Integrating Expert Knowledge With Automated Knowledge Component Extraction for Student Modeling

Rafaella Sampaio de Alencar
University of Pittsburgh
Pittsburgh, PA, USA
ras555@pitt.edu

Mehmet Arif Demirtas
University of Illinois
Urbana-Champaign
Urbana, IL, USA
mad16@illinois.edu

Adittya Soukarjya Saha
North Carolina State University
Raleigh, NC, USA
asaha4@ncsu.edu

Yang Shi
Utah State University
Raleigh, UT, USA
yshi26@ncsu.edu

Peter Brusilovsky
University of Pittsburgh
Pittsburgh, PA, USA
peterb@pitt.edu

Abstract

Knowledge tracing is a method to model students' knowledge and enable personalized education in many STEM disciplines such as mathematics and physics, but has so far still been a challenging task in computing disciplines. One key obstacle to successful knowledge tracing in computing education lies in the accurate extraction of knowledge components (KCs), since multiple intertwined KCs are practiced at the same time for programming problems. In this paper, we address the limitations of current methods and explore a hybrid approach for KC extraction, which combines automated code parsing with an expert-built ontology. We use an introductory (CS1) Java benchmark dataset to compare its KC extraction performance with the traditional extraction methods using a state-of-the-art evaluation approach based on learning curves. Our preliminary results show considerable improvement over traditional methods of student modeling. The results indicate the opportunity to improve automated KC extraction in CS education by incorporating expert knowledge into the process.

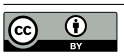
CCS Concepts

• **Applied computing** → **Education**; • **Human-centered computing** → **User models**; • **Social and professional topics** → *Computing education*; • **Computing methodologies** → *Knowledge representation and reasoning*.

Keywords

knowledge components, computing education, student modeling, intelligent tutoring systems, learning curves

ACM Reference Format:
Rafaella Sampaio de Alencar, Mehmet Arif Demirtas, Adittya Soukarjya Saha, Yang Shi, and Peter Brusilovsky. 2025. Integrating Expert Knowledge With Automated Knowledge Component Extraction for Student Modeling. In *33rd ACM Conference on User Modeling, Adaptation and Personalization (UMAP '25), June 16–19, 2025, New York City, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3699682.3728348



UMAP '25, New York City, NY, USA

ACM ISBN 979-8-4007-1313-2/25/06
https://doi.org/10.1145/3699682.3728348

1 Introduction

Student modeling is critical for personalized learning, as it allows intelligent educational systems to track the changing level of student knowledge and provide knowledge-adaptive support when needed [34]. The majority of student modeling approaches model learning knowledge on the level of individual concepts and skills, which are typically referred to as *knowledge components* or KCs [1, 15, 29, 36]. Personalized learning systems based on KC-level modeling achieved success in helping students learn in multiple STEM disciplines such as math [27] and physics [35].

The majority of past research on domain modeling, including math and physics, uses domain experts to construct domain models (i.e., a set of domain KCs) and to identify KCs in student answers (i.e., associate a problem solution or a step in this solution with specific KCs). However, the expert-based manual approach to domain modeling and KC identification has been challenging for domains when KCs are intertwined, such as computing education [13]. Not surprisingly, most domain modeling approaches in the domain of learning programming use *automatic* KC modeling and identification approaches ranging from data-driven KC discovery approaches [31] to extracting KCs from abstract syntax trees (AST) produced by parsing student code submissions [28].

However, the research on automatic KC modeling and identification in learning programming faces two challenges. The first challenge is **fitness** to expected KC properties (see Section 2). For example, the modeled KCs typically suffer from a low fit in the learning curves [4, 28, 31]. The second challenge is that data-driven extracted KCs often lack **interpretability**. One example is that while the KCs extracted recently fit the two KC properties [31], they cannot be interpreted as meaningful parts in students' code. Recent works have explored methods to solve these two challenges; however, they have yet to achieve a balance of fitness and interpretability [30].

In this paper, we explore the feasibility of combining expert-driven KC modeling approaches, known for their good interpretability, with the scaling-up power of automatic modeling to address both the fitness and interpretability challenges. The idea of our approach is to combine an expert-built ontology of semantically important Java concepts with AST-driven KC extraction from Java

programs. Our method uses traditional ASTs to automatically extract expert-defined ontology KCs, addressing the challenge of interpretability. To assess the value of this combination, we use a recently suggested approach [4] to build learning curves from a dataset of student code submissions. As a source of submissions, we use a popular benchmark dataset, which has been used in the CSEDM Challenge [24]. Our research question is: *How do student models using ontology KCs compare to student models using AST-based KCs in fitness and predictive power, while still being interpretable?*

The experimental results presented in this paper show that compared to the traditional AST parser, the ontology KCs demonstrate a much higher improvement over the naive single KC baseline for all problems, addressing the fitness challenge. Finally, the ontology KCs show significantly better predictive performance for new problems compared to other KC models, pointing to the success of the student model. Our preliminary work presents promising results for this hybrid approach, highlights the importance of expert knowledge for accurate student modeling, and provides implications for future work on knowledge tracing with expert-generated models.

2 Related Work

The Knowledge-Learning-Instruction (KLI) framework [15] uses the term knowledge components (KCs) to describe pieces of cognition or knowledge (e.g. concepts, principles, facts, or skills). KCs are used extensively in different domains as a foundation for student modeling and personalization. For example, a KC could be a word in a new language, a formula to calculate the area of a circle, or the value of the gravitational acceleration [15]. In each of these cases, a personalized learning system attempts to maintain an independent estimate of student knowledge of each of these KCs and uses it to support various types of adaptation.

To serve as an effective foundation for student modeling, KCs have to satisfy two key criteria: *cognitive validity* and *predictive validity*. Cognitive validity assesses whether KC represents elementary fragments of domain knowledge by examining *learning curve fitness*. Learning curves are built by analyzing problem-solving logs of students practicing a set of KC by solving a sequence or problems. Since every opportunity to solve a problem that requires an application of a KC is also an opportunity to master the KC, the error rate on the level of individual KCs should decrease with practice. For cognitively valid KCs, this drop in error rate follows an exponential rate form (i.e. exponentially decreasing learning curves) [3, 33]. Predictive validity assesses whether a student model built on top of KCs correctly predicts student's problem-solving performance or test results [9]. These two data-driven properties are the keys to our ability to compare performance of several alternative models in the same domain. In turn, this ability supported research on continuous identification and revision of KCs across disciplines [3, 22, 28].

In the domain of learning programming, the ability to compare domain models is especially important since the majority of domain modeling efforts use automatic KC modeling and identification rather than engaging domain experts [28, 31]. Not surprisingly, recent research on student modeling for programming [5, 28, 31] started to use this exponential rate property of learning curves as one of the measures for the validity of KCs.

Two of the most popular approaches for automatic KC modeling are data-driven *KC discovery* approaches [31] and AST-based *KC extraction* approaches [28]. Data-driven approaches use large volume of student data to find best-fitting KCs. However, KC candidates obtained through these approaches may be harder to interpret. For instance, in [31], KCs are represented only by a set of problems, which makes it difficult for instructors to understand what concrete skill is being practiced by manual inspection. AST-based approaches use nodes of AST produced by parsing student code as KCs (see Section 3.1.1). However, their learning curves did not show the expected reduction in error rate and a majority of their KCs did not capture learning, highlighting an opportunity for refinement [4, 28, 31]. For example, one intuitive way of using `for` loop as a knowledge component yields spiky learning curves, indicating the presence of other interconnected KCs, such as conditionals, variables, etc. Nevertheless, a recent replication of that study suggested that AST-based approaches could produce plausible KCs, given that students are provided with sufficient practice problems [5].

While AST-based KCs have had varied results, ASTs have been widely used in automated code analysis efforts [19]. For example, Rivers used ASTs to identify each syntax structure in a student's submission and compare it to a correct solution [28]. Nutbrown and Higgins discussed how the use of ASTs offered a degree of abstraction that made it easy to search for particular patterns in the tree structure, even while the flow and ordering of the program were preserved [21]. ASTs record the structural components of code snippets and operate as an abstraction of the original code. As a result, grammar symbols and other unimportant elements such as comments, whitespace, and, to a lesser extent, variations in variable names can therefore be dismissed as noise [18], which further refines the code snippets to reflect the general structure of source files and their meaning to be better evaluated. In modeling student success, Piech et al. used ASTs to track how students progress through a programming assignment to predict which students will struggle with the posterior course content [23]. AST approaches have also provided successful results for challenges focusing on knowledge tracing [2, 24]. However, using AST-based approaches for developing more fine-grained student models that tracks progress on individual skills is an open question.

3 Methodology

In this section, we first detail two domain models that define knowledge components for programming knowledge. Then, we explain how we used an approach from recent work for solving the KC attribution problem in open-ended code-writing data, enabling the evaluation of a novel KC model that combines expert knowledge with automated code analysis. For a deeper look at this work's methodology or to reproduce our results, please refer to our code repository in: https://github.com/Rafaellarsa/SummerSchoolCMU24/tree/main.

3.1 Domain Models

To model student knowledge, we applied two domain models to open-ended code-writing data to obtain knowledge components depicted in a tree representation for each submission.

3.1.1 Abstract Syntax Tree KCs. We extracted abstract syntax trees (AST) from Java programs and created knowledge components using AST nodes[1], following recent promising results for automated KC extraction in Python [5].

An AST is a representation of code where each syntactic token is represented as a node in a tree. Previous work on hint generation [25] and knowledge tracing and modeling [28] has utilized abstract syntax trees to compare correct and incorrect submissions. We leverage the use of these trees to compare submissions to correct solutions, find the syntactic tokens that are correctly placed in a submission, and identify the tokens that show a mistake in the submission [5, 28]. Through this method, each syntactic element becomes a KC that can be tracked across multiple attempts as explained in 3.2. However, while AST tokens can be helpful in tracking multiple skills in code writing with this approach, some tokens may correspond to concepts that are too granular to represent an individual skill in the student model, leading to low model fit and predictive power. Thus, some KCs of this granular domain model may not necessarily correspond to important aspects of programming knowledge.

3.1.2 Ontological KCs. To address the issue of granularity, we use JavaParser [12] to extract ontological KCs from code[2]. An ontology provides an index of concepts generated by domain experts, combining static analysis with expert knowledge. JavaParser can identify the occurrences of these concepts in the code by extracting ASTs and mapping them to the Java ontology created by a team of experienced Java instructors[3], instead of directly using the tokens as nodes. The parser returns the lowest-level ontological concepts behind the code, and we use ontology link propagation to reach upper-level concepts and form an alternative tree representation for the code. This can be seen as analogous to an AST, but instead of using syntactic tokens which may be too granular, the nodes are selected from a set of expert-identified concepts. Therefore, ontological KCs address the weaknesses of using syntactic tokens to model student skills but preserve the advantage of having a tree representation for KC attribution algorithms to track multiple skills from a single submission.

3.2 KC Attribution Algorithm

Open-ended code-writing exercises require students to apply many skills at once, whilst only providing feedback on the outcome of the code. This creates additional challenges for modeling KCs that may correspond to interleaved skills. Thus, recognizing successful progress on some KCs in a submission while recognizing incorrectly practiced KCs in the same submission is crucial for accurate student modeling. This has been formalized as the knowledge component attribution problem [32]. In our work, we implement a tree comparison algorithm from recent work [5] that compares an incorrect submission to a correct solution to detect KCs that should be modified to fix the submission. Originally proposed for hint generation, this algorithm relies on identifying *a point of deviation* where the incorrect submission applies a different KC from the known correct solution, taking the student down the wrong path.

Algorithm 1 Knowledge Component (KC) Attribution Algorithm

Input: Set of all student submissions on a given problem (S), submission of student i on the same problem (S^i), set of test cases for the problem

Output: List of correctly practiced knowledge components $KC+$, list of incorrectly practiced knowledge components $KC-$

1: Scorrect $\leftarrow$ filter(S_p, test cases)
2: $KCTree[S_{\text{correct}}] \leftarrow$ Extract KC trees from S_{correct}
3: $KCTree[S^i] \leftarrow$ Extract KC tree from S^i
4: **if** S^i passes test cases **then**
5: $KC+ \leftarrow KCTree[S^i].aslist(); KC- \leftarrow \emptyset$
6: **return** $KC+, KC-$
7: **end if**
8: $KC[S_{\text{closest}}] \leftarrow$ MinimizeTF-IDF($KC[S^i], KC[S_{\text{correct}}]$)
9: **return** CompareDFS($KCTree[S^i], KCTree[S_{\text{closest}}]$)
10: **function** CompareDFS(KC[in], KC[sol])
11: **if** $KCTree$[in] = $\emptyset$ **then**
12: $KC+ \leftarrow \emptyset; KC- \leftarrow KCTree$[sol].$aslist()$
13: **else if** $KCTree$[sol] = $\emptyset$ **then**
14: $KC+ \leftarrow \emptyset; KC- \leftarrow KCTree$[in].$aslist()$
15: **else if** $KCTree$[in].$head \neq KCTree$[sol].$head$ **then**
16: $KC+ \leftarrow \emptyset; KC- \leftarrow KCTree$[in].$aslist()$
return $KC+, KC-$
17: **else**
18: $KC+, KC- \leftarrow$ CompareDFS(Children($KCTree$[in]), Children($KCTree$[sol]))
19: $KC+ \leftarrow KC+ \cup KCTree$[in].$head$
20: **end if**
21: **return** $KC+, KC-$
22: **end function**

While the original algorithm is proposed for ASTs, we show that it can be generalized to ontological trees, providing a novel solution to the tradeoff between interpretability (utilizing expert knowledge) and model fitness (refining KC models automatically with data). The full method can be found in Algorithm 1.

KC attribution starts by building a set of correct solutions by filtering all student submissions that pass test cases to capture different approaches that may solve the same open-ended problem. Then, KCs in all submissions are extracted as trees (either using ASTs or ontology nodes). At this stage, if the student submission being evaluated also passes the test cases, all the KCs identified in the tree are attributed as correct attempts. If the submission fails at least one test case, it is compared to the most similar correct solution (determined by TF-IDF [17]).

This solution is used as the ground truth to identify which KCs are correctly placed in the submission and which KCs are mistaken by recursively comparing the trees for both solutions (line 10 in Algorithm 1). The comparison is similar to a depth-first search, where the head node of the trees are compared at each level of recursion. If the head nodes match, the KC represented by the node is attributed as a correct attempt. If the submission tree has a different node than the solution tree, the KC represented by that node is attributed

[1]https://github.com/Rafaellarsa/SummerSchoolCMU24/blob/main/KC-examples/AST-model.txt
[2]https://github.com/Rafaellarsa/SummerSchoolCMU24/blob/main/KC-examples/ont-model.txt
[3]http://www.sis.pitt.edu/~paws/ont/java.owl

as an incorrect attempt. The algorithm outputs two lists: KCs that the student practiced correctly in this submission (correct attempts, $KC+$) and KCs that the student had made a mistake (incorrect attempts, $KC-$). Thus, running the attribution algorithm on a single submission updates the student model by contributing one attempt per knowledge component used in the submission. By running the algorithm repeatedly on all submissions of a student on different problems, we collect time series data showing improvement of the student on each KC across multiple attempts.

4 Experiments

We evaluated our approach by applying learning curve analysis on a benchmark dataset for knowledge tracing, including previous data mining challenges [2, 24]. This dataset has been used as a benchmark in numerous previous student modeling and personalization works [7, 31, 37]. In this section, we describe the dataset, the learning curve analysis method for student modeling, and the analysis results for two domain models.

4.1 Dataset

We analyzed a Java programming dataset collected from the Codeworkout platform [8] for a CS1 course at a public US university in the Spring of 2019. This dataset is publicly available and was used in the 2nd CSEDM data challenge [24] and saved in the ProgSnap2 [26] format. Additionally, the dataset has been anonymized for ethical reasons, and no personally identifying information about specific students such as addresses or GPAs is made public. It contains 50 programming exercises, divided into 5 assignments of 10 exercises each, covering conditionals, logical operations, loops, and arrays. Each exercise has a prompt for the student, such as the one below:

> The number 6 is a truly great number. Given two int values, a and b, return true if either one is 6. Or if their sum or difference is 6. Note: the function Math.abs(num) computes the absolute value of a number.

An example of what a student experiences in CodeWorkout can be seen in Figure 1. The student needs to write a function following the exercise's prompt and, after submitting their code, receives a feedback of the function's performance in assigned test cases. For this dataset, the students used the platform for independent practice outside of class time in an unproctored setting.

Although the complete dataset includes more than 400 students, we used a random sample of 100 students from the platform for our analysis due to computational constraints. Following prior work, we collected only the first attempts at each problem, as students may alter their submission based on the test case feedback in subsequent attempts. Moreover, we took into consideration only the syntactically valid submissions following the previous work [20] as erroneous student submissions may not have a KC linked to them.

4.2 Learning Curve Analysis

We compared knowledge component models using learning curve analysis [3], a data-driven method for tracking student performance on individual KCs. A learning curve shows the error rates for a KC across many opportunities a student has. In our setup, each submission to a new programming problem becomes an attempt

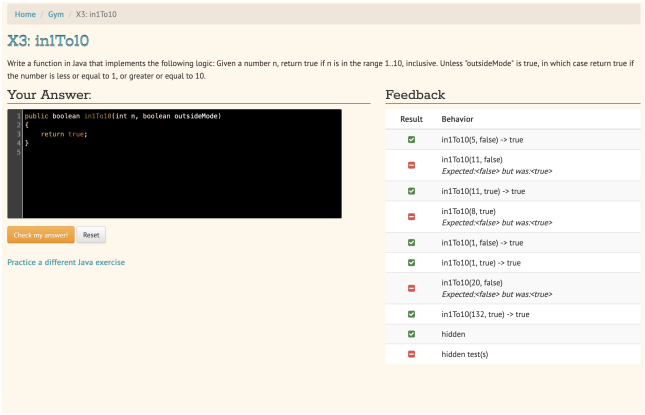


Figure 1: CodeWorkout's exercise interface.

for practicing KCs. According to the KLI framework [15], the error rates are expected to decrease exponentially as the students go through more practice opportunities.

Conversely, if students' practice of a potential KC shows an increasing or constant error rate across subsequent practice opportunities, the potential KC could be representing multiple skills at once or not even representing a skill, making it harder to build an accurate and fine-grained model of the student knowledge [28].

4.2.1 Analysis Platform. In line with the prior literature applying learning curve analysis, we used the modeling tools on PSLC DataShop [14] to compute and evaluate student models.

4.2.2 Evaluation Metrics. In addition to presenting learning curves visually, we also report root mean squared error (RMSE) values to show the power of the statistical model on predicting the error rates on new students (*RMSE (Stud)*) and new problems (*RMSE (Prob*) with the given set of knowledge components [16]. Moreover, following prior work [3, 22], we compare each model to a naive baseline where each opportunity is considered to practice a single knowledge component that represents overall programming experience, and report the improvement over this baseline with Bayesian Information Criterion (BIC). Lower BIC typically indicates a better model fit for the same number of parameters. However, since the number of parameters changes with the number of knowledge components created from the submissions, BIC values cannot be compared across domain models and are only meaningful when comparing to the Single-KC baseline.

4.3 Results

Following previous work, we generated learning curves and used them to evaluate our models. Both the AST and the Ontology KC models capture learning to some extent and show promising fitness to the learning curves, as shown in Figure 2 through the decreasing error rates across multiple opportunities. We can also note that the ontology model has a steeper learning curve inclination than the AST model, pointing to a better model fit.

As explained in 4.2.2, the BIC values for the two different models cannot be directly compared due to their different number of parameters, so we present the Single-KC baseline result for each

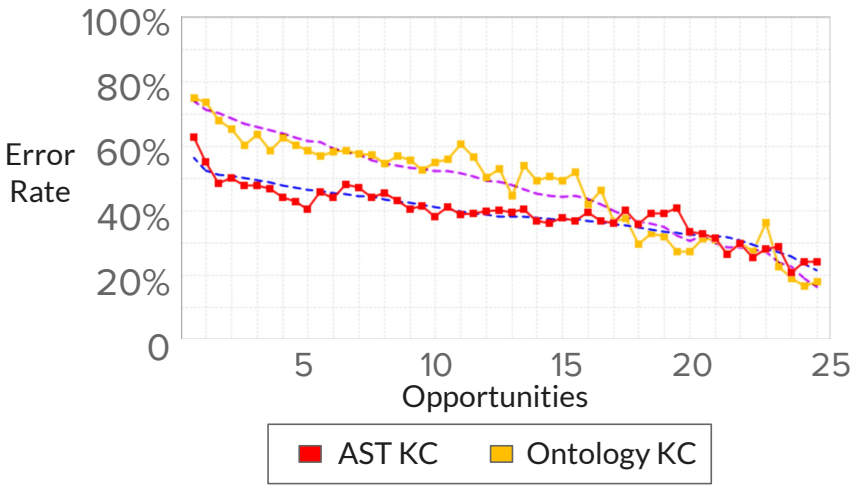


Figure 2: Averaged learning curves for AST KCs (red) and Ontology KCs (orange) and predicted learning curves (dashed lines) computed using Additive Factor Model (AFM).

of the models. Table 1 shows that both models demonstrate better model fit (BIC) than their respective baseline counterparts (Single-KC models). Moreover, the ontology model achieves a larger BIC improvement over its baseline.

Furthermore, Table 1 depicts how the ontology model achieves considerably lower RMSE values when predicting the error rates of new *students* in new opportunities (0.318) and of new *problems* (0.315). Thus, the combined results of learning curves, BIC and RMSE answer our research question by demonstrating that the student model that uses KCs informed by expert knowledge has a higher fitness and predictive power than its AST-based counterpart.

5 Conclusion and Future Work

In this paper, we combine a recent framework of KC attribution for the extraction task with an expert-driven KC modeling approach through an ontology. We compared our approach with a standard AST-based method to answer our research question and found that the ontology KCs achieved better results in fitness and predictive power. Besides the positive numerical results, the Ontology-based model has an inherent advantage in explainability - it makes use of an ontology designed by experts and aligned with the established domain's knowledge. Therefore, we believe these KCs can easily be integrated into user-friendly visualization tools to instructors and students. In addition, our framework formalizes the way to compare models in computing education, serving as baselines for future work for this task. Our work addresses the challenge of modeling user performance in a complex educational domain, and provides conceptualization for more work in the domain of student modeling in computing education.

As the work is preliminary, we still have limitations: 1) our experiment has been preliminary, and so far we have only worked on the CodeWorkout dataset, which is a common benchmark in CS education research [6, 11, 30]. We aim to include a broader coverage of datasets in the future. 2) A key assumption is that KCs are associated with nodes of a tree representation in programs, while some recent research indicates that KCs could be a set of nodes [13] or problems [31]. 3) Our pre-processing pipeline did not include some of the recent methods for filtering abnormal behavior from authentic student interactions [10]. Combining our modeling approach with these developing techniques could yield better modeling success.

In the future, while addressing all the limitations in the work, our goal is to further extend the work to reduce noise in our model. To achieve that goal, some possibilities are allowing other types of KCs (e.g., KCs involving multiple nodes) for a better-covered set of extracted KCs, merging related KCs (e.g. sum and subtraction), limiting the KCs considered in each activity to discard the ones less relevant to that activity (e.g. variable initialization), or splitting KCs to capture multiple levels of student knowledge (e.g. iterating on an integer range, on chars from a string, on items from an array).

Table 1: Quantitative metrics for AST and Ontology KCs: Ontology KCs show a higher improvement over baseline compared to AST, and achieve much lower prediction errors.

KC Model	BIC	RMSE (Stud)	RMSE (Prob)
AST-KC	118,504.44	0.478	0.424
Single-KC-Ast	125,301.28	0.494	0.446
Ontology	**181,749.20**	**0.318**	**0.315**
Single-KC-Ont	396,224.91	0.498	0.498

Acknowledgments

We thank the organizers of the 2024 LearnLab Summer School and sponsors from the SPLICE project (NSF Awards #2213789 #2213790 #2213791 #2213792).

References

[1] Vincent Aleven and Kenneth R Koedinger. 2013. Knowledge component (KC) approaches to learner modeling. *Design Recommendations for Intelligent Tutoring Systems* 1 (2013), 165–182.

[2] David Azcona, Yancy Vance Paredes, Sharon I-Han Hsiao, and Thomas Price. 2019. The 1st CSEDM Data Challenge. Retrieved January 29, 2025 from https://sites.google.com/asu.edu/csedm-ws-lak-2019/accepted-papers

[3] Hao Cen, Kenneth Koedinger, and Brian Junker. 2006. Learning Factors Analysis – A General Method for Cognitive Model Evaluation and Improvement. In *Intelligent Tutoring Systems (Lecture Notes in Computer Science)*, Mitsuru Ikeda, Kevin D. Ashley, and Tak-Wai Chan (Eds.). Springer, Berlin, Heidelberg, 164–175.

[4] Mehmet Arif Demirtaş, Max Fowler, Nicole Hu, and Kathryn Cunningham. 2024. Validating, Refining, and Identifying Programming Plans Using Learning Curve Analysis on Code Writing Data. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (Melbourne, VIC, Australia) *(ICER '24)*. Association for Computing Machinery, New York, NY, USA, 263–279.

[5] Mehmet Arif Demirtas, Max Fowler, and Kathryn Cunningham. 2024. Reexamining Learning Curve Analysis in Programming Education: The Value of Many Small Problems. In *Proceedings of the 17th International Conference on Educational Data Mining*. 53–67.

[6] Zhangqi Duan, Nigel Fernandez, Alexander Hicks, and Andrew Lan. 2024. Test Case-Informed Knowledge Tracing for Open-ended Coding Tasks. *arXiv preprint arXiv:2410.10829* (2024).

[7] John Edwards, Kaden Hart, Raj Shrestha, et al. 2023. Review of csedm data and introduction of two public cs1 keystroke datasets. *Journal of Educational Data Mining* 15, 1 (2023), 1–31.

[8] Stephen Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: short programming exercises with built-in data collection. In *Proceedings of the 2017 ACM conference on innovation and technology in computer science education*. 188–193.

[9] José P. González-Brenes and Yun Huang. 2015. Your model is predictive— but is it useful? Theoretical and Empirical Considerations of a New Paradigm for Adaptive Tutoring Evaluation. In *the 8th International Conference on Educational Data Mining (EDM 2015)*, Olga Santos, Jesús G. Boticario, Cristobal Romero, Mykola Pechenizkiy, Agathe Merceron, Piotr Mitros, José María Luna, Cristian Mihaescu, Pablo Moreno, Arnon Hershkovitz, Sebastian Ventura, and Michel Desmarais (Eds.). http://www.educationaldatamining.org/EDM2015/uploads/papers/paper_113.pdf

[10] Alex Hicks, Yang Shi, Arun-Balajiee Lekshmi-Narayanan, Wei Yan, and Samiha Marwan. 2024. An Approach to Detect Abnormal Submissions for CodeWorkout Dataset. *arXiv preprint arXiv:2407.17475* (2024).

[11] Muntasir Hoq, Yang Shi, Juho Leinonen, Damilola Babalola, Collin Lynch, Thomas Price, and Bita Akram. 2024. Detecting ChatGPT-generated code submissions in a CS1 course using machine learning models. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 526–532.

[12] Roya Hosseini and Peter Brusilovsky. 2013. JavaParser: A fine-grain concept indexing tool for java problems. In *CEUR Workshop Proceedings*, Vol. 1009. 60 – 63. https://d-scholarship.pitt.edu/26270/

[13] Yun Huang, Christian D Schunn, Julio Guerra, and Peter Brusilovsky. 2024. Why Students Cannot Easily Integrate Component Skills: An Investigation of the Composition Effect in Programming. *ACM Transactions on Computing Education* 24, 3 (2024), 1–37.

[14] Kenneth R Koedinger, Ryan SJd Baker, Kyle Cunningham, Alida Skogsholm, Brett Leber, and John Stamper. 2010. A data repository for the EDM community: The PSLC DataShop. *Handbook of educational data mining* 43 (2010), 43–56.

[15] Kenneth R. Koedinger, Albert T. Corbett, and Charles Perfetti. 2012. The Knowledge-Learning-Instruction Framework: Bridging the Science-Practice Chasm to Enhance Robust Student Learning. *Cognitive Science* 36, 5 (2012), 757–798.

[16] Kenneth R. Koedinger, Elizabeth A. McLaughlin, and John C. Stamper. 2012. Automated Student Model Improvement. In *Proceedings of the 5th International Conference on Educational Data Mining, Chania, Greece, June 19-21, 2012*. 17–24.

[17] Bassam Mokbel, Sebastian Gross, Benjamin Paaßen, Niels Pinkwart, and Barbara Hammer. 2013. Domain-Independent Proximity Measures in Intelligent Tutoring Systems. In *Proceedings of the 6th International Conference on Educational Data Mining, Memphis, Tennessee, USA, July 6-9, 2013*, Sidney K. D'Mello, Rafael A. Calvo, and Andrew Olney (Eds.). International Educational Data Mining Society, 334–335. http://www.educationaldatamining.org/EDM2013/papers/rn_paper_68.pdf

[18] Andy Nguyen, Christopher Piech, Jonathan Huang, and Leonidas Guibas. 2014. Codewebs: scalable homework search for massive open online programming courses. In *Proceedings of the 23rd international conference on World wide web*. 491–502.

[19] Anh-Tu Phuong Nguyen, Van-Dung Hoang, et al. 2024. Development of Code Evaluation System based on Abstract Syntax Tree. *Journal of Technical Education Science* 19, Special Issue 01 (2024), 15–24.

[20] Rose Niousha, Muntasir Hoq, Bita Akram, and Narges Norouzi. 2024. Use of Large Language Models for Extracting Knowledge Components in CS1 Programming Exercises. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*. 1762–1763.

[21] Stephen Nutbrown and Colin Higgins. 2016. Static analysis of programming exercises: Fairness, usefulness and a method for application. *Computer Science Education* 26, 2-3 (2016), 104–128.

[22] Philip I. Pavlik, Hao Cen, and Kenneth R. Koedinger. 2009. Performance Factors Analysis –A New Alternative to Knowledge Tracing. In *Proceedings of the 2009 Conference on Artificial Intelligence in Education: Building Learning Systems That Care: From Knowledge Representation to Affective Modelling*. IOS Press, NLD, 531–538.

[23] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 153–160.

[24] Thomas Price and Yang Shi. 2021. The 2nd CSEDM Data Challenge. Retrieved January 29, 2025 from https://sites.google.com/ncsu.edu/csedm-dc-2021/

[25] Thomas W. Price, Yihuan Dong, Rui Zhi, Benjamin Paaßen, Nicholas Lytle, Veronica Cateté, and Tiffany Barnes. 2019. A Comparison of the Quality of Data-Driven Programming Hint Generation Algorithms. *International Journal of Artificial Intelligence in Education* 29, 3 (Aug. 2019), 368–395.

[26] Thomas W Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M Kazerouni, Brett A Becker, Andrew Petersen, Luke Gusukuma, Stephen H Edwards, et al. 2020. Progsnap2: A flexible format for programming process data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 356–362.

[27] Bethany Rittle-Johnson and Kenneth R Koedinger. 2005. Designing knowledge scaffolds to support mathematical problem solving. *Cognition and Instruction* 23, 3 (2005), 313–349.

[28] Kelly Rivers, Erik Harpstead, and Ken Koedinger. 2016. Learning Curve Analysis for Programming: Which Concepts Do Students Struggle With?. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM Press, New York, NY, USA, 143–151.

[29] Yang Shi, Min Chi, Tiffany Barnes, and Thomas Price. 2022. Code-DKT: A Code-based Knowledge Tracing Model for Programming Tasks. In *Proceedings of the 15th International Conference on Educational Data Mining*, Antonija Mitrovic and Nigel Bosch (Eds.). International Educational Data Mining Society, Durham, United Kingdom, 50–61.

[30] Yang Shi, Min Chi, Tiffany Barnes, and Thomas Price. 2024. Evaluating Multi-Knowledge Component Interpretability of Deep Knowledge Tracing Models in Programming. In *Proceedings of the 17th International Conference on Educational Data Mining*. 288–295.

[31] Yang Shi, Robin Schmucker, Min Chi, Tiffany Barnes, and Thomas Price. 2023. KC-Finder: Automated Knowledge Component Discovery for Programming Problems. In *International Educational Data Mining 2023*. ERIC.

[32] Yang Shi, Robin Schmucker, Keith Tran, John Bacher, Kenneth Koedinger, Thomas Price, Min Chi, and Tiffany Barnes. 2024. The Knowledge Component Attribution Problem for Programming: Methods and Tradeoffs with Limited Labeled Data. *Journal of Educational Data Mining* 16, 1 (2024), 1–33.

[33] George S Snoddy. 1920. An experimental analysis of a case of trial and error learning in the human subject. *Psychological Monographs* 28, 2 (1920), i.

[34] Kurt VanLehn. 1988. *Student models*. Lawrence Erlbaum Associates, Hillsdale, 55–78.

[35] Kurt VanLehn, Collin Lynch, Kay Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. 2005. The Andes Physics Tutoring System: Lessons Learned. *International Journal of Artificial Intelligence and Education* 15, 3 (2005), 147–204.

[36] Michael V Yudelson, Kenneth R Koedinger, and Geoffrey J Gordon. 2013. Individualized bayesian knowledge tracing models. In *Artificial Intelligence in Education: 16th International Conference, AIED 2013, Memphis, TN, USA, July 9-13, 2013. Proceedings 16*. Springer, 171–180.

[37] Yingbin Zhang, Juan D Pinto, Aysa Xuemo Fan, Luc Paquette, et al. 2023. Using Problem Similarity-and Order-based Weighting to Model Learner Performance in Introductory Computer Science Problems. *Journal of Educational Data Mining* 15, 1 (2023), 63–99.