A Framework for Parameterized Design of Rule Systems Applied to Algebra

Eric Butler, Emina Torlak, and Zoran Popović

Department of Computer Science and Engineering, University of Washington,
Seattle, WA 98195 USA
{edbutler,emina,zoran}@cs.washington.edu

Abstract. Creating a domain model (expert behavior) is a key component of every tutoring system. Whether the process is manual or semi-automatic, the construction of the rules of expert behavior requires substantial effort. Once finished, the domain model is treated as a fixed entity that does not change based on scope, sequence modifications, or student learning parameters. In this paper, we propose a framework for automatic learning and optimization of the domain model (expressed as condition-action rules) based on designer-provided learning criteria that include aspects of scope, progression sequence, efficiency of learned solutions, and working memory capacity. We present a proof-of-concept implementation based on program synthesis for the domain of linear algebra, and we evaluate this framework through preliminary illustrative scenarios of objective learning criteria.

Keywords: Intelligent Tutoring Systems, Program Synthesis, Automated Domain Modeling, Artificial Intelligence

1 Introduction

Creating an appropriate domain model (i.e., a set of rules capturing expert behavior) is an integral part of designing intelligent tutors. In general, the domain rules are considered intrinsically rigid and tied to the content to be learned, while the student model accounts for variability and specialization. Prior work on creating domain models relies heavily on expert modeling. This affects the expense of tutor development, estimated at 200-300 hours per hour of content [9]. There has been some work on learning domain rules semi-automatically in the context of Intelligent Tutors [4, 7]. Both manual and semi-automatic processes, however, assume the domain model is defined by one canonical set of rules.

We postulate that the domain rule set is not a fixed entity, but one that can be specialized for each learning context by considering factors related to scope of coverage and sequence of progressions. Furthermore, instead of separating all student factors into the student model, we explore the effects of incorporating student population traits in the design of the domain rules. For example, in determining the optimal rule set for introductory algebra, we consider the relative complexity of rule trigger conditions, and the working memory demands related

to the number of rules students need to remember. By considering more factors in the process of the domain model creation, we aim to more precisely target the domain model to the specific scope and sequence goals as well as student traits.

This paper presents RuleSynth, a framework for distilling the domain model that is optimal for a specific set of learning objectives. Given an objective function and an initial (suboptimal) set of rules, RuleSynth produces a new set of rules that collectively optimize the given learning objective. We focus on creating a domain model in terms of condition-action rules (akin to production rules in a cognitive tutor), so that it can be applied to any existing or new rule-based tutoring system. RuleSynth creates new rules using DSL-driven inductive program synthesis.

The key contributions of this work are a new framework for customizing the domain model that optimizes certain feature properties, and a preliminary evaluation of a proof-of-concept implementation in the domain of introductory algebra. Our evaluation shows that different learning objectives lead to dramatically different rule sets and that we can do so efficiently enough for customized intelligent tutoring systems to become a reality in the near future.

2 System Description

In this paper, we focus on domain models for solving linear algebraic equations. We represent a domain model as a set of *condition-action rules* (akin to production rules in a cognitive tutor). Each rule has a *condition* for when the rule may be applied, and an *action* to perform the rule. Our goal is to automatically produce a domain model (i.e., set of rules) optimizing some learning criteria.

To work towards this goal, we built a proof-of-concept system, RuleSynth, that produces the best rule set given an initial domain model for algebra problems and an objective function. The objective function captures student constraints (such as limited working memory) and goals (such as solving problems in a few steps). Our starting set of rules is listed in Table 1; we call these rules *axioms*. Our axioms, along with backtracking search, are sufficient to solve a large class of linear algebra problems. However, while simple to state and suitable for automated problem solving, this set of rules is difficult for humans to use, as it leads to inefficient solutions with many steps. RuleSynth uses the axioms to synthesize a large set of *macro rules* that lead to shorter solutions, and it uses the objective function to select the best rule set from the resulting pool of rules.

3 Evaluation

To evaluate our framework, we used it to generate several novel rules for solving algebra problems, which are shown in in Table 2. We then investigated a few hypothetical scenarios and objective functions on these synthesized rules. Our objective functions and cost models were hand-crafted, but, in principle, could be based on student data or generically defined based on cognitive principles. These sample scenarios are not exhaustive nor intended to be exemplars for

Table 1. The set of axioms used as input for our system.

Label	Description	Example
A	Additive Identity	$x + 0 \to x$
B	Adding Constants	$2 + 3 \to 5$
C	Multiplicative Identity	$1x \to x$
D	Multiplying by Zero	$0(x + 2) \to 0$
E	Multiplying Constants	$2 * 3 \to 6$
F	Division Identity	$\frac{x}{1} \to x$
G	Canceling Fractions	$\frac{2x}{2y} \to \frac{x}{y}$
H	Multiplying Fractions	$3\left(\frac{2x}{4}\right) \to \frac{(2*3)x}{4}$
I	Factoring	$3x + 4x \to (3 + 4)x$
J	Pushing Negatives	$-(3x) \to (-3)x$
K	Expanding Negatives	$-x \to -1x$
L	Adding to Both Sides	$x + 4 = 2 \to x + 4 + -4 = 2 + -4$
M	Dividing Both Sides	$3x = 2 \to \frac{3x}{3} = \frac{2}{3}$
N	Multiplying Both Sides	$\frac{x}{3} = 2 \to 3\left(\frac{x}{3}\right) = 2 * 3$

what would be used in a real tutor. Rather, they are intended to illustrate how a variety of objective functions can produce different domain models for different situations, all starting from the same synthesized rule set. Based on our results, we believe that, through the crafting of appropriate objective functions (which may depend on student models and live data), tutors could use RuleSynth to automatically adapt their domain models to particular learning situations.

Table 2. A sample of the macro rules found and synthesized by our system, with example applications. Several are common rules taught in algebra such as combining like terms (**IB**) or moving a constant's opposite to the other side of an equation (**LBA**).

Pattern	Example
BA	$x + 2 + -2 \to x$
LBA	$x + 2 = 3 \to x = 2 + -3$
MG	$3x = 6 \to x = \frac{6}{3}$
LBAMG	$3x + 2 = 1 \to x = \frac{1+-2}{3}$
NHG	$\frac{x}{4} = 2 \to x = 2 * 4$
IB	$2x + 4x \to 6x$
LJIBD	$x + 3y = 2 + 3 \to x = 2 + 3 + -(3y)$
KMG	$-x = 5 \to x = \frac{5}{-1}$
BLBA	$3 + x + 2 = 1 \to x = 1 + -5$

3.1 Balancing solution size and rule set size

Our first scenario considers balancing the size of the rule set and the efficiency of solutions. Given a set of rules $\mathcal{R}$ and example problems $\mathcal{E}$, we define the objective function to be a weighted sum of the *rule-set cost* and the *solution cost*, subject to the constraint that all problems in $\mathcal{E}$ are solvable with the chosen rule set $\mathcal{R}' \subseteq \mathcal{R}$. Thus, our objective function takes the form

$$C(\mathcal{R}', \mathcal{E}) = \arg\min_{\mathcal{R}' \subseteq \mathcal{R}} \alpha R(\mathcal{R}') + (1 - \alpha) S(\mathcal{R}', \mathcal{E}) \tag{1}$$

where $\alpha \in [0, 1]$ is a weighting term, $R(\mathcal{R}')$ is the rule-set cost, and $S(\mathcal{R}', \mathcal{E})$ is the solution cost.

We define the rule-set cost to be the sum of the costs of its rules, i.e., $R(\mathcal{R}') = \sum_{r \in \mathcal{R}'} \text{cost}(r)$. The cost of a rule is itself a weighted sum of costs of its condition and action. The condition cost measures the size of the condition expression, thus estimating the amount of work required to evaluate the condition. The action cost is defined as the number of elements that are added or removed to the equation during the application of a rule. For example, moving a term to the other side of an equation costs two: one to remove the term and one to add it to the other side. Intuitively, macros tend to have more expensive conditions (i.e., they are harder to apply) but lower action costs than the axiom subsequence they replace (because they compress the replaced actions into fewer steps).

The solution cost $S(\mathcal{R}', \mathcal{E})$ minimizes the average solution cost over all example problems $\mathcal{E}$. That is, $S(\mathcal{R}', \mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \min_{\{r\}_n \in \text{solns}(\mathcal{R}', e)} \sum_{i=1}^{n} \text{cost}(r_i)$. The function $\text{solns}(\mathcal{R}', e)$ is defined as the set of all finite sequences of rules from $\mathcal{R}'$ which solve the problem e. We therefore take the cost of a problem $e \in \mathcal{E}$ to be the sum of the cost of every rule in the shortest solution to that problem.

We considered two different versions of the optimization problem defined by Equation 1, which set the weighting term α to nearly 1 and to 0.3, respectively. The first version ($\alpha \approx 1$), which we call Optimization A, represents an objective function that tries to find all of the rules that are used in the shortest solutions to $\mathcal{E}$, discarding only unused rules. The second version ($\alpha = 0.3$), which we call Optimization B, represents a trade-off between having efficient solutions and keeping the total size of the rule set small.

Table 3 compares the results of running each optimization on our synthesized rules. For Optimization A, which considers only average solution cost, the rule set includes a large number of rules based on macros. This is because almost every macro makes at least one example more efficient to solve. Also, there are very few axioms. Most of them, while generally applicable, are obsolesced by one or more macro rules. On the other hand, Optimization B, which balances average solution cost with total rule cost, contains many more axioms and fewer macros. The axioms, while making solutions more expensive since more steps are required, are more broadly applicable since they can be used in combination. However, some macro rules which are themselves very broadly applicable (e.g., combining like terms) remain in the optimal solution. Making this weighting dynamic in a live tutor would enable the tutor to adjust the domain model along the spectrum of maximizing solution efficiency or minimizing rule set size.

3.2 Adapting rule sets to teacher-specified problem sequence

As another example scenario, suppose that we have a sequence of problems we wish to use (perhaps provided by a teacher), broken up into discrete units. RuleSynth can automatically find a sequence of rule sets that cover the entire sequence of problems, introducing only the minimal number of rules when needed for each unit. That is, for each unit of problems, we would like the minimal

Table 3. Optimization results for an objective function minimizing average solution cost and total rule cost. Optimization A considers only average rule cost whereas Optimization B balances the two.

Rules from Optimization A		Rules from Optimization B	
A	KMG	A	L
C	LBAMGLBA	B	IB
D	LJIBDLBA	C	NHG
MC	LJIBD	D	MG
F	IB	MC	LBA
G	NHG	F	
J	LBAMG	G	
BLBA	MG	H	
KNHGMG	LBA	J	
NHGHMBHGMG		K	

set of rules (with respect to the cost defined in Equation 1) that covers these problems and is a superset of the rule set for the previous sequence. Given a sequence of problem sets $\mathcal{E}_1, \ldots, \mathcal{E}_n$, we solve n sequential optimizations, where the i^{th} objective function is $C(\mathcal{R}_i, \mathcal{E}_i)$, subject to the constraints that all of $\mathcal{E}_i$ are solvable with $\mathcal{R}_i$ and either $i = 1$ or $\mathcal{R}_i \supseteq \mathcal{R}_{i-1}$. We chose such an example problem sequence and ran this optimization (with $\alpha = 0.3$), showing the results in Table 4. As can be seen, RuleSynth finds a small number of rules to add for each successive unit of problems. We only chose a few basic features for this optimization, but with a richer domain model, future versions of our framework can have a more sophisticated function for choosing progressions of rules.

Table 4. Optimization results for generating a sequence of rules. Each column is a successive unit of example problems. This tables shows which new rules (in addition to all previous columns) are required to cover the new set of problems.

Percent Coverage	25%	50%	75%	100%
New Rules Added	LBA, MB, BLAB D, C, A	NHG, LJIBD, E	IB, J	KMG, KNHGMG NHGHMHGMG

4 Related Work

There is a long history of work in learning within cognitive architectures [5]. In some of these architectures, there is a concept of "chunking" rules to create new rules. Our system explicitly is performing a similar kind of chunking by finding *macros* of the given set of rules, and synthesizing conditions and actions for these macros to create novel rules. More recently, researchers have looked at methods to help automate authoring domain models in tutors, including rule learning [4]. Closely related to our system is SimStudent, which is capable of inductively learning rules (for primarily algebra but also other domains) using Inductive Logic Programming (ILP) [7] and unsupervised learning of deep domain features with probabilistic grammars [8]. Other work used ILP to search for

rules given example applications from experts [3]. Our work is similarly inductive but uses program synthesis techniques. Other research has explored approaches to adapting content on the fly to students, by, for example, using multi-arm bandits for problem selection [1]. We are specifically concerned with choosing rules and domain models instead of problems. Inductive Programming / Synthesis has been applied to problem and solution generation [2], hint/feedback generation [6, 11], and rule generation [10]. Previous work in rule learning focused on learning individual rules, while we explore adapting rule sets to given learning criteria.

5 Conclusion

This paper presents RuleSynth, a framework for generating custom domain models that optimize desired learning objectives. RuleSynth employs discrete optimization to select the best set of rules from a pool of axioms and synthesized macros, according to a desired objective function. Our proof-of-concept implementation for algebra is able to synthesize several novel macro rules and produce optimal rule sets for example objective criteria. Our plans for future work include expanding to other domains to evaluate the generality of our approach, and exploring the impact of this system in the tutor design process.

References

1. Clement, B., Roy, D., Oudeyer, P.Y., Lopes, M.: Multiarmed bandits for intelligent tutoring systems. Journal of Educational Data Mining 7(2) (2015)
2. Gulwani, S.: Example-based learning in computer-aided stem education. Communications of the ACM 57(8), 70–80 (2014)
3. Jarvis, M.P., Nuzzo-Jones, G., Heffernan, N.T.: Applying machine learning techniques to rule generation in intelligent tutoring systems. In: Intelligent Tutoring Systems. pp. 541–553. Springer (2004)
4. Koedinger, K.R., Brunskill, E., Baker, R.S., McLaughlin, E.A., Stamper, J.: New potentials for data-driven intelligent tutoring system development and optimization. AI Magazine 34(3), 27–41 (2013)
5. Langley, P., Laird, J.E., Rogers, S.: Cognitive architectures: Research issues and challenges. Cognitive Systems Research 10(2), 141–160 (2009)
6. Lazar, T., Bratko, I.: Data-driven program synthesis for hint generation in programming tutors. In: Intelligent Tutoring Systems. pp. 306–311. Springer (2014)
7. Li, N., Cohen, W., Koedinger, K.R., Matsuda, N.: A machine learning approach for automatic student model discovery. In: Educational Data Mining 2011 (2010)
8. Li, N., Schreiber, A.J., Cohen, W., Koedinger, K.: Efficient complex skill acquisition through representation learning. Advances in Cognitive Systems 2 (2012)
9. Murray, T.: Authoring intelligent tutoring systems: An analysis of the state of the art. International Journal of Artificial Intelligence in Education 10 (1999)
10. Schmid, U., Kitzelmann, E.: Inductive rule learning on the knowledge level. Cognitive Systems Research 12(3), 237–248 (2011)
11. Singh, R., Gulwani, S., Solar-Lezama, A.: Automated feedback generation for introductory programming assignments. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (2013)