# Using Student Logs to Build Bayesian Models of Student Knowledge and Skills

Huy Nguyen and Chun Wai Liew

June 19, 2018

# Using Student Logs to Build Bayesian Models of Student Knowledge and Skills

Huy Nguyen
Lafayette College
nguyenha@lafayette.edu

Chun Wai Liew
Lafayette College
liewc@lafayette.edu

## ABSTRACT

Recent works on Intelligent Tutoring Systems have focused on more complicated knowledge domains, which pose challenges in automated assessment of student performance. In particular, while the system can log every user action and keep track of the student's solution state, it is unable to determine the hidden intermediate steps leading to such state or what the student is trying to achieve. In this paper, we show that this information can be acquired through data mining, along with the type, frequency and context of errors that students made. Our technique has been implemented as part of the student model in a tutor that teaches red-black trees. The system was evaluated on three semesters of student data. Analysis of the results shows that the proposed framework of error analysis can help the system in predicting student performance with good accuracy and the instructor in determining difficulties that students encounter, both individually and collectively as a class.

## Keywords

Data structures, Bayesian Learning, Error analysis

## 1. INTRODUCTION

An important goal in assessing student performance is to find out why the student makes certain errors, as it helps the instructor adapt the teaching style/materials accordingly to address the cause of such errors. With the rise of educational technology, it is now often the tutoring system that performs grading tasks in place of the instructor, thereby raising the need for an automated error analysis mechanism. Traditionally, tutoring exercises are often designed as multiple choice questions, where there is a single correct option, while the incorrect options are each worded in a way that targets a specific misconception (e.g., [6]). In this case, knowing which option a student picked is sufficient to infer why she made that decision. However, multiple-choice questions can be answered by pure guessing, and the options presented might not capture the full space of misconceptions that students have, especially if each decision is not a simple primitive choice. Furthermore, recent development of tutoring systems has moved on to more complicated knowledge domains, such as protein folding [2], programming language [7], and database [19]. These domains require students to engage in high level problem-solving tasks instead of simple multiple-choice and short-answer questions. In turn, they also pose challenges to the tutoring system in assessing student performance, namely (1) recognizing when the student is correct, (2) identifying the analyzing the errors made, and (3) predicting when a previous error might occur again.

Tree data structure exercises are an example of problems where the steps are best input graphically to show how the data structure is transformed at each step. Conventional question formats such as multiple-choice would therefore greatly constrain the student's answer and allow for the possibility of guessing. An ideal input mechanism, in this case, should allow the student to freely and easily specify the tree structure, reveal no clue or bias about the solution, and support automated assessment of student answer. In other words, it is to closely resemble a paper exam where students construct their answers from scratch.

The solution to an insertion/deletion tree problem is a sequence of transformations (steps) to be applied to the initial tree; alternatively, it can also be viewed as a list of trees, each resulting from applying a transformation to the tree before it. Determining if an answer is correct is straightforward - we simply check that the default solution's final tree matches that of the student's answer. If they differ, however, determining where and how the student made an error is much more difficult. The primary reason is that there can be multiple valid solutions, each with a different partial ordering of the same set of transformations. Furthermore, when unconstrained by the system, students also tend to combine several base (primitive) steps together into a macro-step, in which case their solution sequence can be shorter than the default solution yet still correct. Despite these difficulties, the assessment task plays an important role in both assigning partial credits to test submissions and informing the instructors about difficulties that students are facing, so that necessary interventions may take place.

This paper presents our approach in solving the assessment problem in the domain of red-black tree, a type of self-balancing binary search tree. In particular, based on analysis of the tutoring system's log data and student answers, we have devised a framework to identify and categorize the errors in their problem-solving process. The output is then used to construct a Bayesian student model, which predicts student performance throughout the tutoring session (i.e., whether the student's next answer will be correct or not). We show that categorizing the identified errors by not only their types but also contexts can help the model achieve good accuracy and provide insights into common patterns of problem-solving behavior in our chosen domain. The contexts can be identified by mining the logs and judiciously combining data to identify contexts that might be temporally connected. The mined data can help identify when temporally adjacent contexts affect student decisions and point out non obvious connections.

## 2. RED-BLACK TREES

A red-black tree is a self-balancing binary search tree with a number of properties which guarantee an $O(\log N)$ height when the tree has $N$ nodes [5]:

1. The nodes of the tree are colored either red or black.
2. The root node is always black.
3. A red node cannot have any red children.
4. Every path from the root to a null node contains the same number of black nodes.

Search in a red-black tree's operation is identical to that in a conventional binary search tree, while insertion and deletion are performed differently. The top-down algorithm to insert or delete a value from a red-black tree starts at the root and, at every iteration, moves down to the next node, which is a child of the current node. At each node, it applies one or more transformation rules; there are six rules used in insertion: *color flip, single rotate, double rotate, insert node, forward*, and *color root black*. Deletion involves another two, *switch value* and *drop rotate*. The role of these transformations is to change the tree in such a way that when the actual insertion (or deletion) is performed at the leaf node, in most cases no subsequent modifications to the tree are needed in order to preserve its properties. Other types of balanced trees also employ a similar approach. In our work we used red-black tree as an exemplar to evaluate our ideas and implementations, but they should be applicable to balanced trees in general.

In a standard curriculum, students learn about red-black trees right after finishing binary search tree, but often struggle because the tree transformations are quite complicated, especially on a medium-sized tree of more than 10 nodes. Furthermore, the insertion and deletion version of the same transformation (for example, *color flip*) operate differently, causing another source of confusion. A previous study on this domain by Liew & Xhakaj [15] found that red-black trees can be taught and learned effectively using a *granularity approach* - students should iteratively break down the problem into three steps of (1) identifying the current node, (2) selecting the applicable transformation, and (3) applying the selected transformation. Our tutoring system also follows the same approach.

## 3. RELATED WORK

There are well-studied advantages and disadvantages of both multiple-choice and free-response questions [10]. As the domain knowledge gets more complicated, it becomes more difficult to design multiple-choice tests that accurately reflect the student's level of understanding; on the other hand, free-response questions are not scalable because of the need for human graders. In practice, many intelligent tutoring systems opted for the middle ground by using a restricted language such as numerics for student answers. In this way, there is still a large solution space that makes guessing ineffective while the information derived from students' assessment is accurate enough to be used in constructing a student model. For example, physics tutors such as ANDES [4] and OLAE [16] teach college-level Newtonian mechanics by having students identify the forces acting on a physical object and express them in a system of equations.

Several past works have explored automated assessment in complex domains. For example, [3] uses an online judge system for an introductory programming course that is capable of detecting plagiarism and performing efficient, bias-free assessment. [17] constructs an adaptive grading system that can grade multiple and complex computer literacy assignments while being able to "learn" the correct and incorrect responses and add them to the rubric. Combining both human graders and computer graders, [8] introduces a collaboration framework that aims to minimize human effort in the domain of medical case analysis, using supervised machine learning.

Efforts have also been made to output not only a binary result (correct/incorrect) or numerical score, but also to provide reasonable feedback for both the students and the instructors. Many research works in the domain of introductory programming have been following this direction [9, 20, 21]. In other domains, [14] shows that in the PHYSICS-TUTOR system, where students enter algebraic equations as answer, it is possible to check for dimensional correctness and isolate errors by parsing the submitted answers into binary expression trees. Finally, [11] proposes using case-based reasoning to deliver past instructor feedback to new students who are solving a similar problem, which has been adapted in various tutoring systems.

Predicting student performance is one of the primary goals of student modeling. Traditionally, Bayesian network and its variations [1] are often used because of their accuracy and interpretability. This line of technique has been shown to be effective for tutoring systems that have no prior knowledge about their students, such as the ANDES physics tutor. Later on, ITSs are often deployed multiple times in successive semesters, and the data log from past student interactions can be analyzed by data mining techniques to better predict future students' performance. For instance, [12] builds a logistic regression model on the ANDES dataset to correctly identify 70% of the student's performance, while [18] uses pattern classifier and genetic algorithm to improve the tutoring system's prediction accuracy, which helps identifying weak students early on even in large classes.

In the domain of red-black trees, [15] was among the first tutoring systems developed. Its result shows that the granularity approach, which require students to follow explicit small steps, helped significantly improve their performance in insertion exercises. The system was built only for tutoring, while the tests were conducted on paper and evaluated by a human instructor. [13] proposes preliminary results in automating the test environments and grading with an algorithm that can detect the first error made by students in tree insertion questions.

## 4. THE RED-BLACK TREE TUTOR

Our tutoring system has three sections - the pre-test, the tutor, and the post-test. In the test sections, a typical insertion (deletion) problem for red-black trees involves inserting a sequence of numbers to a starting tree (or deleting from it). Students have to show the state of the tree after every insertion/ deletion; they are also encouraged to show any intermediate states (the trees that are created along the path to the solution). To this end, the test interface displays a "blank" binary tree canvas of 31 empty nodes. The student can click on any node to specify its value and color - submitting a tree is therefore equivalent to entering all of its nodes to the corresponding position in the tree canvas; nodes that are left empty are assumed to be null black nodes. The interface is designed to look like a sheet of paper with blanks to fill in - in this way, we ensure that (1) the tests do not provide any hints or clues as to what the desired answer would be, and (2) the student's answer is always in a format that can be interpreted and analyzed by the system.

In the tutoring section, students perform the same task of inserting to (or deleting from) a starting tree. However, a node-by-node modification of the current tree is not required; instead, students only need to select a node and the transformation to apply at that node from a drop-down list. The tutoring system has a solver module that can generate a solution for any problem and also check the correctness of the student's selection. If it is correct, the system will automatically apply the chosen transformation and update the information shown in the interface; otherwise, a message is displayed to the student indicating that the current selection is incorrect. We chose this approach based on the finding that learners often have difficulty identifying the transformations rather than applying them [15]; students also find the task of repeated application of the transformations tedious and time consuming.

## 5. PREDICTION OF STUDENT PERFORMANCE

In order for the system to be dynamic (i.e., to generate dynamic exercises that address an individual student's weakness), it needs to have knowledge of what the student knows and does not know at any given time. In the context of our tutor, the system should be able to predict whether the student's next answer is correct, based on her performance so far in the tutoring session and in the pre-test. To our knowledge there has not been any prior work on performance prediction in the domain of binary search tree. Therefore, to get a better sense of how well the student model performs, we implemented and evaluated three approaches.

### 5.1 Baseline prediction

Every time the student submits an answer in the tutoring session, the system predicts that the answer is correct with a fixed $p = 0.5$ probability. The performance of this method will serve as a baseline to compare with that of the next two methods.

### 5.2 Bayesian model with error contexts

We first analyze student answers in the pre-test and identify the first error made (if any) each time the student attempts to insert/delete a single node to/from a tree. Besides the type of error - incorrect node selection/incorrect transformation selection/ incorrect transformation application - and its location - how far did the student progress when the error was made - we are also interested in its context. In insertion exercises, an error context is the subtree surrounding the node at which the error occurred, which includes its parent and two children. In deletion exercises, the context subtree also contains the node's sibling and sibling's children. These definitions were devised based on the knowledge that (1) the transformation to select and apply at each node depends on the subtree surrounding it, and (2) even the same tree transformation may operate differently in different contexts, so it's important to recognize which specific context poses problems for the student.

We then construct a two-part Bayesian network using Bayesian Knowledge Tracing (BKT) [22] similar to that of the ANDES tutor [4]. This architecture is summarized in Figure 1. The domain-general network encodes long-term knowledge and represents the system's assessment of the student's rule mastery after the last performed exercise. It consists of two kinds of nodes: Rule node, which conveys the student's rule mastery in general, and Context-Rule node, which conveys rule mastery in a specific context. Both have as value a mastery probability $0 \leq p \leq 1$, while the conditional probability of each Context-Rule given its parent Rule is

$$P(\text{Context}_i \mid \text{Rule} = T) = 1,$$
$$P(\text{Context}_i \mid \text{Rule} = F) = \text{diff}_i,$$

where $\text{diff}_i$ is the difficulty of context $i$, determined by the number of errors in context $i$ divided by the total number of time that such context occurs (in the pre-test).

The task-specific network encodes the student's rule mastery in a specific exercise. We employ three kinds of nodes: Context-Rule, Fact and Rule-Application. Each Fact node expresses a property of the current tree, i.e., the current node is black or the parent node is red. These nodes represent the hypotheses that the student is aware of what to look for in the preconditions of the next step. The Rule-Application node has a boolean value, which is set to True if the student applies the rule correctly, and False otherwise. In essence, the system analyzes the current context, expressed by the Fact nodes, to bring up the corresponding Context-Rule node, whose probability value is used to predict the student answer's correctness. After the student submits the answer, the system records whether the prediction is right or wrong and updates the posterior value of the Context-Rule node, according to BKT. The rationale is that if the student previously made an error in a particular context, when that context shows up again in the current exercise, we would like to see whether the same error occurs. If no error is made, the student's mastery in this context

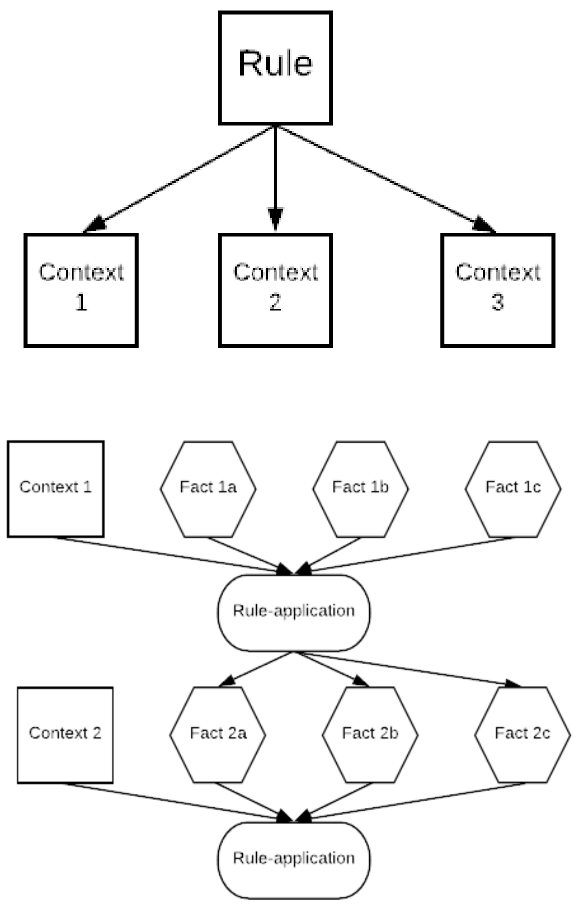

**Figure 1: The structure of the domain-general network (top) and task-specific network (bottom).**

has improved. This mechanism is expressed by the weight of the edge leading to each Rule-Application node:

$$P(\text{Rule-Application} = T \mid \text{all parents} = T) = 1 - P(S),$$
$$P(\text{Rule-Application} = T \mid \text{at least one parent} = F) = P(G),$$

where $P(S)$ and $P(G)$ are the slip and guess probabilities, which are part of the BKT parameters and set to a default value of 20%.

Once the student finishes an exercise, its task-specific network is discarded, but the context rule mastery probabilities are saved back to the domain-general network, so that they can be used as prior probabilities for future exercises.

## 5.3 Bayesian student model with extended error contexts

So far we have considered each transformation in isolation, but the nature of the solution to a red-black tree problem is a sequence of transformations, one following another. Our third approach experiments with the idea that the correctness of a student's answer may also depend on her previous answer. We perform pre-test analysis and Bayesian modeling as described in Section 5.2, but now the error context includes both the surrounding subtree and the previous transformation. With this distinction, there will be more contexts to analyze, and we would like to see how it affects the sytem's accuracy.

## 6. EVALUATION & RESULTS

We evaluated our approaches on four semesters of data from students in a computer science class at our institution. The semester enrollments are 20 (Fall 2016), 50 (Spring 2017), 26 (Fall 2017) and 33 (Spring 2018).

The pre and post tests are identical in content, both consisting of a small number of exercises in which students attempt to insert (delete) a node, given a starting tree. Problems in the insertion tutor require students to insert 9 numbers to an empty tree. Similarly, problems in the deletion tutor require students to delete all values from an initial tree with 9 nodes. The number of questions in each session is listed in Table 1.

| | **Pre-test** | **Tutor** | **Post-test** |
|---|---|---|---|
| Insertion | 4 | 20 | 4 |
| Deletion | 7 | 25 (F2017, S2018)<br>20 (others) | 7 |

**Table 1: Number of questions in each session. Each question has 9 parts, each of which requires multiple steps to solve.**

In the tutoring section, Each time the student submits an answer, the system attempts to predict whether that answer is correct, based on the student model's knowledge. Then the actual grading is performed to check whether this prediction is right. The accuracy of the student model is defined as the number of correct predictions divided by the total number of predictions. In all subsequent tables, unless otherwise specified, the data are averaged across all students in each semester.

## 6.1 Evaluating performance prediction accuracy

### *6.1.1 Baseline prediction*

When predicting with fixed probability, the resulting average accuracies approximate 50% in all semesters, with small standard deviations (5%).

### *6.1.2 Bayesian model with error contexts*

We evaluate the Bayesian student model on both the insertion tutor and deletion tutor (Table 2). The columns, from top to bottom, respectively refer to the followings: number of average and total correct predictions, mean accuracy, standard deviation of accuracy, lowest and highest accuracy across all students in the semester.

Note that because we decided to add five more exercises in Fall 2017 and Spring 2018, the number of answers submitted (and the number of predictions) in this semester is higher than in the others. We can see that data across the fall semesters are consistent. There is more variation in Spring 2017 due to the larger number of students enrolled, but only

| **Insertion** | **F2016** | **S2017** | **F2017** | **S2018** |
|---|---|---|---|---|
| Correct/Total | 268/372 | 259/399 | 267/371 | 272/375 |
| Accuracy | 72% | 66% | 72% | 73% |
| Stdev Acc | 4% | 8% | 5% | 5% |
| Min Acc | 63% | 50% | 62% | 65% |
| Max Acc | 81% | 86% | 83% | 84% |

| **Deletion** | **F2016** | **S2017** | **F2017** | **S2018** |
|---|---|---|---|---|
| Correct/Total | 270/383 | 268/383 | 351/461 | 360/472 |
| Accuracy | 70% | 70% | 76% | 74% |
| Stdev Acc | 5% | 4% | 4% | 4% |
| Min Acc | 64% | 61% | 68% | 65% |
| Max Acc | 82% | 80% | 83% | 81% |

**Table 2: System's accuracy on the insertion tutor and deletion tutor, using Bayesian modeling.**

in the insertion tutor. The system achieves the highest accuracy (76%) when predicting performance in the deletion tutor of Fall 2017 - this can be explained by the increased number of exercises, which allows the Bayesian network more opportunities to update itself and to yield better predictions in turn. Overall, using Bayesian modeling yields a 20% improvement in accuracy, compared to baseline prediction.

### *6.1.3 Bayesian model with extended error contexts*

Table 3 shows the model's accuracy when accounting for the previous transformations in the contexts. The average accuracy is around 80%, while the maximum accuracy can reach as high as 96% (in Fall 2017). Hence this approach has by far yielded the best accuracy, about 10% more than using Bayesian model with the standard error context, and 30% more than baseline prediction.

| **Insertion** | **F2016** | **S2017** | **F2017** | **S2018** |
|---|---|---|---|---|
| Correct/Total | 310/372 | 325/399 | 322/371 | 330/375 |
| Accuracy | 85% | 81% | 87% | 83% |
| Stdev Acc | 7% | 7% | 7% | 8% |
| Min Acc | 63% | 62% | 58% | 60% |
| Max Acc | 92% | 94% | 96% | 92% |

| **Deletion** | **F2016** | **S2017** | **F2017** | **S2018** |
|---|---|---|---|---|
| Correct/Total | 320/383 | 305/383 | 388/461 | 390/472 |
| Accuracy | 82% | 79% | 85% | 81% |
| Stdev Acc | 7% | 8% | 7% | 7% |
| Min Acc | 62% | 57% | 65% | 61% |
| Max Acc | 91% | 87% | 90% | 88% |

**Table 3: System's accuracy on the insertion tutor and deletion tutor, using Bayesian modeling with extended error context.**

We then performed additional analysis in this direction to see whether there is room for improvement and what problem-solving patterns students might have. Table 4 breaks down the accuracy in more detail; each prediction is categorized as either correct (C), false positive (FP) or false negative (FN). False positive occurs when the student answer is incorrect but predicted to be correct; false negative occurs when the student answer is correct but predicted to be incorrect. We see that in most cases, if the student is correct, the system can predict so. The majority of incorrect predictions occur in the false positive condition, where the system thinks that the student has mastered the transformation but in actuality the student still has an erroneous model. This suggests that we may be able to fine-tune the Bayesian network's behavior, in particular by decreasing the conditional probability that the student can submit a correct answer if the system thinks she understands the corresponding transformation.

| **Insertion** | **F2016** | **S2017** | **F2017** | **S2018** |
|---|---|---|---|---|
| C | 85% | 81% | 87% | 85% |
| FP | 11% | 14% | 10% | 13% |
| FN | 4% | 5% | 3% | 2% |

| **Deletion** | **F2016** | **S2017** | **F2017** | **S2018** |
|---|---|---|---|---|
| C | 82% | 79% | 85% | 80% |
| FP | 15% | 16% | 13% | 14% |
| FN | 3% | 5% | 2% | 6% |

**Table 4: System's prediction results on insertion tutor and deletion tutor, averaged by students.**

Next, we look at the cumulative statistics for each semester. Specifically, we would like to know the transformations involved in the answers that the system can predict accurately and in those that the system cannot. Table 5 breaks down this information from Fall 2017 based on the three categories C, FP and FN mentioned above. Here the tree insertion transformations of interest are Insert node (Insert), Color flip (Cflip), Single rotate (SingleR), Double rotate (DoubleR). Data from the other two semesters are also similar.

| | **Insert** | **Cflip** | **SingleR** | **DoubleR** |
|---|---|---|---|---|
| C | 3353 (90%) | 792 (73%) | 331 (79%) | 348 (80%) |
| FP | 327 (9%) | 229 (21%) | 59 (14%) | 66 (15%) |
| FN | 53 (1%) | 71 (6%) | 31 (7%) | 23 (5%) |
| Total | 3733 | 1092 | 421 | 437 |

**Table 5: System's prediction result count for insertion tutor, cumulative in Fall 2017.**

| | **Delete** | **Cflip** | **SingleR** |
|---|---|---|---|
| C | 3413 (89%) | 786 (71%) | 341 (74%) |
| FP | 385 (10%) | 243 (22%) | 79 (17%) |
| FN | 52 (1%) | 78 (8%) | 41 (9%) |
| Total | 3850 | 1107 | 461 |

| | **DoubleR** | **DropR** | **Switch** |
|---|---|---|---|
| C | 367 (80%) | 292 (68%) | 795 (95%) |
| FP | 54 (12%) | 101 (24%) | 27 (3%) |
| FN | 33 (7%) | 36 (8%) | 15 (2%) |
| Total | 454 | 429 | 837 |

**Table 6: System's prediction result count for deletion tutor, cumulative in Fall 2017.**

Table 6 presents the same kind of data for the deletion tutor in Fall 2017. Here the tree transformations of interest are Delete node (Delete), Color flip (Cflip), Single rotate (SingleR), Double rotate (DoubleR), Drop rotate (DropR) and Switch value (Switch).

We also look at, among all the error contexts identified, which pair of sequential transformations (i.e., the current transformation following a previous transformation) occurs the most, since our analysis includes the previous transformation in the error contexts. Table 7 shows that, in red-black tree insertion, students are most likely to make mistakes in rotation operations if they previously performed an insert node operation. This pattern can be explained by the fact that in most tree insertion problems, the final step is to insert a new node at a leaf node's child. However, in some cases, this leaf is already red; adding a red child to it would then yield two consecutive red nodes, violating the properties of red-black trees. Hence another rotation at the newly inserted node is required to remedy the situation, which students tend to forget. It should be noted that a color flip may also result in consecutive red nodes, thereby forcing a rotation to follow; the third and fourth row in Table 7 represent this case. In general, from our teaching experience, all four cases occur very often, but this is the first time we obtain a relative ranking of their frequencies.

| Transformation | Previous Trans | Count |
|---|---|---|
| SingleR | Insert | 90 |
| DoubleR | Insert | 72 |
| SingleR | Cflip | 65 |
| DoubleR | Cflip | 50 |

**Table 7: Most common pairs of insertion transformations in students' errors across three semesters.**

Table 8 shows that, in red-black tree deletion, students are most likely to make mistakes in *delete node* following *switch value*. Interestingly, as we previously analyzed, students usually perform *switch value* correctly. However, after this step, they tend to move straight to the leaf whose value was switched and delete it - this is correct in normal binary search trees, but in red-black trees, we still have to traverse down one node at a time until reaching the leaf, performing necessary transformations along the way before the actual deletion. Another noteworthy point is that students tend to forget to execute the *drop rotate* operation, but only when it is necessary to do so at the root (in this case, drop rotate is the first transformation in the solution sequence, so it has no previous transformation).

| Transformation | Previous Trans | Count |
|---|---|---|
| Delete | Switch | 98 |
| DoubleR | Cflip | 78 |
| SingleR | Cflip | 76 |
| Drop rotate | - | 27 |

**Table 8: Most common pairs of deletion transformations in students' errors across three semesters.**

### 6.2 Assessing students' test performances

While the previous study by Liew & Xhakaj [15] reported an improvement in individual student performance from pre-test to post-test, it was conducted on a small sample of 12 students. To measure this effect on a larger scale, we performed a paired samples t-test to compare the student's number of first errors in the pre-test $e_{pre}$ and in the post-test $e_{post}$. Results show that in tree insertion, there was a significant difference between $e_{pre}$ $(M = 2.81,\ SD = 1.35)$ and $e_{post}$ $(M = 1.72, SD = 1.35)$; $t(137) = -8.23$, $p = 2.95 \cdot 10^{-13}$. Similarly, in deletion, there was a significant difference between $e_{pre}$ and $e_{post}$ $(M = 3.63,\ SD = 1.08)$ and $e_{post}$ $(M = 2.68,\ SD = 1.48)$; $t(137) = -5.33$, $p = 3.12 \cdot 10^{-7}$. Hence the impact of the tutoring system on reducing student errors is statistically significant at the 1% level, which is consistent with [15].

Further analysis on the total number of errors overall and per each transformation rule reveals that the errors in node selection decrease across all semesters; in insertion exercises there is a steady 50% reduction from pre test to post test, whereas the differences vary more in deletion exercises. Interestingly, the number of errors in applications do not seem to decrease by much; in particular, errors in *single rotation* and *double rotation* do not decrease significantly, and even increase in some cases, between the pre and post test. The reason is that in the pre-test, because most students forget about *color flip*, they do not have many opportunities to apply *single rotation* or *double rotation*, resulting in few application errors reported. On the other hand. in the post-test, students have already mastered *color flip*, which then prompted them to apply rotations on more occasions, in which case more application errors were likely to occur. On further analysis, if we only consider students who did make rotation errors in the pre-test, then their number of rotation errors in the post-test also decreased significantly, by almost 75%. A more detailed breakdown of students' test performance is presented in [13].

## 7. CONCLUSION

This paper has described how we have mined logs of student actions on red-black tree operations to build a Bayesian model of their mastery of the skills involved. The analysis of the logs has helped us to determine (1) the most frequent errors that the students make, and (2) the contexts in which the errors are made. This knowledge can and will be used to improve both the tutoring system and the classroom instruction. The instructors can use the data to modify and customize their instruction to focus more attention on the problematic areas.

Results from this study also open up several future directions. First and foremost, the student model has demonstrated a reasonable performance and can now be used to build an adaptive learning system, which can potentially further reduce the number of errors. Second, gathering more student data would allow the implementation of more sophisticated techniques, such as hierarchical Bayesian learning or deep learning, in student model construction, which would in turn enhance the model's accuracy. Finally, balanced trees in general share many common properties and transformations; an adaptation of the current system to a related domain (e.g., AVL trees, AA trees, splay trees), could therefore provide insights on how general the underlying framework is.

## 8. REFERENCES

[1] Almond, R. G., Mislevy, R. J., Steinberg, L. S., Yan, D., and Williamson, D. M. *Bayesian networks in educational assessment.* Springer, 2015.

[2] Bauer, A., and Popović, Z. Collaborative problem solving in an open-ended scientific discovery game.

*Proc. ACM Hum.-Comput. Interact. 1*, CSCW (Dec. 2017), 22:1–22:21.

[3] Cheang, B., Kurnia, A., Lim, A., and Oon, W.-C. On automated grading of programming assignments in an academic institution. *Computers & Education 41*, 2 (2003), 121–131.

[4] Conati, C., Gertner, A., and Vanlehn, K. Using bayesian networks to manage uncertainty in student modeling. *User modeling and user-adapted interaction 12*, 4 (2002), 371–417.

[5] Cormen, T. H. *Introduction to algorithms*. MIT press, 2009.

[6] Forlizzi, J., McLaren, B. M., Ganoe, C., McLaren, P. B., Kihumba, G., and Lister, K. Decimal point: Designing and developing an educational game to teach decimals to middle school students. In *8th European Conference on Games-Based Learning: ECGBL2014* (2014), pp. 128–135.

[7] G H Al-Bastami, B., and Abu Naser, S. Design and development of an intelligent tutoring system for c# language. 8795–8809.

[8] Geigle, C., Zhai, C., and Ferguson, D. C. An exploration of automated grading of complex assignments. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale* (2016), ACM, pp. 351–360.

[9] Helmick, M. T. Interface-based programming assignments and automatic grading of java programs. In *ACM SIGCSE Bulletin* (2007), vol. 39, ACM, pp. 63–67.

[10] Kastner, M., and Stangla, B. Multiple choice and constructed response tests: Do test format and scoring matter? *Procedia-Social and Behavioral Sciences 12* (2011), 263–273.

[11] Kyrilov, A., and Noelle, D. C. Using case-based reasoning to improve the quality of feedback provided by automated grading systems. *International Association for Development of the Information Society* (2014).

[12] Lee, Y.-J. Analyzing log files to predict students' problem solving performance in a computer-based physics tutor. *Journal of Educational Technology & Society 18*, 2 (2015).

[13] Liew, C. W., and Nguyen, H. Determining what the student understands - assessment in an unscaffolded environment. In *Proceedings of the Fourteenth International Conference on Intelligent Tutoring Systems (ITS)* (2018).

[14] Liew, C. W., and Smith, D. E. Checking for dimensional correctness in physics equations. In *FLAIRS Conference* (2002), pp. 299–303.

[15] Liew, C. W., and Xhakaj, F. Teaching a complex process: Insertion in red black trees. In *International Conference on Artificial Intelligence in Education* (2015), Springer, pp. 698–701.

[16] Martin, J., and VanLehn, K. Student assessment using bayesian nets. *International Journal of Human-Computer Studies 42*, 6 (1995), 575–591.

[17] Matthews, K., Janicki, T., He, L., and Patterson, L. Implementation of an automated grading system with an adaptive learning component to affect student feedback and response time. *Journal of Information Systems Education 23*, 1 (2012), 71.

[18] Minaei-Bidgoli, B., Kashy, D. A., Kortemeyer, G., and Punch, W. F. Predicting student performance: an application of data mining methods with an educational web-based system. In *Frontiers in education, 2003. FIE 2003 33rd annual* (2003), vol. 1, IEEE, pp. T2A–13.

[19] Mitrovic, A., Ohlsson, S., and Barrow, D. K. The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education 60*, 1 (2013), 264–272.

[20] Piech, C., Huang, J., Nguyen, A., Phulsuksombati, M., Sahami, M., and Guibas, L. Learning program embeddings to propagate feedback on student code. *arXiv preprint arXiv:1505.05969* (2015).

[21] Singh, R., Gulwani, S., and Solar-Lezama, A. Automated feedback generation for introductory programming assignments. *ACM SIGPLAN Notices 48*, 6 (2013), 15–26.

[22] Yudelson, M. V., Koedinger, K. R., and Gordon, G. J. Individualized bayesian knowledge tracing models. In *International Conference on Artificial Intelligence in Education* (2013), Springer, pp. 171–180.