How should knowledge composed of schemas be represented in order to optimize student model accuracy?

Sachin Grover, Jon Wetzel, and Kurt VanLehn

(sachin.grover, jwetzel4, kurt.vanlehn)@asu.edu
Arizona State University

Abstract. Most approaches to student modeling assume that students' knowledge can be represented by a large set of knowledge components that are learned independently. Knowledge components typically represent fairly small pieces of knowledge. This seems to conflict with the literature on problem solving which suggests that expert knowledge is composed of large schemas. This study compared several domain models for knowledge that is arguably composed of schemas. The knowledge is used by students to construct system dynamics models with the Dragoon intelligent tutoring system. An evaluation with 52 students showed that a relative simple domain model, that assigned one KC to each schema and schema combination, sufficed and was more parsimonious than other domain models with similarly accurate predictions.

1 Introduction

In a recent review of the student modeling literature, Pelánek [21], pointed out that there are two major problems addressed by the field. One is to estimate a student's mastery of a set of knowledge components (KCs) given the student's performance on a set of items and a Q-matrix. The Q-matrix maps items to the knowledge components involved in responding correctly to the item. In one common formulation, the rows of the Q-matrix correspond to items, the columns correspond to KCs, and there is a 1 in a cell if that KC is involved in generating a correct response to that item. The Q-matrix is determined by the task domain alone. It is not a function of a student's knowledge or behavior.

The second problem addressed in the student modeling literature, and in this paper, is referred to as *domain modeling* by Pelánek [21]. An algorithm is given a large set of student responses to items, and it defines KCs, finds a Q-matrix and estimates the mastery for each student of each KC. Although there are methods for solving the domain modeling problem when posed in this fashion [1,8], the resulting KCs are hard to interpret. The only information about them is the Q-matrix. This makes it difficult to use the resulting profile of student mastery to guide instructional decision making.

Thus, the more common version of the domain modeling problem assumes that an expert defines both the KCs and the initial version of the Q-matrix,

and then student response data are used to improve the Q-matrix. This has the advantage that the set of KCs is not changed, so it is as interpretable and useful for decision-making as the expert can make them. Several methods have been developed to solve this version of the domain modeling problem [5,6,7,8,18].

Yet another version of the domain modeling problem is to manipulate the granularity of the KCs. One starts with an initial set of KCs, then either combines KCs to form large grained KCs, or splits KCs to form finer-grained KCs [2]. Other contributions to the study of KC granularity have compared several different domain models that differ in the granularity of their KCs [10,16].

All these domain modeling approaches make a convenient assumption: When students practice an item involving a certain set of KCs, only the KCs associated with that item (i.e., the 1's in that step's row of the Q-matrix) are affected by the practice. All other KCs maintain their same probability of mastery. Let us call this the KC independence assumption.

This KC independence assumption colors our way of thinking about learning. For example, when trying to see how competence grows, we can graph the probability of mastery of a KC over time. This is called a learning curve. The KC independence assumption means that the time-axis of the learning curve only needs to have the steps where that KC was involved. By assumption, its probability of mastery cannot change during other steps, so they can be left off the time-axis.

The KC independence assumption would seem natural to behaviorist who view all "knowledge" as a stimulus-response rules; each such rule is a KC. It would also seem appropriate to early information-processing psychologists who viewed all knowledge as production rules; each production is a KC. Information-processing psychologists who viewed knowledge as semantic nets could assume that most types of links in the network were KCs.

However, cognitive psychology today recognizes that people organize much of their knowledge into larger structures, often called schemas, scripts or frames. The key idea of such larger structures is that they are activated all at once. Thus, for instance, when students were read an algebra word problem slowly, as soon as they heard "A river boat is traveling upstream at 5 mph..." they activated an algebraic word-problem schema that not only contained the key equations but could even predict what the rest of the problem would say [19]. Similarly, when you hear a story that begins, "Roger asked the maître-de for a table near the window..." then you'll activate your restaurant schema, which predicts events such as ordering food and objects such as waiters and the bill.

After some rocky initial controversies, AI now accepts that schemas can be represented as production systems or semantic nets. For example, one can represent the river-current schema by using two kinds of productions: Some fire when a schema should be activated and dump an element such as "river-current-schema" into working memory. Other productions match that working memory element and dump a plan or other information into working memory that tell the agent how to enact the schema. It may take dozens or hundreds of production

rules to represent a single schema. Similarly, it make take dozens of hundreds of semantic network links to represent a single schema.

Although schemas are activated all at once, they might be too big and complex to be learned all at once. Thus, it might be inaccurate to treat each schema as a KC. If we want to use a domain model based on a Q-matrix, then perhaps we need to decompose schemas into pieces such at that each piece can be learned independently of the others.

This particular problem hasn't been addressed in the student modeling literature [20,21,23,24]). Nonetheless, we had to face it when trying to develop a student model for the Dragoon intelligent tutoring system [14,27,28,29]. We evaluated several different mappings of KCs to schemas. Like most explorations of domain models, our evaluation was based on the fit of the model to student data. We describe first Dragoon, then its schemas, then the domain models and finally the evaluation and its results.

2 Dragoon

Dragoon is a step-based tutoring system for teaching students how to construct mathematical models of systems that change over time. Although such models are often represented with differential equations, Dragoon use a node-link representation that was pioneered by Stella [9,22]. Figure 1 (which appears later), shows a screen capture of a simple problem and its model.

Dragoon has three types of quantities: parameter, accumulator and function. A parameter (diamond shaped node) is a quantity whose value is given as part of the problem statement and its value does not change over time. The other two types of quantities have values that can vary over time. Time is discrete. If a quantity is an accumulator (square shaped node), then its value at the next time is its value at this time plus the sum of its inputs. That is, it accumulates its inputs (which may be negative, so its value can go down). A function (circular node) is just a mathematical function of its inputs.

Students construct a model by adding nodes, one at a time, to a canvas. To define a node, they fill out a form. The form allows the user to specify properties of the node. Besides the name of the node, there are five properties:

- Description: A phrase describing the quantity
- Type: Either parameter, accumulator or function.
- Value: Parameter nodes have a numeric value. Accumulator nodes have an initial value. Function nodes do not have a value.
- Units: Some quantities have units, such as meters or kg.
- Expression: Function nodes have an algebraic expression for calculating their value. Accumulator nodes have an algebraic expression for calculating the increment to their value. Parameter nodes do not have an expression.

Students must type or click in the Expression. The other property values are selected from menus. Specifying one of these properties is the smallest unit of user interface behavior that can be interpret as contributing to the solving

of the problem. Thus, specifying a property is a step. As usual when dealing with multi-step problems, the Q-matrix maps KCs to step instead of items. This is a name change only. Just like items on a test, the student's responses to steps are assumed to be conditionally independent given the student's mastery of knowledge. From now on, we will refer only to steps instead of items.

Like many step-based tutoring systems, Dragoon colors a step green if the student enters a correct response and red if the entry is incorrect. If the student fails several times, Dragoon fills in the step correctly and colors it yellow. It has other features as well, but they are not relevant to this paper.

This section has introduced the syntax of the Dragoon notation for models. The concepts of node type (parameter, accumulator and function) and node properties (name, description, type, value, units and expression) will be important later, when we describe how to define domain models.

3 Schemas

In mathematics, electronics, physics, computer programming and many other task domains, there is evidence that experts and competent novices know many schemas [25,26]. For instance, when physicists notice that an object is flying through the air and that air friction can be ignored, they know its trajectory is a parabola, and they know equations for its horizontal and vertical positions over time. When programmers need to count something, then they know to define a variable, initialize it to zero, and increment it for each item counted.

Schemas have been extensively investigated for arithmetic and algebra word problems. For instance, Mayer [17] found that a mere 90 schemas sufficed to solve almost all the algebra word problems in 10 algebra textbooks. Author's in [4] provide a list and brief review of hundreds of relevant studies. Constructing a Dragoon model when given a succinct description of a system, such as the text shown in Figure 1, is nearly the same as solving a mathematical word problem. In both cases, one must understand the text and generate a mathematical model of the system described by the text. Thus, we assume that a Dragoon student's knowledge can be represented as schemas. Each schema has a part that matches the system description and a template that generates the model. The template consists of a set of nodes, but the properties of the nodes have only been partially filled in or constrained. Typically, the type and expression properties of the template nodes have been filled in, but the other properties have not. To apply a mentally held schema, the student notices that it can be applied, enters the appropriate nodes into Dragoon, and then fills in the missing information.

Although earlier studies with Dragoon used a variety of schemas, the study described here uses only three: linear, exponential and acceleration. These schemas are explicitly taught in system dynamics courses.

For example, the model shown in figure 1 is an instance of the exponential schema. The only difference between the schema and the model of figure 1 is that the model has specific information from the problem entered into the name, description, unit and value properties. A linear schema has only an accumula-

tor node and a parameter node, because the accumulator's value changes by a constant amount with each time step.

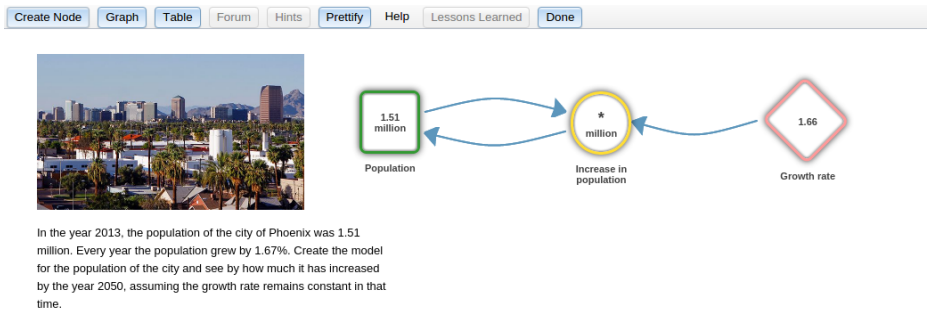

Fig. 1: Completed model in Dragoon showing *Exponential* Schema for Population Growth

3.1 Representing schema mastery with KCs

Our domain models are all based on Bayesian Knowledge Tracing (BKT). We use the original [3] version, were every KC has four parameters: slip, guess, learning rate and initial mastery. Forget is always set to 0.

Our first domain model, assumes that each schema corresponds to a KC for BKT. That is, to model a student, we could use just one number per schema, and that is the probability that the student has mastered the schema. For example, when the student creates the model shown in Figure 1, the student must fill in 5 properties for the "Population" node, 4 properties for the "Increase in population" node, and 4 properties for the "Growth rate" ode. That is, there are 13 steps in solving this problem. If we assume that each schema is a KC, then all 13 steps are mapped by the Q-matrix to the same KC, named Exponential. Let us call this the schema-only domain model.

However, we have observed informally that students find accumulator nodes to be quite confusing. On the other hand, parameter nodes seem more easily understood than the other node types. This suggests that we assigned different KCs to the different nodes of the schema's template. By having different KCs for the same schema, we can assume that the BKT parameters are different. For example, the exponential schema would be represented with three KC: Exponential_accumulator, Exponential_function and Exponential_parameter. Now we can express the conjecture mentioned earlier because the learning rate parameter for Exponential_parameter could be larger than the learning rate parameter for Exponential_accumulator. With the schema-only domain model, there is only one learning rate parameter for the whole schema. In our example of a student creating the model of Figure 1, the Q-matrix would map the 5 steps for defining

the "Population" node to the Exponential_accumulator KC. It would map the 4 steps for defining the "Increase in population" to the Exponential_function KC, and it would map the 4 steps for defining the "Growth rate" node to the Exponential_parameter KC. Let us call this the schema-type domain model.

The main goal of this study presented later is to determine which of these two domain models provides a more accurate assessment of students' learning. However, there is a complexity that needs to be discussed first.

3.2 Representing Compound Schemas

Some problems require applying more than one schema, and in most cases, the schema applications are linked by a shared node. Figure 2 is an example. It shows the solution to this problem:

> A bus fleet starts with 105 operational buses. Technicians can repair only 6 buses a week, but 9% of the bus fleet fails each week. Graph the number of bus in service each week for 100 weeks.

The "6 buses per week" part of the problem matches a linear schema. The "9% of the buses fail each week" matches the exponential schema. They share the quantity "number of buses in service" since both schemas affect it. Studies suggest that students have extra trouble when combining schemas to solve a problem. In particular, they can solve single-schema problems with high reliability, thus indicating mastery of the schemas, but fail miserably at solving a problem that combines the two schemas [13,15]. This suggests that more KCs are needed than those mentioned so far.

Several researchers have suggested that there is no generic skill for combining schemas, but instead students learn compound schemas [26]. A compound schema is itself a schema, but it is larger. For instance, a single compound schema would match the bus fleet problem of Figure 2. However, this solutions doesn't make sense for Dragoon. A student who has already mastered the linear and exponential schemas can easily construct three of the nodes in Figure 2; it is only the shared node, "bus fleet," that will cause them trouble. To put it differently, if we were using the schema-only domain model and we assigned all the steps of Figure 2 to the compound schema – let's call it exponential_linear – then the domain model would predict that students would make lots of errors on the nodes that are not shared. This seems unlikely to us.

Our solution is to map only the shared nodes to the KC representing the compound schema. Table 1 shows how the Q-matrix maps KCs to the steps required for constructing the model of Figure 2. It shows both domain models. Notice that only the accumulator, "Bus fleet," is mapped to the KCs representing the compound schema. The Q-matrix maps the other nodes to the same KCs as the single-schema problems.

3.3 Representing node properties

The format of a step can make a big difference to the guess and slip parameters of BKT. For instance, if a step is a menu that has only 3 items, and the student

Node	Schema-only domain model	Schema-type domain model
Buses fixed each week	Linear	Linear_parameter
Bus fleet	Exponential_linear	Exponential_linear_accumulator
Buses failing each week	Exponential	Exponential_function
Proportion of buses that fail	Exponential	Exponential_parameter

Table 1: KCs associated with steps of nodes in Figure 2.

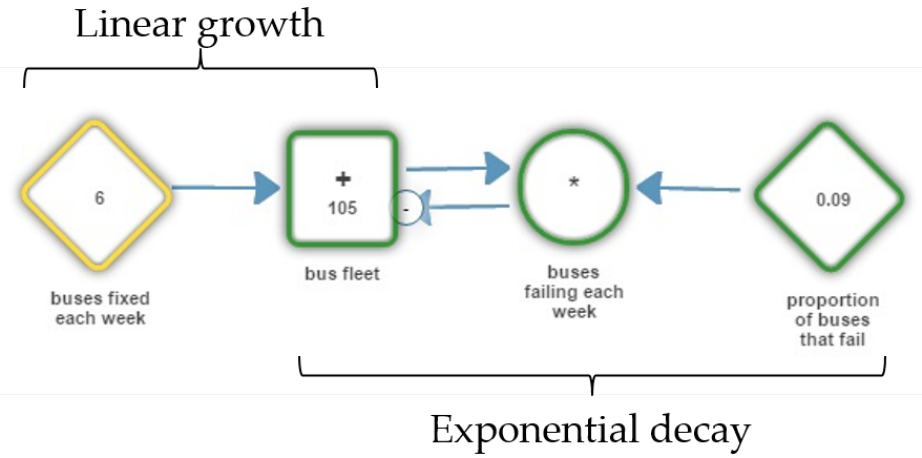

Fig. 2: Schema Overlap for a Node in Dragoon Problem - *Bus Fleet*

has not mastered the knowledge involved in answering that step, then the chance of a guessing correctly is probably about 1/3. If the step requires typing in an algebraic formula, then the chance of a lucky guess is quite low, and the chance of a slip is quite high. Thus, the format of steps affects the guess and slip parameters of BKT.

As mentioned earlier, when defining a node, the student must enter five properties: its description, type, value, units and expression. The expression is an algebraic expression that is typed or clicked in, but the others are entered by selecting from a menu. With both domain models, schema-only and schema-type, there is a single guess and slip parameter for all 5 properties. This is not likely to yield accurate predictions of error rates given the disparity in step formats.

One possible solution is to decompose the KCs, reducing their granularity even further. Thus, we would replace, say, linear_accumulator with 5 KCs, such as linear_accumulator_units, which would represent mastery of entering the units property of the accumulator node of a linear schema. Unfortunately, this may make the KCs overly independent. It predicts, for example, that someone could master linear_accumulator_value but have low mastery of linear_accumulator_units. This seems unlikely.

A better solution is to use a combination of BKT with logistic modeling. The basic ideas is to use functions to replace the slip and guess parameters, which in standard BKT have constant values per KC. Each function outputs a value between 0 and 1, which is the range of the slip and guess parameters. The input to the functions are details about the steps, such as whether the step is a menu or a type-in box. If it is a menu, the number of menu items could also be an input to the function. This would allow the functions to compute more values for the chances of a guess or a slip that are sensitive to the details of the step and thus likely to be more accurate than constant values. Logistic regression equations are often used for this purpose. They combine the inputs with a polynomial, then use a logistic function to compress the value of the polynomial, which can be any real number, into the 0.0 to 1.0 interval. They are called regression equations because calibration methods use regression to find values for the coefficients of the polynomial.

All logistic regressions have a constant term as well as one term for each input. The constant term is used to represent the dependence among details, such as the properties. For example, consider a logistic regression for the slip parameter of the linear_accumulator KC. Suppose we want it to be sensitive to the properties of steps. Since there are 5 properties, the polynomial part of the equations would be:

$$C_0 + C_1 * Descr + C_2 * Type + C_3 * Value + C_4 * Unit + C_5 * Expr \qquad (1)$$

The inputs are the variables $Desc$, $Type$, $Value$, $Unit$ and $Expr$. If the step is selecting from the description menu, then $Descr = 1$ and the others are 0, so the value for the slip parameter is the logistic function of $C_0 + C_1$. Similarly, if the step is typing in an algebraic expression, then $Expr = 1$ and the others are zero, so the value for the slip parameter is the logistic function of $C_0 + C_5$. Thus, the constant C_0 represents the portion of the slip parameter that is intrinsic to this KC, linear_accumulator, and the other coefficients represent the portions of the slip parameter that depend on the format/property of the step.

This version of BKT is called FAST [12]. Although it allows a domain model where every BKT parameter is replaced by a logistic regression equation, González-Brenes and other researchers found that replacing just the slip and guess parameters gives the best performance. We tried several ways of using FAST, and we too found that replacing only the slip and guess parameters gave the best performance. Our domain models thus assume that the other two BKT parameters, initial mastery and learning rate, are constants.

We also tried all logical combination of inputs to the regression equation. Thus, we ended up with 6 domain models, shown in Table 2. The first column shows the two KC models discussed earlier, schema-only and schema-type. It also includes a simple KC model, which assumes that the domain has just 3 KCs, one for accumulators, one for functions and one for parameters. It serves as a baseline; all the schema-based domain models should fit better, if the schema idea has merit. The second column indicates what kinds of terms, if any, are included in the polynomial part of the logistic regression equations for slip and guess. If the

column says "none" then the slip and guess parameters are constants, as in the standard BKT domain models. Otherwise, it lists the way steps are represented to the polynomial. The Parameters column lists the number of parameters in the domain model. The AUC columns present results from fitting these domain models to data from a study, which is described next.

Serial number	KC model	Polynomial terms	Parameters	AUC
1	Node-type	Node properties	25	0.805 ± 0.01
2	Schema-type	Node types	47	0.699 ± 0.006
3	Schema-type	Node properties	87	0.826 ± 0.0041
4	Schema-only	Node types	44	0.685 ± 0.012
5	Schema-only	Node properties	64	0.826 ± 0.0053
6	Schema-only	Node types & properties	108	0.833 ± 0.035

Table 2: Domain models and their fit to the data. AUC is the mean over ten datasets with the standard error value.

4 Evaluation: the study

We conducted a study to gather data that could be used to test which of the domain models fit best. The study simply had students solve Dragoon problems while log data were collected. We were only interested in how competence increased due to learning the two schemas that were taught, linear and exponential. Thus, we wrote problems that avoided or equated many known sources of difficulty with word problems [4]:

- The problems did not contain clue words, such as linear and exponential.
- The problems did not contain extraneous, irrelevant quantities.
- All problems contained about the same amount of imagery and details. The problem of Figure 1 is typical.
- The problems all contained moderately complicated numbers. They avoided simple numbers such as small whole numbers and they avoided complicated numbers, such as 0.0021379.

Participants: Participants were a mix of undergraduate and graduate students with at least high school algebra mathematics background. They were not asked about their background in modeling dynamic systems, as we have found in previous studies that very few students know anything about dynamic systems. 52 students took part in the study.

Procedure: Students came to our lab for one session where they finished 14 Dragoon problems. There was no upper limit to the time it would take. On average, students completed the problems in 2 hours 11 minutes. Students were compensated with money.

Materials: Two online workbooks were created to teach dynamic system modeling to students. The first workbook introduced Dragoon's interface to students along with a few basic concepts about modeling. There were three Dragoon problems in this workbook, with some multiple choice questions as well. The second workbook introduced students to the concept of schemas and then gave students 10 Dragoon problems to solve. The order of the first 7 problems was random. By randomly ordering the problems, we avoided a known problem with embedded assessments: competence is hard to detect when problems increase in difficulty [12]. The last 3 problems were always given in a fixed order and were quite difficult. This allowed us to compare the predictions of the models across students. Two of the problems included a novel schema, acceleration, which had not been taught to students.

4.1 Results

As stated earlier, two different kinds of models were fit and parameters were learned using FAST algorithm [11]. Data were fit using 5 fold cross validation (with 42 students for training and 10 students for test data) and results are presented as an average of the folds. Table 2 shows the average AUC values achieved for the test data. A higher value means a better fit.

5 Discussion

There is a clear pattern in Table 2, which is that a good fit requires that the node property be a part of the model. This makes sense, given that slip and guess are likely to be much different for menus than for typing in an algebraic expression. If that difference is ignored by the domain model, then the AUC is less than 0.7. If that difference is included in the domain model, then the AUC is around 0.8. In retrospect, this is a rather obvious finding. However, it does not seem to have been quantified in the literature before.

One of our main goals was to compare the fits of the schema-only domain model and the schema-type domain model. Table 2 shows that as long as the domain model include node properties, then the fits are very close. The best schema-type model had an AUC of 0.819 and the best schema-only model had an AUC of 0.833. Such close fits are common in educational data mining. If taken at face value, they say that the schema-only model is better than the schema-type model.

Among the 4 domain models that include node properties, the three that include schemas (models 3, 5 and 6) have higher AUCs than the AUC of the first domain model, which ignores schemas entirely. This vindicates including schemas in the domain model.

Now we are down to considering just three domain models (3, 5 and 6). They all have about the same prediction accuracy (AUC). However, model 5 (schema-only with a linear regression that includes node properties) has the fewest parameters. Thus, parsimony suggests it is the best model for these data.

This ambiguity points to a familiar problem: there may have been too little data to distinguish the models empirically. Even though the students generated 9336 steps, our schema-based domain models had over many parameters.

Moreover, the problems may have been too easy. Overall, students responded correctly on their first attempt to 85% of the steps. Even on steps in the problems involving the acceleration schema, which was not taught explicitly, students averaged 77% correct on their first attempt.

Nonetheless, if we take the results at face value, then a rather simple domain model emerges as the best fit to the data: Assign one KC to each schema; Assign one KC to each nodes shared by two schemas, and include node properties in the logistic regression equations for slip and guess.

Somewhat ironically, it appears that not only are schemas activated all at once, perhaps (again, taking the result at face value) these schemas are also learned all at once. Or at least, assigning one KC per schema is a good approximation for student modeling purposes.

6 Acknowledgments

This research was supported by NSF IIS-1628782, NSF IIS-1123823, ONR N00014-13-C-0029, ONR N00014-12-C-0643 and US Army, W911NF-04-D-0005, Delivery Order No. 0041.

References

1. Barnes, T., Stamper, J., Madhyastha, T.: Comparative analysis of concept derivation using the q-matrix method and facets. In: Workshop at Educational Data Mining at AAAI2006. pp. 21–30 (2006)
2. 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)
3. Corbett, A.T., Anderson, J.R.: Knowledge tracing: Modeling the acquisition of procedural knowledge. User Modeling and User-Adapted Interaction **4**(4), 253–278 (1994)
4. Daroczy, G., Wolska, M., Meurers, W.D., Nuerk, H.C.: Word problems: a review of linguistic and numerical factors contributing to their difficulty. Frontiers in Psychology **6**, 348 (2015)
5. De La Torre, J.: An empirically based method of q-matrix validation for the dina model: Development and applications. Journal of Educational Measurement **45**(4), 343–362 (2008)
6. DeCarlo, L.T.: Recognizing uncertainty in the q-matrix via a bayesian extension of the dina model. Applied Psychological Measurement **36**(6), 447–468 (2012)
7. Desmarais, M., Beheshti, B., Xu, P.: The refinement of a q-matrix: Assessing methods to validate tasks to skills mapping. In: Educational Data Mining 2014 (2014)
8. Desmarais, M.C., Naceur, R.: A matrix factorization method for mapping items to skills and for enhancing expert-based q-matrices. In: International Conference on Artificial Intelligence in Education. pp. 441–450. Springer (2013)

9. Doerr, H.M.: Stella ten years later: A review of the literature. International Journal of Computers for Mathematical Learning **1**(2), 201–224 (1996)
10. Feng, M., Heffernan, N., Mani, M., Heffernan, C.: Using mixed-effects modeling to compare different grain-sized skill models. Educational Data Mining: Papers from the AAAI Workshop. Menlo Park, CA: AAAI Press (2006)
11. Gonzalez-Brenes, J., Huang, Y.: Your model is predictive—but is it useful? theoretical and empirical considerations of a new paradigm for adaptive tutoring evaluation. In: Proceedings of the 8th International Conference on Educational Data Mining. University of Pittsburgh (2015)
12. González-Brenes, J., Huang, Y., Brusilovsky, P.: General features in knowledge tracing to model multiple subskills, temporal item response theory, and expert knowledge. In: The 7th International Conference on Educational Data Mining. pp. 84–91. University of Pittsburgh (2014)
13. Heffernan, N.T., Koedinger, K.R.: The composition effect in symbolizing: The role of symbol production vs. text comprehension. In: Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society. pp. 307–312 (1997)
14. Iwaniec, D.M., Childers, D.L., VanLehn, K., Wiek, A.: Studying, teaching and applying sustainability visions using systems modeling. Sustainability **6**(7), 4452–4469 (2014)
15. Koedinger, K.R., Nathan, M.J.: The real story behind story problems: Effects of representations on quantitative reasoning. The Journal of the Learning Sciences **13**(2), 129–164 (2004)
16. Koedinger, K.R., Yudelson, M.V., Pavlik, P.I.: Testing theories of transfer using error rate learning curves. Topics in Cognitive Science **8**(3), 589–609 (2016)
17. Mayer, R.E.: Frequency norms and structural analysis of algebra story problems into families, categories, and templates. Instructional Science **10**(2), 135–175 (1981)
18. Nižnan, J., Pelánek, R., Řihák, J.: Mapping problems to skills combining expert opinion and student data. In: International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science. pp. 113–124. Springer (2014)
19. Paige, J.M.: Cognitive processes in solving algebra word problems. Problem solving (1966)
20. Pavlik Jr, P.I., Brawner, K., Olney, A., Mitrovic, A.: A review of student models used in intelligent tutoring systems. Design Recommendations for Intelligent Tutoring Systems pp. 39–68 (2013)
21. Pelánek, R.: Bayesian knowledge tracing, logistic models, and beyond: an overview of learner modeling techniques. User Modeling and User-Adapted Interaction **27**(3-5), 313–350 (2017)
22. Richmond, B.: Stella: Software for bringing system dynamics to the other 98%. In: Proceedings of the 1985 International Conference of the System Dynamics Society: 1985 International System Dynamics Conference. pp. 706–718 (1985)
23. Shute, V.J., Kim, Y.J.: Formative and stealth assessment. In: Handbook of Research on Educational Communications and Technology, pp. 311–321. Springer (2014)
24. VanLehn, K.: Student modeling. In: Polson, M., Richardson, J. (eds.) Foundations of Intelligent Tutoring Systems, vol. 55, p. 78. Hillsade, NJ: Erlbaum (1988)
25. VanLehn, K.: Problem solving and cognitive skill acquisition. In: Foundations of cognitive science. pp. 527–579. MIT Press (1989)
26. VanLehn, K.: Cognitive skill acquisition. In: Spence, J.T. (ed.) Annual review of psychology, vol. 47, pp. 513–539. CA: Annual Reviews Inc (1996)

27. VanLehn, K., Chung, G., Grover, S., Madni, A., Wetzel, J.: Learning science by constructing models: Can dragoon increase learning without increasing the time required? International Journal of Artificial Intelligence in Education **26**(4), 1033–1068 (2016)
28. VanLehn, K., Wetzel, J., Grover, S., van de Sande, B.: Learning how to construct models of dynamic systems: An initial evaluation of the dragoon intelligent tutoring system. IEEE Transactions on Learning Technologies (2017)
29. Wetzel, J., VanLehn, K., Butler, D., Chaudhari, P., Desai, A., Feng, J., Grover, S., Joiner, R., Kong-Sivert, M., Patade, V., et al.: The design and development of the dragoon intelligent tutoring system for model construction: Lessons learned. Interactive Learning Environments **25**(3), 361–381 (2017)