Investigating Difficult Topics in a Data Structures Course Using Item Response Theory and Logged Data Analysis*

Eric Fouh
Department of Computer Science & Engineering
Lehigh University
Bethlehem, PA 18015
efouh@cse.lehigh.edu

Mohammed F. Farghally
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
mfseddik@vt.edu

Sally Hamouda
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
sallyh84@vt.edu

Kyu Han Koh
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
kyuhan@vt.edu

Clifford A. Shaffer
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
shaffer@vt.edu

ABSTRACT

We present an analysis of log data from a semester's use of the OpenDSA eTextbook system with the goal of determining the most difficult course topics in a data structures course. While experienced instructors can identify which topics students most struggle with, this often comes only after much time and effort, and does not provide real-time analysis that might benefit an intelligent tutoring system. Our factors included the fraction of wrong answers given by student, results from Item Response Theory, and the rate of model answer and hint use by students. We grouped exercises by topic covered to yield a list of topics associated with the harder exercises. We found that a majority of these exercises were related to algorithm analysis topics. We compared our results to responses given by a sample of experienced instructors, and found that the automated results match the expert opinions reasonably well. We investigated reasons that might explain the over-representation of algorithm analysis among the difficult topics, and hypothesize that visualizations might help to better present this material.

Keywords

Item Response Theory, learning analytics, eTextbooks, algorithm analysis, data structures and algorithms

*(Does NOT produce the permission block, copyright information nor page numbering). For use with ACM_PROC_ARTICLE-SP.CLS. Supported by ACM.

1. INTRODUCTION

Knowing what topics are challenging to students helps educators better allocate course resources. We present techniques to automatically determine topics that are most challenging based on student interactions within the OpenDSA eTextbook system [9, 10]. While experienced instructors can identify which topics students most struggle with, automated measures can be useful for a variety of reasons. 1) Identifying key topics takes a lot of time and effort on the part of instructors; 2) They can help instructors teaching new material or with a new approach; 3) They can be used by an intelligent tutoring system (ITS) to automatically direct more instruction to a topic; and 4) They can help find, confirm, and quantify relationships and provide new insights that might be missed even by experienced instructors.

Our study focuses on a post-CS2 data structures and algorithms course (henceforth referred to as "CS3"). We used two approaches to identify difficult course topics. The first is Item Response Theory (IRT), a latent trait models (LTM) technique to analyze student responses to problems. LTM assumes that test performance can be predicted by specific traits or characteristics [13]. IRT provides a model-based association between item responses and the characteristic assessed by a test [7]. The second approach consisted of an analysis of student interactions with exercises to identify harder exercises. We investigated the incidence of guessing, the use of hints, and the level of interactions with embedded model answers by students when solving exercises.

We found that the most difficult topics in the CS3 course are related to algorithm analysis. While this is not surprising to us, we also investigated possible reasons that might explain the topics' difficulty. Based on our study, we present some suggestions on how to make such topics more accessible to students.

2. RELATED WORK

IRT [19] examines test behavior at the item level, and provides feedback on the relative difficulty of the various ques-

tions. Many IRT models have been developed with the assumption of 0 or 1 assigned to each response. We adopted the one parameter (1PL) or Rasch model [16] to characterize items and examinees. In 1PL, the probability of a positive response from a student is a function of item difficulty and is modeled as $P_i(\theta) = \frac{\exp(\theta - b_i)}{1+\exp(\theta - b_i)}$. P_i is the probability of a correct response to item i. θ refers to the latent trait (this is often called *ability*) assessed by the items being analyzed. b_i represents the difficulty of item i.

IRT has been used to evaluate students' coding abilities in an introductory programming course [3]. The authors used students' code scores to build a 1PL Rasch model. They found that students with previous knowledge had a statistically significant higher performance than students with no previous knowledge [3]. IRT was also used to analyze midterm exam questions for an introductory CS course [18]. The goal was to improve the assessment for future semesters by studying questions' item characteristic curves. IRT has been used for problem selection and recommendation in ITS [14]. The authors built a model based on a combination of IRT and collaborative filtering to automatically select problems.

We know of few efforts to identify difficult topics in CS3 courses, as most such work typically has focused on introductory courses [5, 6, 11]. Brusilovsky et al [4] sent a questionnaire to CS educators asking them to report topics that they consider critical to learn, as well as topics that are hard to learn (for students) and hard to teach (for instructors). Instructors' ($n = 61$) five most difficult-to-learn topics included pointers, recursion, polymorphism, memory allocation, and parameter passing. The five most difficult to teach topics included recursion, pointers, error handling, algorithms, and polymorphism. Many of these topics are covered in CS3, but it is typically not the first time that students will have seen them.

3. EXERCISE ANALYSIS

OpenDSA provides a collection of online, open-source tutorials that combine textbook-quality text with algorithm visualizations, randomly generated instances of interactive examples, and exercises to provide students with unlimited practice. Content within OpenDSA is organized into modules, each focusing on a specific topic such as Quicksort or Closed Hashing. The modules contain a wide variety of exercises. Some require that the student manipulate a data structure to show the changes that an algorithm would make on it. We refer to these as "proficiency exercises" (PE exercises). This type of exercise was pioneered in the TRAKLA2 system [15]. OpenDSA uses the Khan Academy (KA) exercise framework [1] to provide multiple choice, T/F, and short answer exercises. We also use the KA framework to implement simpler proficiency exercises.

We studied 143 student participants enrolled in a CS3 course at Virginia Tech during Fall 2014. OpenDSA was used as the main textbook, and students had until the end of the semester to complete the OpenDSA exercises. OpenDSA exercises accounted for 20% of the course final grade.

[1] `http://github.com/Khan/khan-exercises`

3.1 Analysis of correct answer ratios

Our goal is to assign a value to each OpenDSA exercise in terms of "relative difficulty". We seek to find which exercises are relatively difficult for average ability students. From this, we hope to deduce which topics are most difficult for students. This in turn might lead us to refocus our instructional efforts, or come up with new interventions and presentation approaches. Unfortunately, it is not a simple matter to tell whether a question is difficult. OpenDSA works on a mastery-based system, meaning that students can repeat a question until they get it correct. As a result, most students earn full credit on almost all exercises. To confuse the situation further, as is typical with online courseware, some exercises can be "gamed" [1]. In our case, this happens when students repeatedly reload the current page until they get an easier problem instance to solve (though the system is implemented in ways to discourage other forms of guessing on any given question [9]). For these reasons, we cannot simply count how many students got an exercise correct. Instead, we developed alternative definitions for difficulty.

We analyzed OpenDSA exercises with respect to the ratio of correct to incorrect answers as a measure of exercise difficulty, that is, harder exercises should show a lower correct attempt ratio. To assess student performance, we use the fraction $r = \frac{\#\text{of correct attempts}}{\#\text{of total attempts}}$. For each exercise, we compute the difficulty level (dl) as $dl = 1 - \frac{\sum_{i=1}^{n} r_i}{n}$ where n is the number of students and r is the ratio of correct attempts. Similar metrics have been used previously to assess exercise difficulty. In [2], the authors used "how many attempts it takes for a student to determine the correct answer once they have made their initial mistake" as a measure of exercise difficulty for logic exercises. History of attempts coupled with IRT was also used in [17] to estimate exercise difficulty for an ITS.

We ranked the exercises by their dl and grouped them into quartiles. Dl scores ranged from 0 to 0.72. Exercises in the 4^{th} quartile ($dl > 0.25$) consist mainly of exercises covering concepts related to algorithm analysis (22 out of 26 in that quartile), and one was a code writing question. Exercises in the 3^{th} quartile ($0.13 \leq dl \leq 0.25$) covered mainly (14 out of 25) the mechanics of algorithms or data structures. Ten of these exercises covered course concepts. Exercises in the 2^{nd} quartile ($0.05 \leq dl < 0.13$) covered mainly (23 out of 25) the mechanics of algorithms or data structures. The other two were summary exercises covering lists and the introduction chapter. All exercises in the 1^{st} quartile ($dl < 0.05$) covered algorithms or data structures mechanics. These results indicate that students did not seem to have difficulty completing tasks related to the behavior and the mechanics of algorithms and data structures. They seem to have the hardest time mastering algorithms analysis concepts.

3.1.1 IRT analysis

To perform IRT analysis we must dichotomize the answers. We awarded 1 point for $r \geq 0.75$ and 0 point for $r < 0.75$. We analyzed each chapter independently, considering all exercises in a chapter as part of an assignment. We used R statistical software (ltm package) and built a 1PL model for our investigation. For each OpenDSA chapter, we computed the item characteristic curves (ICC), item information curves

(IIC), and test information curves (TIF). For each curve, the $x-axis$ represents the students' ability from -4 to 4, where $x = 0$ means average ability. ICC shows the probability of a score of 1, given a student's ability. IIC shows how much information each exercise can tell us about a students' ability. TIF shows how reliable the overall test (or a collection of exercises) is at distinguishing students with different ability. Harder tests would better distinguish between students with above-average ability, while easier tests would better distinguish between students with below-average ability.

An ICC graph lets us see the probability of getting a score of 1 for students with average ability. Harder exercises will have $P_i(0) < 0.5$. In Figure 1, we see that for three of the most difficult exercises, the probability that a student with average ability will get a score of 1 is less than 0.5, indicating that those exercises distinguish students with average ability from those with above average ability, but do little to distinguish weaker from average students. On the other hand, an easy question on the binary search algorithm has a graph $P_i(\theta) = 1$. Thus it does not give us any information about students' ability. The curves for the easier exercises shown in Figure 2 show differences between students with below average ability in contrast with average and above average ability ($\theta \geq 0$). Another possible interpretation of this result is that these exercises are relatively good at differentiating students who studied from those who did not. The TIF graph is a combination of all IIC curves, and indicates the overall performance of the test.

Algorithm analysis chapter exercises: Most students did not fare well on exercises in the introductory chapter on algorithm analysis, as shown in Figures 1 and 2. Thus these exercises gave us information about which students have above-average ability.

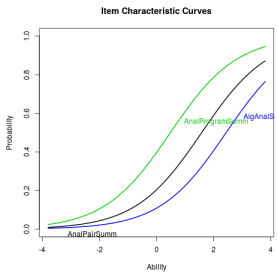

Figure 1: Algorithm analysis ICC

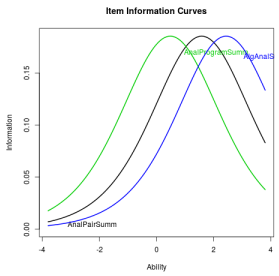

Figure 2: Algorithm analysis IIC

Linear Structures exercises: These students were already familiar with linear structures, since these are taught in prerequisite courses. Students could easily get a score of 1 by our difficulty measure for most problems in this chapter, and so help to identify students with below average ability ($x < 0$). However, three exercises appeared to be not so easy for students. They covered list overhead concepts (a new topic for them), array list concepts, and a small programming exercise. Students who did poorly (bottom quarter) on these exercises scored an average 65 on Midterm 1 compared to 76 for the rest of the class (a significant difference at $\alpha = 0.05$). They received an average score of 73 on Midterm 2 compared to 79 for the rest of the students (a significant difference at $\alpha = 0.05$). They scored an average

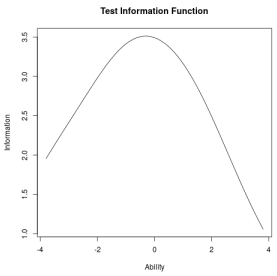

Figure 3: Sorting TIF

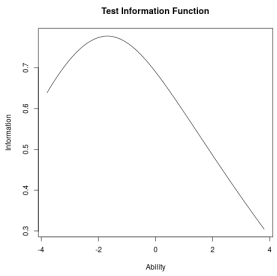

Figure 4: Binary trees TIF

of 106 on the final, compared to 112 for the rest of the class, not a statistically significant difference.

Sorting exercises: The sorting chapter has the most exercises, with varying difficulty levels. Summary exercises covering more advanced sorting algorithms (quicksort, radix sort, mergesort, and heapsort) seemed to provide more information about students with above average ability ($x > 0$). Overall, the sorting chapter exercises seemed to provide a good range of easy to difficult exercises, and provided good information to distinguish between students with different ability levels (TIF curve maximum at ability $= 0$).

Binary tree exercises: Binary trees are typically first introduced in CS2 courses. Only three exercises appeared to be difficult for students. These involved writing a recursive function to traverse a tree, questions on heaps, and computing tree space overhead. Exercises in this chapter provided us with information about students with below average ability (TIF curve maximum at ability < 0).

Hashing and graph exercises: As with other topics, proficiency exercises were relatively easy for the students, while questions on the concepts and analysis were more difficult. The graph chapter only had algorithm proficiency exercises and so were not challenging to students. Therefore, the exercises gave us information to distinguish students with low ability (TIF curve maximum at ability > 0).

We identified 21 (out of 100) exercises with IIC maximum at ability ≥ 0. 19 of those exercises cover the algorithm analysis portions of the different topics. The IRT analysis for all OpenDSA exercises given to students enrolled in the course revealed the following. Across chapters, exercises related to algorithm analysis had IIC curve maximums at ability< 0. Exercises that required students to solve small programming problems also scored as "difficult" by our metric because they tended to require multiple submissions to complete.

3.2 Using Hints and Guessing

Our analysis metric for "incorrect attempts" does not differentiate between using a hint or submitting an incorrect answer. So we looked in more detail at the types of incorrect submission for each exercise. We analyzed OpenDSA exercises with respect the the number of hints used, and the appearance of a trial-and-error strategy to "guess" the answers. Harder exercises are expected to display a higher rate of hints use and/or trial-and-error.

Exercises using the KA framework (multiple choice, T/F, fill-in-the-blank, and one-step proficiency exercises) generate a series of question instances on the topic. The student must get a certain number correct (typically five) to complete the exercise. One point is deducted from the student's credit toward this requirement when they submit an incorrect answer, to discourage guessing. Students can also take one or more hints that explain the answer to the question. In this case, the attempt is not graded (no point is awarded or deducted toward the threshold).

To analyze exercises based on students' hint use, we computed the hint ratio $hr = \frac{\#\text{of hints used}}{\#\text{of total attempts}}$ for each KA exercise. Four exercises are potential outliers as measured by hr, related to quicksort, hashing, calculating overhead for trees, and calculating overhead for lists. To analyze exercises based on the rate of trial-and-error, we calculated the incorrect ratio $ir = \frac{\#\text{of incorrect answers}}{\#\text{of total attempts}}$ for each KA exercise. Inspecting exercises in the fourth quartile (exercises in the highest 25% incorrect ratio), we found that they are related to the topics algorithm analysis, heaps, quicksort, radixsort, shellsort, and heapsort.

The seven exercises shown in Table 1 had high hint or high incorrect answer ratios. They relate to topics covering mathematical background and runtime analysis of quicksort, hashing, and shellsort. 45% of students heavily (third quartile and up for all exercises) used hints, and provided many incorrect answers when solving these seven exercises. We found that most exercises with low incorrect answer and hint ratios are for stacks, arrays, and lists. These are topics that most students know from previous courses. When using high rate of hint use as a measure of exercise difficulty, we found that exercises related to algorithm analysis and mathematics topics appeared to be more "difficult". Algorithm analysis was also identified as difficult by IRT analysis.

Table 1: *IR* and *HR* for difficult exercises

Exercise	*hr*	*ir*	Topic
ListOverhead	0.93	0.6	List Overhead Analysis
TreeOverheadSumm	0.78	0.73	Tree Overhead Analysis
QuicksortSumm	0.32	0.67	Quicksort Analysis
AlgAnalSumm	0.24	0.77	Algorithm Analysis
MthBgSumm	0.25	0.63	Mathematical background
ShellsortSumm	0.16	0.61	Shellsort
QuicksortPartitionPRO	0.27	0.58	Quicksort's partition

3.3 Model Answer Use and Exercise Reset

Algorithm proficiency exercises require students to reproduce the major steps of an algorithm. Proficiency exercises come with a "model" answer that can be viewed at any time (though doing so voids that problem instance for credit, and so the student must do another problem instance). The student can click a "reset" button to get a new problem instance. We analyzed OpenDSA exercises with respect to model answer use and "reset" as a measure of (exercise) difficulty. Students are expected to reset or view model answers more for harder exercises. For each proficiency exercise, we analyzed the number of student attempts and the frequency of student access to the model answer dialog. Our analysis showed that heap and quicksort exercises have a model answer view rate approaching or exceeding 50%, which is greater than the mean ($\mu = 25.5$) plus one standard deviation ($\sigma = 16$) of the rates distribution. This finding indicates that these exercises are relatively more difficult compared to other proficiency exercises.

We also investigated student activity log data to learn when students access the model answer box by computing: (i)% of students who tried the exercise, then opened the model answer dialog before they received enough points to get credit for the exercise; % of students who opened the model answer dialog before attempting the exercise; and % of students who opened the model answer dialog after they received proficiency credit for the exercise.

A model answer shows how to solve a problem with less detail, while slideshows and visualizations (available to the students before attempting the exercise) carefully explain the concepts. We tried to determine if students use model answers as a substitute for viewing slideshows and visualizations. For the heap exercises, we found that about 35% of the students attempted an exercise before going through any slideshow included in the section. This result indicates that students might be using model answers (on certain topics) because they overlook and/or rush through visualizations when studying. We found that a majority of students (62% on average) opened the model answer before attempting the heap exercises. For the quicksort exercise, we found that most students (67%) opened the model answer dialog after an incorrect attempt. 24% of students opened the model answer dialog before attempting the exercise.

For each proficiency exercise, we looked at the percentage of students who returned back to solve the exercise after receiving proficiency credit. We found that exercises with a high model answer view rate have a lower level of post-proficiency attempts. 27% of students solved them post proficiency, compared to almost 50% for other exercises. This is somewhat surprising, as students presumably use an exercise post-proficiency in order to study the material for exams. We might have expected the most difficult exercises to be targets for additional study.

We computed the ratio of correct attempts over number of reset button clicks, and the ratio of all attempts over number of reset button clicks. The correlation between the two ratios was $r^2 = 0.99$. Exercises with lowest ratios (bottom 25%) were related to quicksort, heaps, shellsort, and binary trees topics. When using number of model answer views and use of reset button as measures of exercise difficulty, we found that the hardest exercises are related to the topics of heaps and quicksort. These exercises have higher use of model answers, higher exercise reset rates, and lower levels of post-proficiency attempts compared to other exercises. We note that proficiency exercises cover only algorithm mechanics, and so do not test students on more theoretical concepts. Thus, this analysis is only comparing the relative difficulty

of understanding the mechanics of various algorithms, and so does not address the question of the relative difficulty of algorithm analysis versus algorithm mechanics.

4. INSTRUCTOR SURVEY RESULTS

To validate our process, we compared the results of automated analysis with opinions of course instructors. To that end, we distributed a survey to the CS education community via the SIGCSE mailing list. We asked respondents: (i) how long they have been teaching a post-CS2 course on Data Structures and Algorithms; (ii) what topics from such a course are the most difficult for students to understand; and (iii) what topics from such a course are the most difficult to teach. We received 23 responses with a mean teaching career of 16 years (median 15 years). Since a concept can be defined using different terms, we grouped answers that we considered to refer to the same topic. The result was 12 topics considered most difficult for students to understand, and 8 topics most difficult to teach. Table 2 shows the top 6 difficult topics to learn and to teach. Among the top topics considered hard for students, only trees and heaps are not also present in the list of hard topics to teach.

Table 2: Summary of survey responses

Topic	**N**	**%**
Most difficult topics for students		
Dynamic programming	7	18
Algorithm analysis	6	15
OOP & Design	6	15
Recursion	4	10
Trees, Heaps	3	7
Proofs	3	7
Most difficult topics to teach		
Complex algorithms	8	30
OOP & Design	4	15
Proofs	4	15
Algorithm analysis	3	11
Recursion	3	11
Dynamic programming	2	7

Dynamic programming had the most votes as difficult for students, but we note that most CS3 courses to not cover this in depth. Algorithm analysis received the next highest number of votes. Our IRT and log analyses also identify algorithm analysis as a hard topic for students. Instructors mentioned students' lack of proficiency in mathematics as a major reason why algorithm analysis proves hard. Instructors wrote "mathematical sophistication is the issue here", and "because students are afraid of math". Our analysis of use of trial-and-error also revealed that students are not at ease with mathematics topics. To explain why algorithms analysis is hard to teach, one instructor wrote "I still do not have good instructional material". That reason was also used for other topics like graphs and design. Heaps is another topic that was identified as hard both by our analysis and by instructors. In general, the survey responses correspond fairly well to our automated process.

5. ALGORITHM ANALYSIS IS HARD

Our analysis shows that exercises related to algorithm analysis are harder than exercises covering algorithm mechanics. It also reveals that students might have some difficulty with heaps and quicksort. Algorithm analysis is of particular interest since a main goal of CS3 is to teach students how to analyze algorithms, in order to design efficient software solutions. That is why algorithm analysis sections are present in almost all topics covered in the course. Careful analysis of the data logs reveals certain behaviors by students that could explain why students struggle with these concepts.

5.1 Not spending enough time

We analyzed interaction logs from use of OpenDSA at three universities (Virginia Tech, University of Texas El Paso, and University of Florida). Table 3 shows estimated reading time for the algorithm analysis material from three sorting modules (Insertionsort, Mergesort, and Quicksort). More than 74% of students spent less than one minute on the analysis material for each of the three modules. Based on this result, we believe that most of the students are not reading the analysis material.

Table 3: Time reading algorithm analysis material

University	Module	N	μ(sec)	% < 1 min
VT	Insertionsort	98	63.57	74.48
	Mergesort	96	39.79	78.12
	Quicksort	92	64.71	73.91
UTEP	Insertionsort	26	49.84	80.76
	Mergesort	22	41.45	77.27
	Quicksort	16	16.18	93.75
Florida	Insertionsort	53	40.39	84.90
	Mergesort	44	18.63	95.45
	Quicksort	39	26.12	92.30
All	Insertionsort	177	54.6	78.52
	Quicksort	147	49.2	80.94

86% of students responding to a survey indicated that it is easier for them to understand how an algorithm works than to analyze the running time for that algorithm. Quotes include: *"determining asymptotic running time because it is harder to visualize and less intuitive"*, *"Complexities are confusing and math-like"*, *"I think understanding how an algorithm work is easy. It is the style of presentation"*, *"How the algs work. It is dependent on material, also abstract stuff is harder for me to understand"*.

78% of students who are more comfortable with dynamics attributed this to the material, as algorithm analysis is abstract and requires familiarity with mathematical notations. The other 22% attributed this to how concepts are presented in OpenDSA (dynamics are presented using visualizations, analysis is presented mostly through text). Quotes regarding the usefulness of OpenDSA's algorithm analysis content include: *"Not any more useful than any other book"*, *"Not as much as learning the algorithms themselves, but I felt it was as useful as any resource could be on the topic"*, *"Yes, but not as much as understanding the algorithms"*, *"It could have been more interactive with showing why the analysis was the way that it was"*, *"I found it much more useful on Data structures. Algorithm analysis doesn't benefit quite as much from animations"*, *"It was very detailed and kind of hard to follow"*, *"I'd like there to be more visuals for analysis"*. Clearly respondents did not find the OpenDSA material on algorithm analysis different from other textbooks on that topic. This is not what they expected from OpenDSA, whose goal is to present content interactively.

5.2 Content presentation not engaging

When students were asked to provide suggestions for improving presentation of the analysis material in OpenDSA, most indicated they were expecting a more interactive presentation in the form of visualizations. Quotes include: *"Visualizations definitely help."*, *"I think making the clickthrough pictures into actual animations would be nice"*, *"more animation, the visualizations are great!"*, *"more visualizations is always good"*, *"Visualizations always help :)"*, *"visualizations showing each step of analysis would help"*, *"an animation will make a much bigger difference."*

6. CONCLUSION AND FUTURE WORK

Educational resources are rapidly moving online. As eTextbooks and interactive exercises become more prevalent, techniques to automatically discover the most difficult topics for students will become increasingly important. Doing so allows both instructors and designers of instructional content to focus their resources on the most difficult topics. Perhaps resolving the difficulty might be as simple as fixing a buggy exercise. But more generally, we find that specific concepts are truly hard. By examining the topic in detail, including its method of presentation, we might uncover better approaches to instruction, leading to better outcomes.

To illustrate, we are working on addressing the issues raised by students regarding the lack of visual presentation for algorithm analysis material in OpenDSA. Inspired by the concept of visual proofs [12], a set of Algorithm Analysis Visualizations (AAVs) were implemented for OpenDSA sorting modules [8]. We have collected preliminary data with two small classes using the sorting analysis visualizations. Summary results were collected for two modules teaching Insertion Sort and Quicksort. A Kruskal Wallis tests showed a significant difference ($p < 0.01$) between the time spent for text versus visualizations for these two modules. This indicates that students spend more time on the material when presented as visualizations. Having proved the value of the concept, we will continue to expand on this approach.

7. ACKNOWLEDGMENTS

We gratefully acknowledge the support of the National Science Foundation under Grants DUE-1139861, IIS-1258571, and DUE-1432008.

8. REFERENCES

[1] R. Baker, A. Corbett, and K. Koedinger. Detecting student misuse of intelligent tutoring systems. In *Proceedings of the 7th International Conference on Intelligent Tutoring Systems*, pages 531–540, 2004.

[2] D. Barker-Plummer, R. Cox, and R. Dale. Student translations of natural language into logic: The grade grinder corpus release 1.0. In *Proceedings of the 4th international conference on educational data mining*, pages 51–60, 2011.

[3] M. Berges and P. Hubwieser. Evaluation of source code with item response theory. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 51–56, 2015.

[4] P. Brusilovsky, J. Grady, M. Spring, and C.-H. Lee. What should be visualized?: Faculty perception of priority topics for program visualization. *SIGCSE Bulletin*, 38(2), June 2006.

[5] N. Dale. Content and emphasis in CS1. *SIGCSE Bulletin*, 37(4):69–73, Dec. 2005.

[6] N. B. Dale. Most difficult topics in CS1: Results of an online survey of educators. *SIGCSE Bulletin*, 38(2):49–53, June 2006.

[7] F. Drasgow and C. L. Hulin. Item response theory. *Handbook of industrial and organizational psychology*, 1:577–636, 1990.

[8] M. F. Farghally, E. Fouh, S. Hamouda, K. H. Koh, and C. A. Shaffer. Visualizing algorithm analysis topics. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, page 687, 2016.

[9] E. Fouh, D. A. Breakiron, S. Hamouda, M. Farghally, and C. A. Shaffer. Exploring students learning behavior with an interactive etextbook in computer science courses. *Computers in Human Behavior*, pages 478–485, December 2014.

[10] E. Fouh, V. Karavirta, D. A. Breakiron, S. Hamouda, S. Hall, T. L. Naps, and C. A. Shaffer. Design and architecture of an interactive etextbook–The OpenDSA system. *Science of Computer Programming*, 88:22–40, 2014.

[11] K. Goldman, P. Gross, C. Heeren, G. L. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. Setting the scope of concept inventories for introductory computing subjects. *Transactions on Computing Education*, 10(2):5:1–5:29, June 2010.

[12] M. T. Goodrich and R. Tamassia. Teaching the analysis of algorithms with visual proofs. In *SIGCSE Bulletin*, volume 30, pages 207–211, 1998.

[13] R. K. Hambleton and L. L. Cook. Latent trait models and their use in the analysis of educational test data. *J. of Educational Measurement*, 14(2):75–96, 1977.

[14] P. Jarušek and R. Pelánek. Analysis of a simple model of problem solving times. In S. Cerri, W. Clancey, G. Papadourakis, and K. Panourgia, editors, *Intelligent Tutoring Systems*, volume 7315 of *LNCS*, pages 379–388. Springer, 2012.

[15] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, September 2004.

[16] G. Rasch. *Probabilistic models for some intelligence and attainment tests.* Danmarks Pædagogiske Institut, 1960.

[17] G. Ravi and S. Sosnovsky. Exercise difficulty calibration based on student log mining. In *Proceedings of DAILE: Workshop on Data Analysis and Interpretation for Learning Environments*, 2013.

[18] L. A. Sudol and C. Studer. Analyzing test items: Using item response theory to validate assessments. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 436–440, 2010.

[19] W. J. van der Linden and R. K. Hambleton. *Handbook of modern item response theory.* Springer, 2013.