of a nonsensical generalization error SimStudent currently makes (section 3). After presenting this approach, we present some initial results and discuss conclusions and future work.

2 The SimStudent Architecture

The simulated learner system that we focus on in this paper is SimStudent, a system that induces production rule models from demonstration and problem solving. The SimStudent system is used primarily for three tasks: to model and predict human learning, to author cognitive tutors, and to function as a teachable peer-agent.

In order to understand how SimStudent works and the situations in which it makes unrealistic generalization errors, we will review the types of knowledge used by SimStudent, how this knowledge is represented, and the learning mechanisms SimStudent uses to acquire this knowledge from experience.

2.1 Knowledge and Representation

There are three kinds of knowledge in SimStudent: primitive operator function knowledge, conceptual knowledge, and procedural knowledge. The first kind of knowledge is hand-constructed and consists of the low-level functions for manipulating data available to the system (i.e., adding two values, appending two

strings together, etc.). One example of a low-level function is SkillAdd, which accepts two arguments, each of type arithmetic expression, and returns the sum of these two expressions as a single arithmetic expression. These functions constitute SimStudent's background knowledge. Depending on the task SimStudent is being used for, different kinds of background knowledge may be appropriate.

Head	Body	Prob
Expression	← Number Variable	0.95
Expression	← Minus Variable	0.05
Variable	← x	1.0
Minus	← -	1.0
Number	← 0	0.1
	...	
Number	← 9	0.1

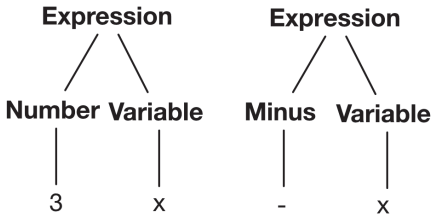

Fig. 1. A simple probabilistic context-free grammar and example parses of two expressions using this grammar.

The second kind of knowledge is conceptual, or representational, knowledge, which is encoded as a probabilistic context-free grammar. It is automatically acquired by SimStudent and is used to interpret the interface and information in it. Figure 1 shows a simple example of the conceptual knowledge SimStudent might possess about expressions for an algebra domain. This knowledge enables SimStudent to automatically extract plausible "chunks" from the input, such as the coefficient or term in an equation, which can subsequently be manipulated by primitive operator functions or procedural rules. Furthermore, this knowledge can be used to determine the likelihood that a given example was produced by the grammar.

If (current-row 'output-cell 'row)
(cell-in-row 'row 1 'left-side)
(is-left-child-of 'left-side 'coefficient)
→ **then** (write-text 'output-cell
(append "divide" 'coefficient)).

Fig. 2. An example production rule for division.

The final kind of knowledge is procedural knowledge, which represents the skills that we desire students to learn. This knowledge is encoded as production rules, which contain conditions under which the rules apply and what to do under those conditions. Figure 2 shows an example of a production rule signifying that when the left side of the equation's parse tree has a left child (here called coefficient), then enter "divide <the coefficient>" into the output cell.

2.2 Learning Mechanisms

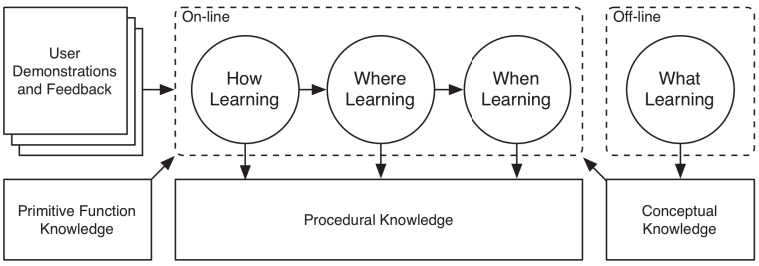

Fig. 3. A diagram of the SimStudent learning mechanisms and how they interact.

Of the three kinds of knowledge manipulated by the SimStudent system, two are learned automatically: the conceptual and procedural knowledge. To acquire these two kinds of knowledge the system employs four learning mechanisms: what learning, where learning, when learning, and how learning. The what learning is used to acquire the conceptual knowledge whereas the where, when, and how learning are used to acquire the procedural knowledge. Figure 3 shows how these four learning mechanisms interact. Before SimStudent is used, the what learning is run to acquire the conceptual knowledge. When SimStudent encounters a situation where it does not know how to act, which is common initially, it requests a demonstration from the author (the tutor developer or student tutor). This demonstration is comprised of four parts:

- *Focus of attention:* the set of relevant interface elements (e.g., the left and right hand sides of an equation);
- *Selection:* the interface element to manipulate (e.g., the output cell);
- *Action:* the action taken in the selection (e.g., update the text value); and,
- *Input:* the argument to the action (e.g, the text string used to update the selection).

Every time the system sees a new demonstration or gets corrective feedback on its performance, it learns or modifies a production rule. Production rule learning is done in three parts: 1) how learning attempts to explain the demonstration and produce the shortest sequence of primitive operator functions that replicates the demonstrated steps and ones like it, 2) where learning identifies a generalized path to relevant elements in the tutor interface that can be used as arguments to the function sequence, and 3) when learning identifies the conditions under which the learned production rule produces correct actions. We will now review each of these learning mechanisms.

What This mechanism operates off-line to acquire a probabilistic context-free grammar from only positive examples. This task can be defined as:

- *Given:* a set of examples of correct input;
- *Find:* a probabilistic context-free grammar with the maximal likelihood of producing the examples.

This task is performed using a grammar induction approach outlined by Li et al. [1], which uses a greedy approach to hypothesize the grammar structure and Expectation Maximization to estimate the grammar parameters.

Whenever a demonstration is given to SimStudent, it augments the provided information with the most likely parse trees of the content of each element in the focus of attention. This additional information is used by SimStudent in the subsequent learning mechanisms to extract deep feature knowledge from the content (e.g., to recognize and extract the coefficient of a term in an equation). The parse trees make this deep feature information directly accessible to SimStudent through the nodes in the parse tree (e.g., the left child of the parse tree for "$3x$" in Figure 1 corresponds to the coefficient).

How This is the first of three mechanisms executed in response to a demonstration. The how learning task can be defined as:

- *Given:* a set of demonstrations consisting of the state of the relevant interface elements and the parse trees of the contents of these elements as well as the resulting input for each state;
- *Find:* a sequence of primitive functions that when applied to each state produces the corresponding input.

This task is performed by exhaustively applying the primitive operator functions over all nodes in the focus of attention parse trees until the input is produced. The iterative-deepening depth-first search strategy is used to find the shortest sequence of functions that explains the data [1]. If no sequence exists, then a special functions is created that takes the states and produces the corresponding inputs.

Where This learning mechanism identifies the path to the relevant tutor interface elements. The tutor interface elements are specified by a hierarchical tree structure (a table is comprised of rows which each contain cells). During interactive instruction, the relevant interface elements are specified by the author teaching SimStudent. For each relevant element, SimStudent generates a parse tree for the contents. The relevant portions of these parse trees are defined as those that are utilized by the operator function sequence acquired through the how learning. The task of learning a general path to this relevant information can be defined as:

- *Given:* a hierarchical representations of the interface elements and their parse trees, the function sequence from the how learner, and a set of elements that have been identified as relevant;

- *Find:* a list of paths through the representation hierarchy to all of the relevant elements and the relevant portions of their parse trees.

The SimStudent approach to this task is to conduct specific-to-general learning over the set of relevant interface elements and parse trees [1]. Returning to the table examples, if the first cell in the first row of the table is always relevant, then a path to that specific cell will be returned. However, if all of the elements in the row are specified as relevant, then the entire row will be returned. After the location to the relevant elements has been identified, the system utilizes the function sequence to identify the relevant portions of the parse trees for each element. This same specific-to-general learning is then conducted over these relevant parse trees (within each element).

When This final mechanism identifies the conditions when the learned production rule is applicable. This task is defined as:

- *Given:* a set of positive and negative examples, each consisting of a set of features and their associated label;
- *Find:* a set of conditions over the features that separate the positive and negative examples.

As specified, this is a supervised learning task. The features used by SimStudent to represent each example are predicates that are automatically generated from the relevant portions of the parse trees. For example, there exists an "is-left-child-of" predicate, which says that a particular argument is the left child of a given node in one of the parse trees. This type of feature enables the retrieval of equations, terms, coefficients, and variables. Given the feature descriptions of each example, the positive and negative labels come from the user instructing the SimStudent system. The first positive example is the initial demonstration. Subsequent examples are generated when SimStudent tries to use the learned rules to solve novel problems and receives yes/no feedback from the author. To derive the set of conditions given the examples, SimStudent uses the FOIL algorithm [2], which uses information theory to perform a general-to-specific exploration of the space of hypothetical conditions.

These four learning mechanisms result in a simulated learning system that accepts user demonstrations and feedback and automatically acquires probabilistic context-free grammar rules and production rules. The system requires little background knowledge; for each task only the primitive functions need to be defined by the developer. However, the cost of this generality is a system that sometimes makes unrealistic generalization errors.

3 An example of an unrealistic generalization error

To explore the types of generalization errors that SimStudent makes, we turn to the algebra domain. One of the skills that students learn in this domain is how to proceed when given a problem of the form $< Symbol >< Variable >=<$

Symbol > (e.g., $3x = 6$). The skill that we desire the student to learn in this situation is to specify that their next step is to divide both sides by the coefficient of the term on the left side of the equation (the production rule from Figure 2).

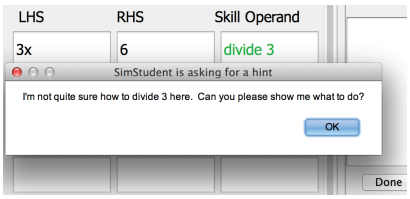

Fig. 4. SimStudent requesting a demonstration in an algebra tutor interface after the author has just entered "divide 3."

When SimStudent is first presented with a problem of this form, such as $3x = 6$, it will inform the author that it does not know how to proceed and ask for a demonstration. The author might demonstrate to SimStudent that the cells containing the left and right hand sides of the equation are relevant to the problem (by double-clicking on these cells) and update the next step interface element with "divide 3" (see Figure 4).

After receiving this demonstration, SimStudent parses the contents of the focus of attention (The first parse tree in Figure 1 shows an example of what the left hand of the equation might look like). Next, it employs the how learning mechanism, which searches for a sequence of functions that when applied to the nodes in the parse tree produce the input. In this example, it might learn to append the left child of the parse tree (for the left side of the equation) to the word "divide" and place it into the tutor interface (the then part of the production rule in Figure 2). Using the locations of the relevant elements (the left child of the parse tree), SimStudent then learns a general path through the representation hierarchy to the relevant elements and the relevant portions of the parse trees for these elements. Finally, SimStudent runs FOIL over the relevant information to learn the conditions under which the learned behavior is applicable. This results in the if portion of the production rule in Figure 2.

The learned production rule is more general than the single demonstration it was learned from; it is applicable for many equations, such as $4x = 12$ or $2x = 8$. However, when SimStudent is presented with a subtly different example that utilizes the same skill, $-x = 2$, it results in the mistaken generation of the input "divide -" (instead of "divide -1"). This is because in this situation the left child of the parse tree on the left hand side of the equation is a minus sign instead of the coefficient (see the second parse tree in Figure 1). In a review of problems of the form $-x =< Constant >$ in the 'Self Explanation CWCTC Winter 2008 (CL)' dataset accessed via DataShop [3], none of the human student made this error– therefore it is an example of unrealistic generalization error.

4 Reflecting before Acting

One reason that humans do not make this error is that they have a "sense" for what are reasonable output actions and they (subconsciously) reflect on actions before taking them. When a student is faced with the problem $-x = 2$ they may mentally produce the output "divide -," but realize that a "-" by itself is not mathematically grammatical because they have never seen an instance where this has occurred. This might lead them to consider a different action or to ask for help.

To reproduce this type of behavior, we modified SimStudent to utilize its conceptual knowledge, the probabilistic context-free grammar trained on example inputs (described as "what" learning in section 2). The acquired grammar is used to recognize when a potential output is not grammatical (when it cannot be parsed) and automatically flag the situation as a negative example. In other words, the system supervises itself and provides negative feedback (which the when learner uses) to improve its learning.

Now, when SimStudent is presented with a problem and finds an applicable rule, it simulates the execution of the rule and constructs a probabilistic parse of the value generated by the rule. If the value cannot be parsed by the current grammar (there is a 0% probability that the grammar produced the value), then SimStudent flags the trace as a negative instance and re-runs the when learning, which refines the conditions of the rule so that it no longer applies in the erroneous situation. If SimStudent has no other applicable rules, then it request a demonstration from the author, exactly like a human student.

5 Initial Results

To evaluate the effectiveness of this metacognitive loop, we have tested the probabilistic parser's ability to separate correct from incorrect actions based on the parse probability defined by the probabilistic context-free grammar. Table 1 shows five problems where SimStudent might make unrealistic errors. The first three are problems where SimStudent might induce a rule for dividing by the symbol before the variable instead of the coefficient. The last two problems correspond to inducing a rule retrieving the symbol after the variable and division sign instead of the entire denominator. On all five problems, the probabilistic grammar was capable of identifying the correct from the incorrect actions.

These results suggest that this approach is capable of identifying these kinds of errors. In general, this approach will be effective at identifying errors that result in non-grammatical output, where grammatical is defined by the probabilistic context-free grammar. This is effective because the rules are learned specific-to-general on a substantial amount of positive example inputs. By bringing this previous experience to bare, SimStudent can avoid nonsensical generalization errors and produce its own negative feedback, which enhances the effectiveness of its other learning mechanisms (more self-labeled examples for the when learning). Furthermore, this requires no additional work from an author and should reduce the amount of required author feedback.

Table 1. Five examples of problems where SimStudent might make the generalization error of retrieving the character before the variable or after the variable and the division sign, the corresponding correct and incorrect actions, the validity of these actions, and the parse probability of the actions.

Example	Possible Action	Valid	Parse Probability
$-x = 2$	divide $-$	No	0.00%
	divide -1	Yes	19.64%
$(-2)x = 6$	divide)	No	0.00%
	divide (-2)	Yes	0.09%
$3(x + 1) = 6$	divide (	No	0.00%
	divide 3	Yes	27.90%
$x/(-3) = 3$	multiply (	No	0.00%
	multiply (-3)	Yes	0.09%
$x/-5 = 1$	multiply $-$	No	0.00%
	multiply -5	Yes	19.64%

This task of verifying the output could alternatively be viewed as applying constraints to SimStudent's output and learning from constraint violations. Viewed this way, our work is related to the work on constraint-based tutoring systems [4]. In our case, there is only one constraint, "the output must be grammatical" where grammatical is defined as the probability of the output being produced by the grammar must be greater than 0%. We use a threshold of greater than 0% to signify grammatical, but one could imagine using a different threshold (e.g., greater than 0.05%). Thus, this constraint could be viewed as a probabilistic constraint that is automatically acquired from positive training examples.

6 Conclusion and Future work

In this paper, we outlined a novel approach to detecting and learning from unrealistic generalization errors that can be employed by simulated learner systems. The implications of this approach are threefold: (1) its use will result in models of learning that more closely aligns with human data, (2) teachable agents using this approach will be more realistic for the students using them, and (3) developers can produce cognitive tutor models with less work.

While this approach shows promise, it clearly has some shortcomings that should be remedied in future work. First, a more in-depth analysis of the alignment between SimStudent and human students is necessary. Previous work [5, 6] has looked at the human errors that SimStudent is capable of predicting, but a more detailed analysis of the unrealistic generalization errors, or errors that SimStudent makes that human students do not, would be useful. This would serve as a baseline to evaluate the SimStudent model and to evaluate the effectiveness of this approach.

A second direction for future work is to compare this approach to other approaches that might reduce these errors. We could imagine a system that has additional condition knowledge for the operator functions so that it would not generalize to situations where the function sequence would not be applicable (such as trying to divide by a symbol instead of a number). It would also be interesting to explore how reflection might facilitate the acquisition of this additional condition knowledge for the operator functions.

Finally, we are interested in applying this approach in other more complex and open-ended domains such as in RumbleBlocks, an educational game that teaches K-3 children about the relationships between the concepts of stability, low center of mass, wide base, and symmetry. We have been exploring how probabilistic grammars can be used to learn conceptual knowledge in RumbleBlocks [7] and we believe that this approach should scale up to this more complex domain.

References

1. Li, N., Schreiber, A.J., Cohen, W.W., Koedinger, K.R.: Efficient Complex Skill Acquisition Through Representation Learning. Advances in intelligent tutoring systems **2** (2012) 149–166
2. Quinlan, J.R.: Learning Logical Definitions from Relations. Machine Learning **5** (1990) 239–266
3. Koedinger, K.R., Baker, R.S.J.d., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J.: A Data Repository for the EDM community: The PSLC DataShop. In Romero, C., Ventura, S., Pechenizkiy, M., Baker, R.S.J.d., eds.: Handbook of Educational Data Mining. CRC Press (2010)
4. Mitrovic, A., Ohlsson, S.: Evaluation of a constraint-based tutor for a database language. International journal of artificial intelligence in Education **10** (1999) 238–256
5. Lee, A., Cohen, W.W., Koedinger, K.R.: A Computational Model of How Learner Errors Arise from Weak Prior Knowledge. In Taatgen, N., van Rijn, H., eds.: Proceedings of the Annual Conference of the Cognitive Science Society, Austin, TX (2009) 1288–1293
6. Matsuda, N., Cohen, W., Sewall, J., Lacerda, G., Koedinger, K.R.: Evaluating a Simulated Student using Real Students Data for Training and Testing. In Conati, C., McCoy, K., Paliouras, G., eds.: Proceedings of the International Conference on User Modeling. (2007) 107–116
7. Harpstead, E., MacLellan, C., Koedinger, K.R., Aleven, V., Dow, S.P., Myers, B.: Investigating the Solution Space of an Open-Ended Educational Game Using Conceptual Feature Extraction. In: Proceedings of the International Conference on Educational Data Mining. (2013)