Student Modeling in a Non-Scaffolded Testing Environment

Huy Nguyen

Advisor: Professor Chun Wai Liew

Contents

List of Figures

List of Tables

Abstract

The student model is one of the main components of an Intelligent Tutoring System and plays a vital role in the system's decision making, adaptability and evaluation. In many tutoring systems, a common way of initializing the model is to analyze students' performance in a pre-test prior to the tutoring session. The test items are often designed in conventional formats such as multiple-choice for ease of use, but may impact the model's accuracy due to their scaffolded nature, especially in a graphical knowledge domain. This thesis describes an approach to building a student model from a non-scaffolded assessment environment in the domain of balanced binary search trees. The model is then used to predict student's performance throughout the tutoring session. Evaluation on four semesters of data from using a red-black tree tutor shows that having a consistent and bias-free format for the pre and post-test makes the overall assessment more accurate, and that an effective Bayesian student model can be constructed from unconstrained student inputs in this setting.

Acknowledgements

First and foremost, I would like to express my gratitude to Professor Chun Wai Liew for his continued guidance in the past three years. Professor Liew's insights and experience were fundamental in helping me overcome the challenges in this thesis work and improve my research skills. I am also very grateful to my best friend and colleague, Darren Norton, who did a phenomenal job in setting up the database and website application during the early development stage. In addition, my sincere thanks goes to Professor Linda Plotnick and Professor Qin Lu for their encouragement and feedback as committee members.

I also truly appreciate all CS 150 students who have taken part in this study. My motivation during the most difficult times comes from none other than the thank-you I received emails from you. Witnessing your growth from the tutoring system has been nothing short of an amazing experience for me.

Last but not least, I would like to thank ACM SIGCSE and Lafayette College's Excel program for providing me with the funding to pursue this project work.

Chapter 1

Introduction

Data structures are an important part of a computer science education and often listed as one of the programming fundamentals in ACM's recommended curriculum [29]. The structures typically covered in an introductory course include arrays & linked lists, stack & queues, heaps, trees and hash tables. While iterative structures like arrays are straightforward, other recursive structures such as trees often pose challenges to students. Nevertheless, trees are highly valuable thanks to their logarithmic complexity. At Lafayette College, a type of tree being taught is red-black tree, a state-of-the-art balanced binary search tree available through the Java API [37]. In this thesis I present my work on the development of an Intelligent Tutoring System that teaches red-black tree, using the system as a platform to model students' learning process and identify common error patterns that will in turn help improve the teaching and learning experience in this knowledge domain.

1.1 Intelligent Tutoring Systems

An Intelligent Tutoring System (ITS) is a computer system that provides immediate and customized feedback to learners without human intervention. The incentive to develop ITSs comes from educational studies showing that individual tutoring is much more effective than group teaching [8, 9]. The need for a tool to promote effective learning on a large scale, in combination with the rising field of Artificial Intelligence, brought together researchers from various disciplines to collaborate on the development of ITS in the late 1970s.

An ITS has four major components: domain modeling, student modeling, tutor modeling, and user interface [20]. Domain modeling refers to encoding the domain knowledge in the system. Student modeling means quantifying and capturing all relevant parameters about a student, as well as updating them over time. Tutor modeling is a mechanism that combines knowledge from the domain and student model to determine the system's intervention, such as whether it should offer any hint or feedback. Finally, the front-end user interface presents relevant information to students and allows them to interact with the system. Among these, student modeling plays the most important role in providing immediate and individualized feedback. While the other three components are often developed beforehand by researchers and domain experts, the construction of a student model only takes place once the student begins using the system. Typical parameters of interest within the model include student performance, knowledge, emotion and motivation, which are used to (1) predict a student's behaviors or answers during and after the tutoring session, and (2) assess the student's mastery of the domain knowledge. A good student model allows the system to become *adaptive* - to vary its content, representation and responses according to the state of the learner [17]. Currently, Adaptive Learning Systems are at the forefront of ITS thanks to their ability to completely replace a human instructor in offering one-on-one tutor [5].

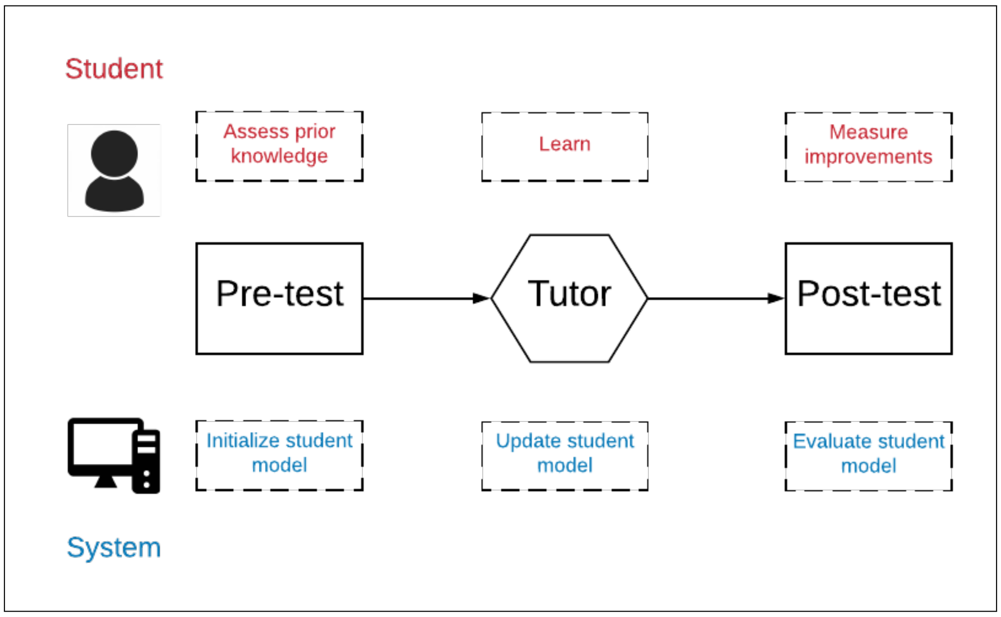


Figure 1.1: A workflow diagram of a tutoring system's components.

While there have been some initiatives to make ITSs openly available online [3], the systems are usually deployed in a classroom setting. Researchers often cooperate with local school teachers to integrate ITS materials into the curriculum and set up study conditions, as this is the best way to minimize external influences on the study's results. In this scenario, a typical ITS study consists of three stages: the pre-test, tutor and post-test [14]. Figure 1.1 describes the student and the system's respective role at each stage. From the students' view, they first take a pre-test to evaluate prior knowledge, then learn from the tutor, and finally take a post-test to measure improvements. From the system's view, the student model is initialized from pre-test data, then updated as the student interacts with the tutor, and finally evaluated based on post-test performance.

Across different ITSs, the design and implementation of each stage can vary greatly, depending on the tutoring domain. For example, the tests may be conducted online or on paper, using multiple-choice, fill-in-the-blank or free response questions. The tutor can also include different types of activities; typically students work on some exercises (with feedback & hints provided by the system), but it may also be helpful to show them a complete worked out example [28] or have them identify errors in a given flawed solution [31]. Therefore, to discuss the Red-black Tree Tutor's design, we first introduce the domain knowledge.

1.2 Red-black Tree

A red-black tree (Figure 1.2) 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 [15]:

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.

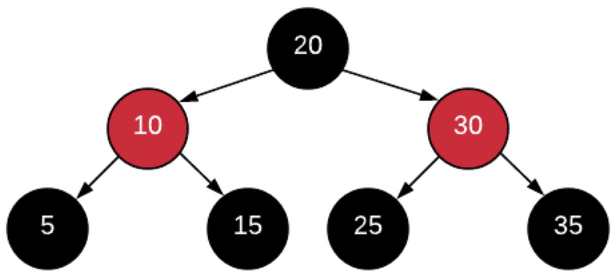


Figure 1.2: An example red-black tree with 7 nodes.

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. To insert or delete a value, we first make a top-down[1] traversal from the root node to the appropriate leaf. At each node along the way,

[1]There are also bottom-up algorithms for insertion and deletion, but the current tutor only teaches the top-down approach.

depending on its color and the surrounding nodes, a tree transformation is performed. There are six transformations used for insertion and eight for deletion (Table 1.1). 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, there will be no need for any subsequent modifications to the tree to preserve its properties. Thus the solution for a single node insertion (or deletion) consists of a sequence of transformations ending with the actual insertion (deletion). Another perspective is that the solution consists of a sequence of red-black trees starting with the initial tree and ending with the final tree having one more (or less) node, which is the node just inserted (deleted).

Insertion	**Deletion**
Color flip	Color flip
Single rotate	Single rotate
Double rotate	Double rotate
Insert node	Delete node
Forward	Drop
Color root black	Color root black
	Switch value
	Drop rotate

Table 1.1: Transformations in red-black tree insertion and deletion.

In insertion, *color flip* helps minimize the number of rotations that may arise from inserting a new leaf node that is colored red. *Single rotation* and *double rotation* are used to correct any violation of two consecutive red nodes resulting from either *color flip* or *insert node*. An illustration of these operations is shown in Figure 1.3. On the other hand, in deletion, both *color flip* and *single/double rotation* are used to make the current node red; their preconditions depend on the colors of the sibling node and its children. In cases where they do not apply, *drop rotate* is used to move down to the next level. By convention, the root node is black so at the end of each insertion/deletion, a *color root black* might be applied if necessary.

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 tree with more than 7 nodes, or three levels of depth. Furthermore, some transformations such as *color flip* operate differently for insertion and deletion, causing another source of confusion. A previous study on this domain by Liew & Xhakaj [27] 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.

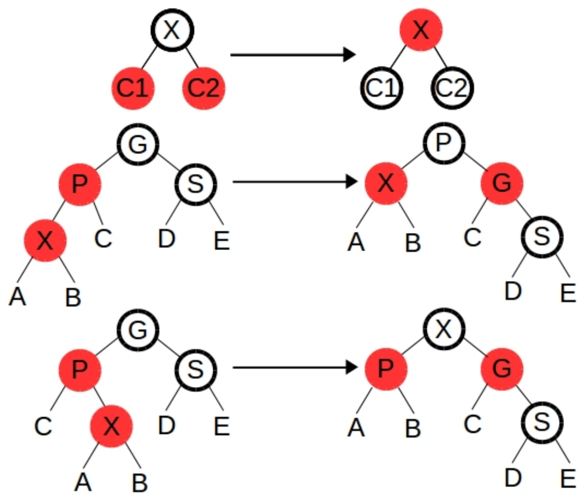


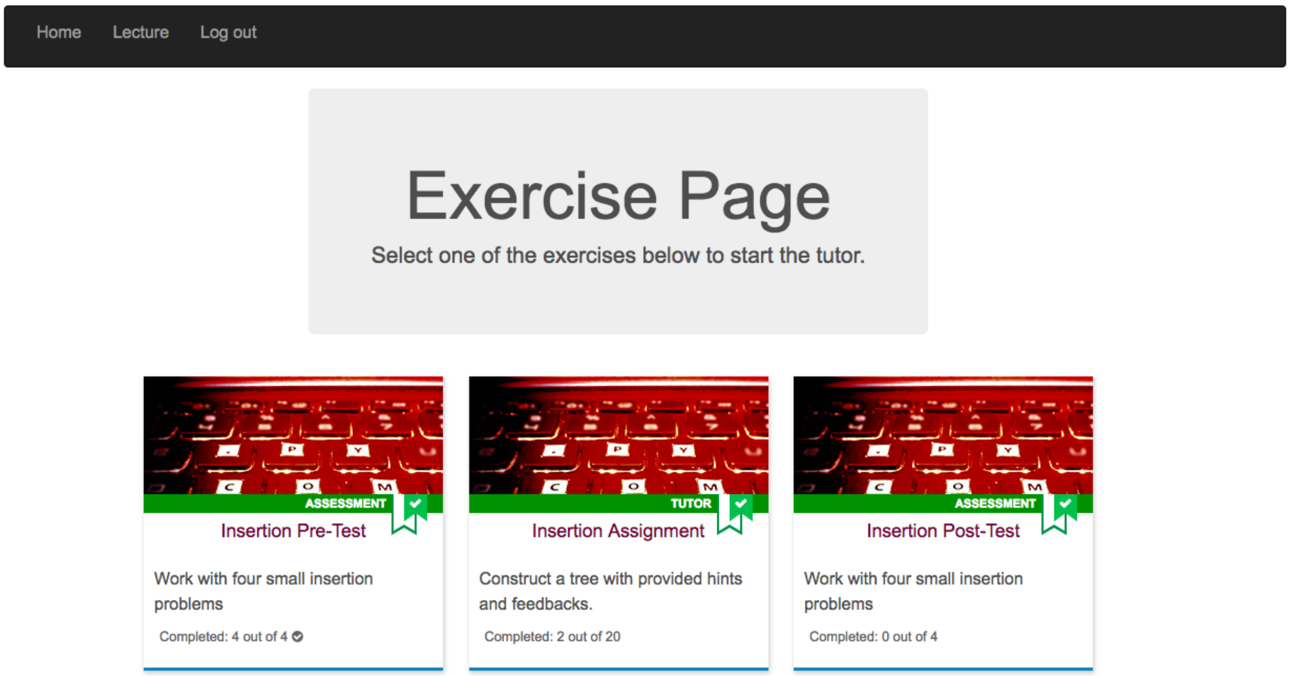
Figure 1.3: Top-Down insertion rules (top to bottom): color flip, single rotation, double rotation. X is the current node and relative to X, C1, C2, P, S, and G are the left child, right child, parent, sibling and grandparent. A, B, C, D and E are other nodes which may or may not be present.

1.3 The Red-black Tree Tutor

The tutoring system is a web-based tutor that allows students to log in and complete assignments. It has been developed since Summer 2016 in Scala & Play Framework [1], following the Model-View-Controller mechanism. The front-end interface contains HTML, CSS and JavaScript code, while the back-end is managed by MySQL and Play. Currently it is deployed on one of Lafayette's computing servers at `139.147.9.189` for the studies conducted here.

The tutor environment was adopted from [27], while the test environment was built to replace the paper version of the pre and post test. Figure 1.4 depicts a student's dashboard which shows the current tasks and progress. Students have to work through three tasks: pre-test, assignment (tutor) and post-test, in that order. Each task only unlocks when all preceding tasks have been finished. There is also a link to the lecture page, which is a summary of relevant red-black tree rules and algorithms; students are free to consult this link, but only during the assignment. There are also three corresponding tasks for tree deletion are not shown here. Instructors have a separate dashboard to manage the classroom, change task settings, view student answers and track their progress.

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



Home Lecture Log out

Exercise Page

Select one of the exercises below to start the tutor.

ASSESSMENT
Insertion Pre-Test
Work with four small insertion problems
Completed: 4 out of 4 ✓

TUTOR
Insertion Assignment
Construct a tree with provided hints and feedbacks.
Completed: 2 out of 20

ASSESSMENT
Insertion Post-Test
Work with four small insertion problems
Completed: 0 out of 4

Figure 1.4: The tutoring system's web interface from the student's view.

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 (Figure 1.5). 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 into 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 (Section 3.1) 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 interface; otherwise, a message is displayed to the student indicating that the current selection is incorrect. This approach is based on the finding that learners often have difficulty identifying the transformations rather than applying them [27]; students also find the task of repeated application of the transformations tedious and time consuming.

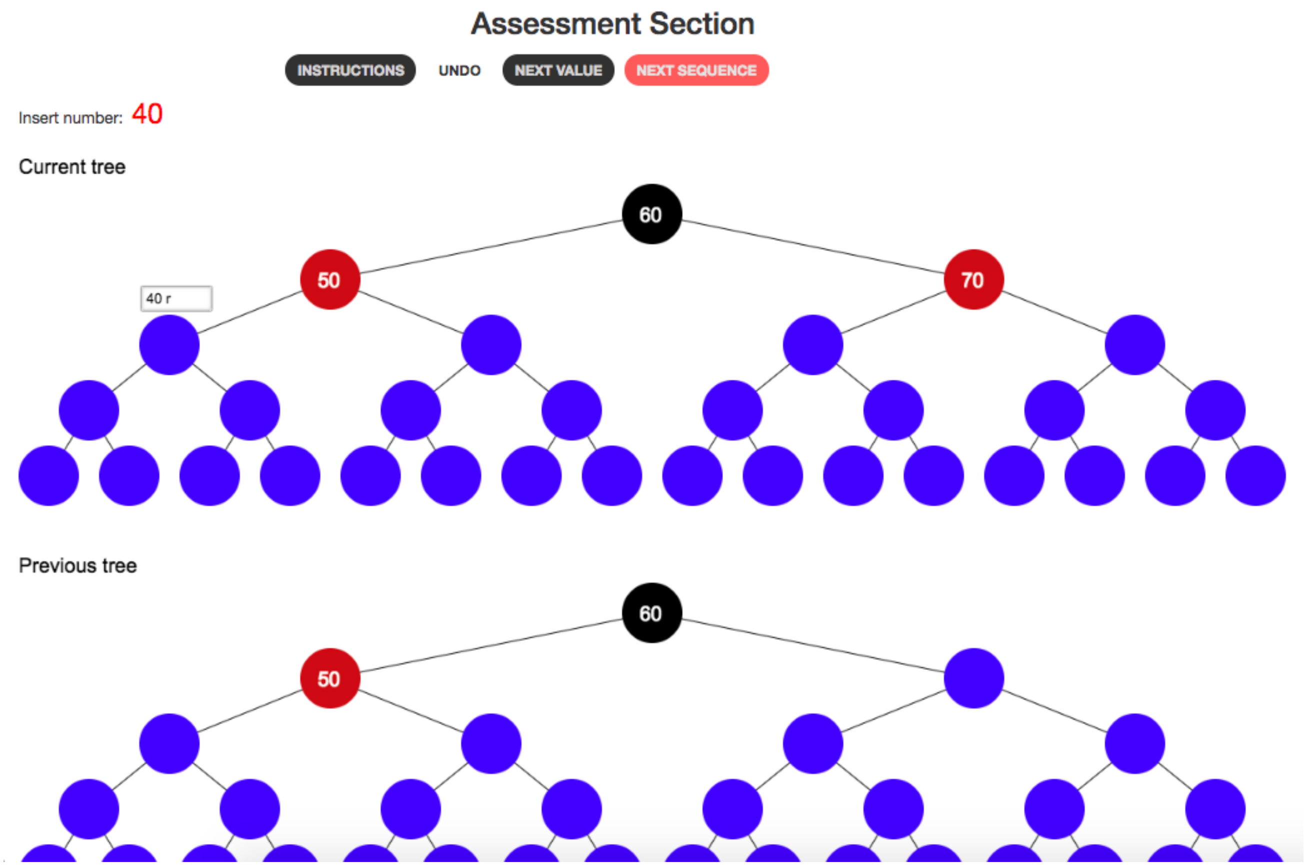


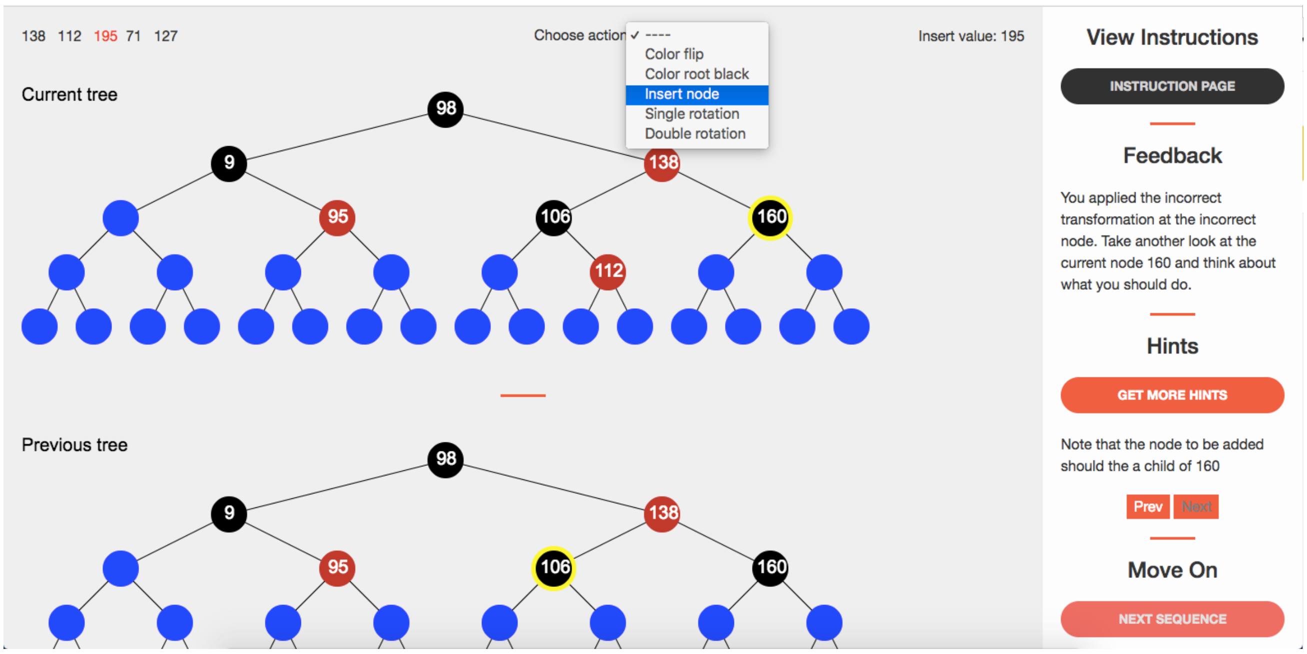
Figure 1.5: The assessment interface. The current tree is shown on top and the previous tree bottom. In this problem, students have to insert 40 to the tree. Clicking on **NEXT VALUE** will submit the student's answer for inserting 40 and present the next number to insert. Clicking on **NEXT SEQUENCE** will present the next problem with a new starting tree.

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 [18, 41]. For example, consider the following probability problem:

Question. If you roll a fair dice twice, what is the probability of getting a 6 both times?	
(a) 1/6	(b) 1/36
(c) 1/3	(d) 1/12

Table 1.2: A sample multiple-choice question.

This question tests knowledge of the probability multiplication rule, i.e., if A and B are independent events then $P(A, B) = P(A) \cdot P(B)$. In this case, the correct choice is 1/36, but other choices are also designed to reflect common errors that students have typically made. The instructor could then construct a direct mapping from the choices picked to the associated misconceptions, based on teaching experience (Table 1.3). In other words, knowing which options



You applied the incorrect transformation at the incorrect node. Take another look at the current node 160 and think about what you should do.

Note that the node to be added should the a child of 160

Figure 1.6: The tutor interface. The student has picked 160 as the current node and is now selecting the transformation to apply. If the student is correct, the chosen transformation will be applied by the system, resulting in a new current tree, while the existing current tree becomes the previous tree. If the student is incorrect, she will be informed in the Feedback section.

the student picked is sufficient to infer why she made that decision. However, there are difficulties in adapting this approach in the domain of red-black trees:

Choice	Misconception
1/6	Student does not understand joint distribution of indepedent events.
1/36	No misconception. Student understands multiplication rule.
1/3	Student mistakes addition rule for multiplication rule.
1/12	Student understands the sample space incorrectly.

Table 1.3: Mapping between choices and misconceptions in the sample probability question.

- Multiple-choice questions can be answered by pure guessing. In our domain, the correct tree can be identified quite easily if presented in multiple-choice format. Furthermore, the options presented might not capture the full space of misconceptions that students have, especially if each decision is not a simple primitive choice.

- Tree data structures are an example of problems where the steps are best input graphically to show how the data structure changes internally. As we will discuss in Section 2.2, visualization is an essential feature in data structures tutoring system. Conventional question formats such as multiple-choice would therefore greatly constrain the student's answer.

- Although not part of the study, the midterm and final exam require students to construct a red-black tree on paper from scratch. The tutor's testing environments should therefore be consistent and as close to this realistic format as possible.

1.4 Bayesian student modeling

Consider a hypothetical scenario: a student is given a set of exercises related to some knowledge P to test her understanding of the subject - a procedure that human teachers often follow [10]. Assume there are ten questions; if the student answers all ten correctly, it's reasonable to assume that she knows P well[2]. However, what can be inferred about her knowledge if she answers four correctly and six incorrectly? Would there be a difference in her answering the first four correctly and her answering the last four correctly? One could argue that, in the former case, the student only understands P in some but not all contexts, while in the latter case, it's also possible that the student realizes her mistake in the first four answers and successfully fixes them in the remaining six. Furthermore, a correct answer may be due to lucky guessing, while an incorrect answer could be a careless slip, which should also be taken into account.

Bayesian belief network, or Bayesian network in short, is a method that can capture the above pedagogical nuances and represent them in a sound mathematical system. Specifically, Bayesian networks have been used in intelligent tutors to support classification and prediction, model student knowledge and predict student behavior [33]. They operate on the assumptions that:

(i) Real-world data, such as those collected from ITS, should be treated with some degree of uncertainty and are therefore not always suited for clean statistics tests.

(ii) Student knowledge is abstract and not directly observable, but has a direct influence on student actions (problems solved, hints requested, answers submitted), which can be explicitly observed.

(iii) In education, prediction of students' current and future performances should be based on past performances. The belief of a teacher or system about student knowledge should depend on both new observations and past evidence.

[2]to the extent covered in the given exercises.

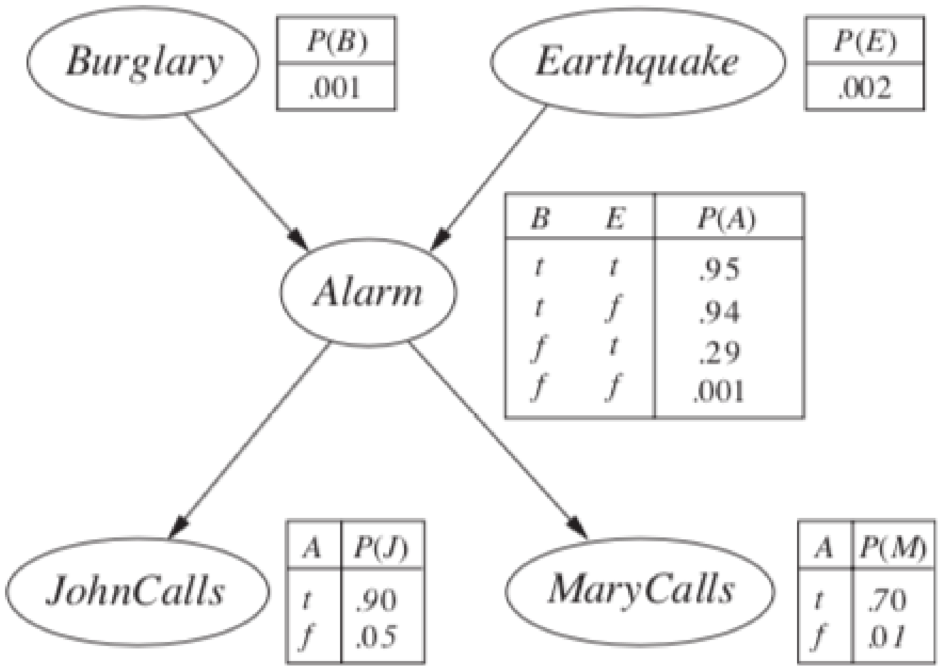


Figure 1.7: An example Bayesian network. Source: [39].

1.4.1 Bayesian network

A Bayesian network [39] is a directed graph that represents the causal relationship between random variables. Each node captures one variable, while a directed edge from node A to B depicts the fact that A causes B; A is then called the parent and B the child. Each edge has an associated weight, which is a conditional probability table depicting the probabilities of the child node, given each possible configuration of its parents (Figure 1.7). The nodes without parents have a prior probability. Once the topology and weights of a network have been specified, it can be used to calculate the probability of any given joint distribution of its variables. For instance,

$$\begin{aligned} P(j, m, a, \neg b, \neg e) &= P(j \mid a)P(m \mid a)P(a \mid \neg b \wedge \neg e)P(\neg b)P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 \\ &= 0.000628. \end{aligned}$$

Note that Figure 1.7 is an example of a static Bayesian network, which is suitable for probabilistic inference over variables that don't change over time. In contrast, dynamic Bayesian network [16] is often used to model variables that can change, such as student knowledge and mastery, since we expect the student to improve over the course of learning. In particular, a dynamic network tracks the posterior probability of variables whose values change over time, given a sequence of observations. It consists of time slices representing relevant temporal states in the process to be modelled [11]. For instance, let us now denote X_i as the state of variable X at time t_i. The network in Figure 1.8 represents the fact that C_{i+1} depends probabilistically on A and B (just like in a static network), but also on its immediate previous state C_i. This additional dependency allows us to model a variety of pedagogical nuances; setting $P(C_{i+1} = \text{True} \mid C_i = \text{False}) = 0.1$, for example, would model forgetting by expressing that there is a 10% chance the student doesn't know about Concept C at time t_{i+1} despite knowing it at time C_i. While the student model in this thesis work does not take into account forgetting, the use of a dynamic network can still help satisfy assumption (iii) as mentioned earlier, which is most relevant in education.

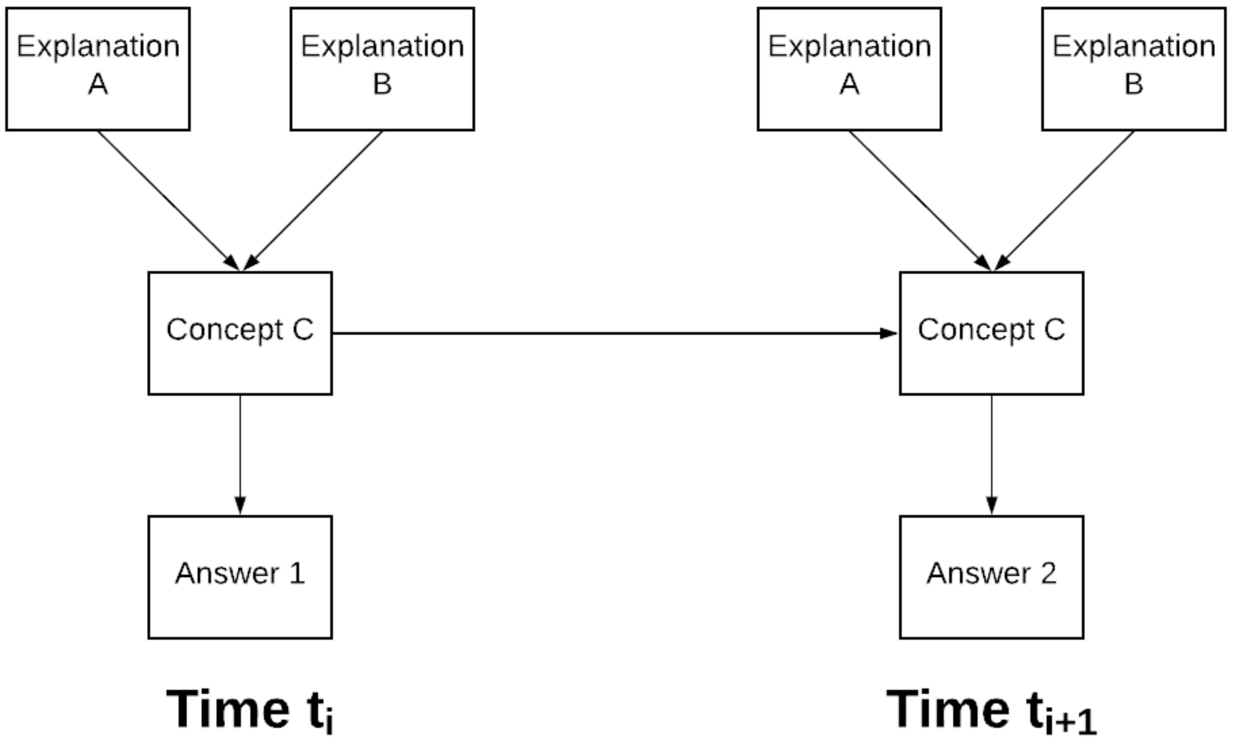


Figure 1.8: An example dynamic Bayesian network. Source: [11].

1.4.2 Bayesian Knowledge Tracing

In the context of ITS, a popular method of applying dynamic Bayesian network is called Bayesian Knowledge Tracing (BKT) [13, 48], which models student knowledge as a latent variable, updated by observing the correctness of a student's answer when she applies the corresponding skill. All variables in the network are binary; the skills are measured as mastered or not, and the correctness observations are either right or wrong. BKT associates each skill with four parameters:

- $P(L_0)$ - prior: the probability that the student has known this skill beforehand.

- $P(S)$ - slip: the probability that the student makes a mistake when applying a known skill.
- $P(G)$ - guess: the probability that the student answers correctly despite not knowing a skill.
- $P(T)$ - transition: the probability that the student's skill mastery changes from *False* to *True* after a time step.

Let $P(L_t)$ be the probability that the student knows a skill, or the mastery probablity in short, at time step t (each time the student submits an answer is labeled with time step, following the convention of a dynamic network). The variable values are then specified as follows:

1. The initial mastery probability is taken from the prior: $P(L_1) = P(L_0)$.
2. At time step t, to predict the correctness of a successive future answer $P(C_{t+1})$:
$$P(C_{t+1}) = P(L_t) \cdot (1 - P(S)) + (1 - P(L_t)) \cdot P(G).$$
3. Each time the student submits an answer, which is immediately graded for correctness, the posterior mastery probability $P(L_{t+1})$ is calculated based on the current mastery probability $P(L_t)$ serving as the prior:
   - If answer is correct:
$$\begin{aligned} P(L_t \mid \text{Correct}) &= \frac{P(L_t \;\&\; \text{Correct})}{P(\text{Correct})} \\ &= \frac{P(L_t) \cdot (1 - P(S))}{P(L_t) \cdot (1 - P(S)) + (1 - P(L_t)) \cdot P(G)}. \end{aligned}$$
   - If answer is incorrect:
$$\begin{aligned} P(L_t \mid \text{Incorrect}) &= \frac{P(L_t \;\&\; \text{Incorrect})}{P(\text{Incorrect})} \\ &= \frac{P(L_t) \cdot P(S)}{P(L_t) \cdot P(S) + (1 - P(L_t)) \cdot (1 - P(G))}. \end{aligned}$$
   - Update:
$$P(L_{t+1}) = p(L_t) + (1 - p(L_t)) \cdot P(T).$$

An advantage of Bayesian network is that it is intuitive and can operate with or without training data. In particular, the network weights can either be refined from training data or specified in advance by domain experts. ITS studies usually follow the latter option, as it is generally difficult to gather sufficient data to perform meaningful training; at Lafayette College, for instance, on average there are only about 30 students in CS 150 each semester. Hence the network's flexibility makes it an appropriate tool for my study. Another important property is its conservativenes [45]; the network is not influenced by minor changes in information. If a small amount of data contradicts the prior knowledge, which the network has high confidence in, its belief only changes slightly. Similarly, human instructors and tutoring systems should not update their belief about student knowledge based on one single action, as it might be the result of the student making a lucky guess or careless mistake, especially in multiple-choice questions.

1.5 Contributions

As mentioned earlier, Xhakaj's previous papers and thesis work [27, 46, 47] have demonstrated the effectiveness of the *granularity approach* in teaching red-black trees. My thesis work builds upon the tutoring system developed in these studies, with the goal of constructing an effective student model. I have made improvements in the following aspects:

- **Automated assessment**. 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. Thanks to ITS, 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. I have developed a grading algorithm that can evaluate students' test answers and identify the first error in the majority of cases [25]. Beyond automating the test procedures, the algorithm also provides input for subsequent student modeling and data mining tasks.

- **Bayesian student model**. In many tutoring systems, a common way of initializing a student model is to analyze student's performance in a pre-test prior to the tutoring session. The test items are often designed in conventional formats such as multiple-choice for ease of use, but may impact the model's accuracy due to their scaffolded nature. I have proposed an approach to build a student model from a non-scaffolded test environment in the domain of red-black trees [34]. The model can predict students' performance throughout the tutoring session with good accuracy.

- **Data mining**. While Xhakaj's work focused on individual students' improvement through the use of the tutoring system, I extended this analysis to class-wide statistics that reveal both the common difficulties that students face in general while learning and the aspects of the tutoring system that can further improved [35].

Chapter 2

Related Work

This chapter provides a brief overview on two areas closely related to the Red-Black Tree Tutor: student modeling and Computer Science ITS. In student modeling, I discuss the two general research directions and present a case study of the ANDES physics tutor [12], which shares many similarities with the Red-black Tree Tutor. In Computer Science ITS, I introduce the breadth of the field and then focus on ITSs in data structure domains.

2.1 Student modeling research

According to [20], student modeling, despite being only one of the four main components of an ITS, has received the most attention from researchers. One reason is that student modeling can be applied to a wider range of research questions beyond ITS [40]. Another lies in the interdisciplinary challenges inherent in student model construction, which bring together combined efforts from several areas. A student model should ideally capture as much information about the student's cognitive and affective state as possible while being dynamic enough to accurately reflect the student's improvement over time [36]. To this end, several approaches, both from a cognitive science view and from a machine learning view, have been proposed.

Traditionally, student models are constructed based on cognitive science, with the assumption that how humans learn can be modeled as a computational process [36]. Two common techniques in this area are model tracing and constraint-based modeling. Model tracing [23] believes in the computer's ability to model the learning process step by step and rule by rule; in this approach the tutor attempts to infer the steps that the student could have taken to arrive at the solution and performs necessary interventions. In particular, Xhakaj's thesis work involved building an example-tracing tutor [2], which is an extension of this direction. In contrast, constraint-based modeling [33] proposes that only errors can be recognized and captured by the computer; the tutor's intervention is based only on the student's solution state, not the steps taken to get there. A study comparing the two techniques [22] shows that model tracing can provide more effective interventions and is less dependent on the domain knowledge, but takes more efforts to develop than constraint-based modeling.

Recently machine learning techniques have also been developed for the task of student modeling. These rely on the abundance of log data from students' interactions with the tutoring system and offer two advantages over cognitive-based techniques [20]. First, machine learning does not rely on any assumption about computers being able to model human learning. Second, they can be used to model a variety of properties, such as performance, emotion and robust learning. It should be noted that machine learning and cognitive-science based techniques are not mutually exclusive. The Red-Black Tree Tutor, for instance, employs approaches from both model tracing and Bayesian learning.

A crucial requirement for machine learning techniques is that the input data must be accurate. If the data collection method (i.e., test questions) contain instructional scaffoldings, they could potentially overestimate the students' performance; in some cases, students might be able to answer a question correctly without fully understanding the underlying knowledge, thanks to the provided scaffold. In this sense, multiple choice questions are not a preferred option; while frequently used thanks to their convenience, they pose several challenges in assessing a student's knowledge or skills. Specifically, the formulation of questions and answers could provide implicit hints to the correct answers. Furthermore, this format is more suitable for testing isolated knowledge rather than high-level thinking [7], and in many cases students' partial knowledge is ignored [6]. Finally, questions can be answered by lucky guessing, which undermines the reliability of the test [49]. Students could potentially make a different set of mistakes if they had to synthesize the answer from scratch. Free response questions do not face any of these issues, but are difficult to administer on a large scale 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 student model. For example, physics tutors such as ANDES [12] and OLAE [30] teach college-level Newtonian mechanics by having students identify the forces acting on a physical object and express them in a system of equations. ANDES, in particular, builds a static Bayesian network for each exercise (task-specific network); the network represents how each step can be derived from combining the previous steps and relevant physics rule [11]. When a problem solving step is entered by the student, the corresponding node in the network is set to True and the posterior probabilities of all other nodes are updated, given this evidence. The task-specific network is used to guess the implicit reasoning behind the students' action; its accuracy is influenced by how many steps the student did not show (i.e., the student performed them on scratch paper or in her mind) and how many alternative solutions there are. Another network being used is the domain-general network, which tracks long-term evolution of student knowledge. The Red-black Tree Tutor also employs a similar two-network architecture; its structure and interaction will be elaborated later on in Chapter 3. In the case of ANDES, this design choice was demonstrated to be capable of both knowledge tracing and plan recognition. However, the prior probabilities of rule nodes in the network were determined from a multiple-choice pre-test, which has pedagogical limitations as mentioned earlier. In addition, because each question can only involve a few rules, only a small portion of rule nodes had their priors determined, while the others were set to 0.5 by default. To address this issue in my tutor, I introduced non-scaffolded test questions that involve comprehensive domain knowledge and more closely match real world situations.

2.2 ITS in Computer Science & Data Structures

Several ITSs have been developed for various topics in computer science education, such as SQL [32], Python [38], Java [21], PHP [44], and API usage [24]. However, most of these systems belong to a programming-related domain. To my knowledge, there are only a few which focus on data structures. Learning a data structure requires not only basic coding knowledge but also the ability to visualize how the associated operations change the internal state of the structure. Therefore, the tutors in this domain usually have a graphical interface which students can interact with. Here I present an overview on a number of them.

ADIS [42] is a prototype animated data structure ITS used for teaching linked-lists, stacks, queues, trees and graphs. It was one of the first interactive web-based tutoring systems developed. In the examples presented, the system displays a visualization of a linked list; students can pick any node and select the atomic operation (add pointer, remove pointer, add temp node, ...) to perform at that node. The student model follows the constraint-based modeling approach, and the system only provides error feedback when some constraints have been violated, which guarantee a wrong solution. While ADIS was only a proof of concept, a similar ITS on linked list developed later [19] was evaluated in a real classroom setting; its results showed that students who used the tutor demonstrated good learning gain, and there was a positive correlation between their in-tutor performance and final grade in the class.

A study conducted by Estela, Carlos & Juan [4] compared the teaching of binary search trees in three conditions: (i) traditional instruction, (ii) web instruction, and (iii) multimedia-interactive. The learning media was a PDF file in condition (i), a static webpage in condition (ii), and a multimedia system with sound and images in condition (iii). Students in condition (iii) also did exercises on an interactive interface with a canvas tree and a list of numbers to work with, very similar to that of the Red-Black Tree Tutor; however, the only atomic operation supported was drag & drop, as binary tree does not involve any color change or rotation. The results showed that students in the multimedia condition performed significantly better than those in the other two.

In the specific domain of red-black trees, Liew & Xhakaj's work [27] was the first ITS developed. Its results show 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. [26] introduces an improved ITS that provides an online assessment environment and a grading algorithm, thereby automating the test procedures. My work builds upon this foundation by proposing a more accurate grading algorithm [25], building a student model [34], and performing cross-semester data mining [35].

Chapter 3

Student Modeling and Performance Prediction

The domain of red-black trees is very similar to Newtonian physics in that there are rules (tree transformations) which students learn to apply in different contexts[1]. The context in this case is the subtree surrounding the current node that is considered in each insertion (deletion) transformation's precondition. Student modeling requires keeping track of the student's mastery of each transformation in all relevant contexts, as their performance might differ based on the context and not just the rule. To achieve this goal, I followed the approach used in the ANDES system of having two Bayesian networks [11], one for domain-general knowledge and one for task-specific knowledge.

As mentioned earlier, the design of a Bayesian network requires establishing the connection between the rules and student actions. In multiple-choice format, we could construct a mapping from each option to its corresponding knowledge/misconception (Table 1.3). While this approach still works in the tutoring session, where students select a node and then the rule to apply, it cannot be applied to the non-scaffolded test environments, where the only inputs are raw binary trees. As information from the pre-test is required to initialize the student model, I devised a grading algorithm to extract relevant information from test answers as follows.

3.1 Grading Algorithm

To be fully automated, the system has to grade answers both in the tutor and in the tests. I will first introduce the tutor grading module as the easier case, then show how it can be utilized in the test grading module. Note that each module works on one subproblem (i.e.,

[1]From now on we will refer to tree transformations as rules in the student model.

the insertion/deletion of one single number) at a time. As each exercise typically involves inserting/deleting several numbers, a new module will be invoked for each number.

3.1.1 Tutor grading module

Grading in the tutoring section takes place in real-time, so that immediate feedback and hints can be provided to students. The inputs/outputs of this module is described in Figure 3.1. At each time step, we are given the state of the current tree as well as the number to insert/delete, and have to check whether the next student answer is correct, while accounting for possible alternative solutions. In the previous thesis work [46], Xhakaj built an example-tracing tutor to perform this task, but had to manually input the solution sequence, and only one default solution is accepted. To address this shortcoming, I implemented the following algorithm:

1. **Input**:
    - The question prompt, which includes the starting tree T_0 and the number to insert/delete n.
    - The state of the current tree T. [a]
    - The next student answer s^*, which includes the position of a node in T and the transformation to apply at that node.
2. **Output**:
    - Is the student correct? If not, how do we determine the type of error (incorrect node or transformation) and provide specific feedback?

[a]Note that the tutor only allows the student to progress on submitting correct answers. Hence we can assume the student has been correct on all previous trees leading to T.

Figure 3.1: A summary of the tutor grading problem.

1. From the starting tree T_0 and the input number n, following the textbook algorithm [43], generate a canonical sequence of steps (s_i) used to insert/delete n.
   For example, if T_0 is an empty tree and $n = 5$, (s_i) includes two steps: s_1 = insert node 5 (at root) and s_2 = color root black.

2. Based on (s_i), generate the sequence of resulting trees (T_i), where the first tree is T_0 and

any subsequent T_i results from applying s_i to T_{i-1}:

$$T_0 \overset{s_1}{\Longrightarrow} T_1 \overset{s_2}{\Longrightarrow} T_2 \overset{s_3}{\Longrightarrow} \ldots \overset{s_k}{\Longrightarrow} T_k.$$

3. Let j denote the position of the current tree T within (T_i), so $T = T_j$. In other words, the student has completed the steps $s_1, s_2, \ldots, s_j$ so far. We compare the submitted answer s^* to the expected next step s_{j+1}:

   - If the two match, the student is correct. Set $T = T_{j+1}$ as the current tree and display it on the tutor interface.
   - If the two don't match, try applying s^* to T, resulting in T^* and generate the solution for the remaining steps required to insert n to T^*.
     - If the generation works, s^* is correct; replace the canonical solution with this new solution.
     - Otherwise, if at any point an exception is raised, s^* is incorrect. Further check to see if s^* differ s_{j+1} with respect to the node position or selected transformation, and display the appropriate output message (e.g., Figure 1.6).

4. If the student has finished inserting n, invoke a new module for inserting/deleting the next number. The new initial tree is the final tree in this subproblem, T_k.

The above algorithm has been tested and successfully deployed since the first iteration of the Red-Black Tree Tutor. It can generate solutions and grade answers to any given exercises. This versatility means that the instructor only needs to specify the initial tree and number sequence, while the rest is handled by the tutoring system. In fact, the system currently allows the instructor to randomly generate any number of exercises as additional practice for students outside of the study.

3.1.2 Test grading module

Grading in the test sections only takes place after students have finished the study. Unlike the tutoring environment, the test environments' non-scaffolded design (Figure 1.5) means that we do not have any information about which node the student is at or which transformation she is trying to perform at any time, which makes grading a harder problem. The module's inputs and outputs are described in Figure 3.2.

1. **Input**:
   - A starting tree T_0 and the number to insert/delete.
   - A sequence of trees that the student submitted $T_1, T_2, \ldots, T_k$.
2. **Output**:
   - Is the student correct? If not, where did the first error occur? What is its type and context?

Figure 3.2: A summary of the grading module.

It is important to note that there are three different types of errors: (i) incorrect node selection, (ii) incorrect transformation selection, and (iii) incorrect transformation application [2]. Furthermore, each tree operation can be applied in more than one context; for example, color flip at the root node is performed differently from color flip at a non-root node (Figure 3.3). The student might be able to apply a certain transformation in one context but not the other; for each error, we would therefore also like to know the context in which it occurred. To this end, I developed a grading algorithm as follows:

1. From the starting tree and the number to insert/delete, generate a sequence of transformations $s_1, s_2, \ldots, s_n$ as the default canonical solution.

2. Look at every pair of input trees (T_i, T_j) in the student's tree sequence. Perform a set of searches to see whether there exists a transformation (or combination of transformations) that turns T_i into T_j. Brute-force is used to detect single transformations:

```
detect_trans(tree1, tree2):
    for N in tree1's nodes:
        for T in all possible trans:
            treeResult = apply T at node N
            if treeResult matches tree2 completely:
                return (N, T, Correct App)
            if treeResult matches tree2 structurally, but not color-wise:
                return (N, T, Incorrect App)
    return Unrecognized
```

[2]type (iii) does not occur during the tutor because the system performs the transformations in place of the students.

while heuristics are to detect combinations of transformations. In particular, we check if any combination reported in Table 3.1 is responsible for turning T_i into T_j.

Insertion combinations	**Deletion combinations**
Insert to empty tree and color root black	Switch value and delete node
Color flip at root and color root black	Color flip at root and color root black
Insert node and single rotate	Single rotate and delete node
Insert node and double rotate	Double rotate and delete node

Table 3.1: Common combinations of transformations in test answers.

If any transformation (or combination) is detected, keep a record of it; otherwise move on to the next pair.

3. Based on the results in Step 2, infer the sequence of transformations $s_1, s_2, \ldots, s_m$ that the student was following, for example:

$$\overbrace{T_0 \quad T_1 \quad T_2}^{s_1=\text{ color flip}} \underbrace{\quad T_3 \quad}_{s_2=\text{ single rotate}} \overbrace{T_4 \quad T_5}^{s_3=\text{ insert node}}$$

The node at which each transformation is applied is also recorded.

4. Compare the solution sequence (S) with the student's solution (s) elementwise. If the two match completely, the student is correct. If at any point there is a mismatch ($S_i \neq s_i$), look for any alternative solution (S') where the step at index i can be s_i (for example, *color root black* could occur at different places in a correct solution).

   - If (S') exists, replace (S) with (S') and continue the comparison.
   - Otherwise, the student is incorrect. Further examine the difference between S_i and s_i to deduce the error type.

5. Record the error contexts (if any). In insertion problems, the context is the node at which the error occurred, along with its grandparent, parent and children (if they exist). In deletion problems, the context also includes the sibling and sibling's children. This definition is based on which nodes are included in the preconditions of insertion and deletion transformations.

The grading algorithm uses the constraints imposed on binary search trees (ordering and relationship of nodes) and its knowledge of red-black trees to construct the canonical solution

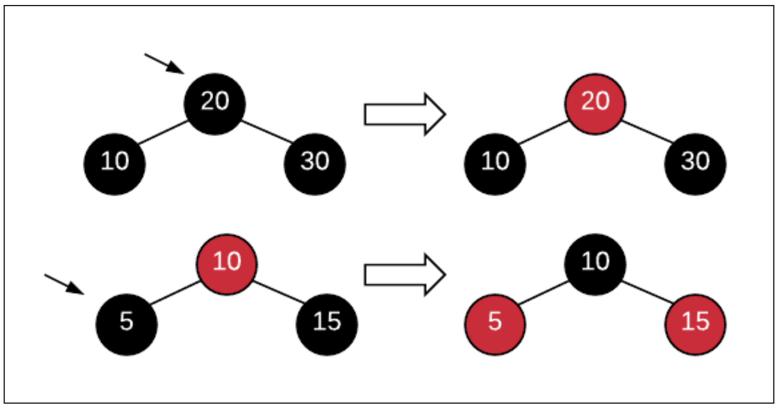


Figure 3.3: Different contexts of color flip in tree deletion. Color flip at root (top) only switches its color, whereas color flip at non-root (bottom) also changes the parent and sibling.

sequence. The key assumption is that the algorithm has a sufficient set of transformations (both primitive and those that are combinations of primitives) to recognize all transformations that a student might make. If there is a step that the system does not recognize, then the step is skipped and the next step is checked to see if it is the result of some step or combination of steps. The result of the comparison steps is a sequence of transformations that the algorithm has identified. In Step 4, this sequence is compared to the sequence from the solution to try and find a match taking into account that a different ordering of transformations can also lead to a solution. Essentially, we first try to infer the steps that the student has attempted from the raw input trees; these steps, along with the problem prompt, are then input to the tutor grading module. As we will discuss later on, these heuristics are sufficient to analyze most of the student answers recorded in this study.

While single step detections are rather straightforward, and can be handled by brute-force, multiple step detections are based on actual data on what steps the students tend to combine (Table 3.1). When students are proficient in the domain knowledge, which usually occurs in the post test, they are able to visualize the resulting tree in their mind. Then they tend to simply overwrite the current tree on screen with the resulting tree. Having the capability to detect when students combine multiple steps is therefore an important factor in reducing the proportion of unrecognized transformations. Note that because all input trees have at most 31 nodes, grading can execute and finish almost immediately. Therefore the grading results do not have to be saved to the database; they can simply be computed every time the result page is loaded.

The output of the grading algorithm is then used to construct the topology of the following Bayesian networks.

3.2 Domain-general network

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 all contexts, and Context-Rule node, which conveys rule mastery in a specific context. Both have boolean values, *True* for mastered and *False* for unmastered. The Rule nodes are independent of each other and each Rule node is connected to its corresponding Context-Rule nodes as depicted in Figure 3.4. The conditional probability table of each Context-Rule given its parent Rule is:

(1) $P(\text{Context}_i = T \mid \text{Rule} = T) = 1$, because by definition, mastery of a rule means being able to apply it in all contexts.

(2) $P(\text{Context}_i = T \mid \text{Rule} = F) = \text{diff}_i$, the difficulty of context i, or the number of times the student makes an error in context i divided by the total number of times that context i appears in the test.

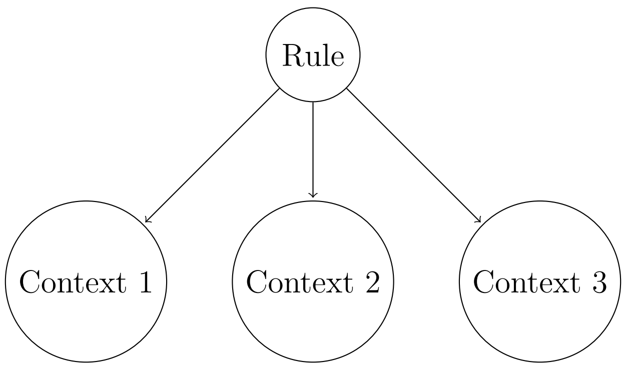


Figure 3.4: The subgraph of a Rule and its corresponding Context-Rules. The domain-general network contains several such subgraphs, one for each Rule.

The network is constructed in real time right after the assessment of the pre-test; each tree transformation gets one representative Rule node, while the Context-Rule nodes are based on the error contexts identified by the grading algorithm. This is an improvement from ANDES' network, whose nodes are specified by experts beforehand as part of the domain knowledge. The mechanism to dynamically generate the network structure would allow each student to have an individualized model and the tutor's framework to be more applicable in other domains.

As an example, consider a portion of a student's domain-general network after the insertion pre-test (Figure 3.5). Every time she makes an error, the surrounding context is recorded and added to the domain-general network; if that context is already present, its difficulty will get updated instead.

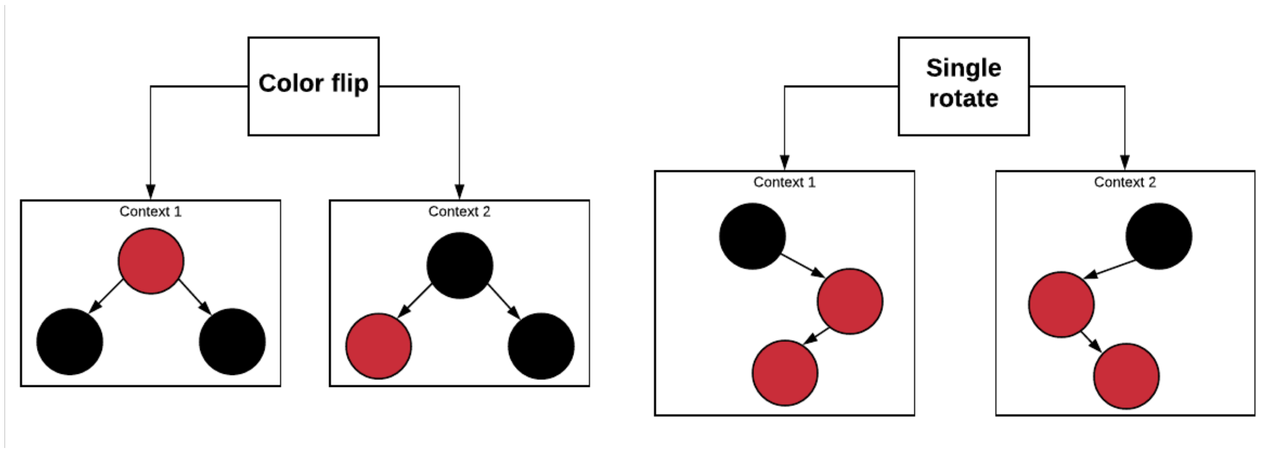


Figure 3.5: An example domain-general network with two Rules: color flip and single rotate.

The left side of the graph indicates two error contexts of *color flip*. Context 1 likely results from mistaking the precondition of *color flip* in insertion for that in deletion; Context 2 shows that the student incorrectly believes the current node having one red child already leads to *color flip*. The right side of the graph shows two error contexts of *single rotate*. Both of these indicate that the student mistakes the precondition of *double rotate* for that of *single rotate*. In fact, we can see the two are symmetric and may as well be merged into one; however, the system records these contexts automatically while having no knowledge of "symmetry," so it still considers them as distinct. Subsequent analysis shows that, even while having some contexts isomorphic to each other, the student model still yields good accuracy in performance prediction. An interesting future work would be to see if having a mechanism to merge isomorphic contexts could improve such accuracy in any significant way.

3.3 Task-specific network

The task-specific network encodes the student's rule mastery in a specific exercise and is automatically generated by the solver module along with the exercise solution. We employ three kinds of nodes: Context-Rule, Fact and Rule-Application[3]. Each Fact node expresses a property of the current tree, e.g., "the current node is black" or "the parent node is red". The Rule-Application node has a boolean value, which is set to True if the student applies the rule correctly, and False otherwise. Figure 3.6 shows the node connections in the network. In essence, the system analyzes the current context, expressed by the Fact nodes, to bring up the corresponding

[3]The ANDES network also has Goal and Strategy nodes because it is designed for additional purposes, including plan recognition and self-explained examples. However, in the scope of this work, I am only concerned with knowledge tracing.

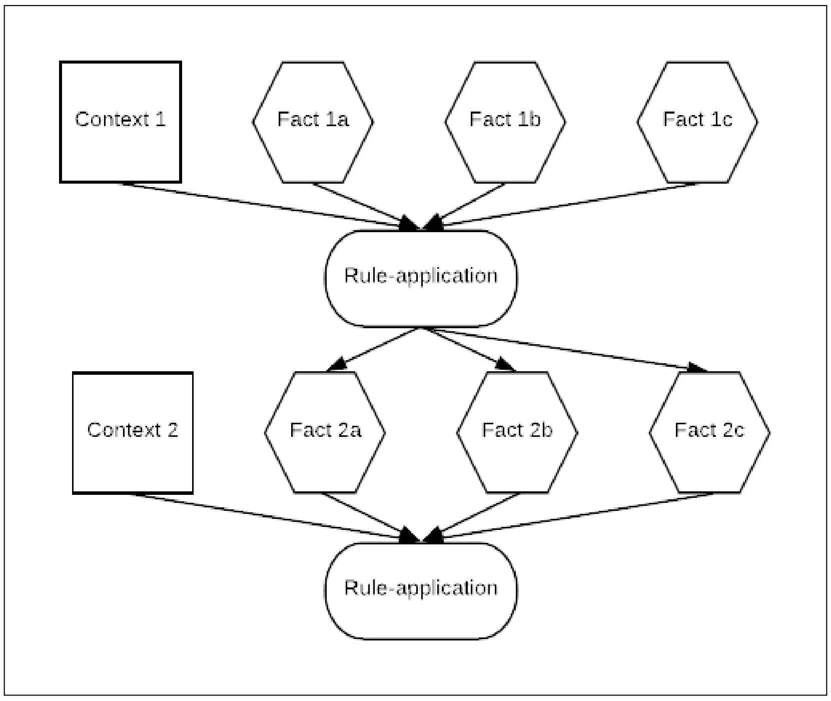


Figure 3.6: The task-specific network.

Context-Rule node. 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 has improved. This mechanism is expressed by the weight of the edge leading to each Rule-Application node. For efficiency, instead of specifying a full conditional probability table, we only state two rules:

(1) $P(\text{Rule-Application} = T \mid \text{all parents} = T) = 1 - P(S)$.

(2) $P(\text{Rule-Application} = T \mid \text{at least one parent} = F) = P(G)$.

Here $P(S)$ and $P(G)$ are respectively the slip and guess probabilities, which we set to an initial value of 20%. Equation (1) states that if the student sees the current context and knows the rule to apply, she should be able to perform a correct rule application, unless she makes a careless mistake (slips). Likewise, (2) states that if the student either is unaware of the current context or does not know the rule, she is unlikely to perform the correct action without making a guess. Observations from past studies show that some students accidentally picked the wrong choice from the drop-down menu despite knowing the answer, whereas others tried to game the system by submitting every possible choice until they were correct. The probabilities $P(S)$ and $P(G)$ are introduced to account for these cases; they are also part of the BKT parameters.

When the student submits a correct answer, the Rule-Application node is set to *True*, and the posterior probability of its parent Context-Rule is updated. The system then performs the selected application, resulting in a new state of the current tree with different properties (Facts 2a, 2b and 2c), and the next Context-Rule (Context 2) is taken into account. At the end of each exercise, the task-specific network is discarded, but the probabilities of all Context-Rules are saved to the domain-general network, so that they can be used as priors for the next time these contexts appear. This process is called *roll-up* in ANDES.

3.4 Three methods for evaluation

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 the Red-black Tree Tutor, the system should be able to predict whether the student's next answer is correct or not, based on her performance so far in the tutoring session and in the pre-test. To my knowledge there has not been any prior work on performance prediction in the domain of binary search tree. Therefore, to get a better idea of how well the student model performs, I implemented and evaluated three approaches:

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.

2. **Bayesian model with error contexts.** A two-part Bayesian network, as defined above, is constructed for each student. The system makes its predictions based on the probability value of the Context-Rule node which corresponds to the current context. As defined earlier, context refers to the structure of the subtree surrounding the current node.

3. **Bayesian model with extended error context.** The same Bayesian model functionality in method 2 is used, but the definition of a context is extended to include both the surrounding subtree and the previous operation. The rationale is that 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. It might be the case that how the student performs in the current step also depends on her previous step. With this distinction, there will be more contexts to analyze, and I would like to see how it affects the system's accuracy.

Chapter 4

Evaluation & Results

The proposed approaches were evaluated on data from students in CS 150 at Lafayette College. The data was taken from four semesters - Fall 2016, Spring 2017, Fall 2017 and Spring 2018. The semester enrollments are 29, 50, 26, and 33, respectively. Insertion and deletion concepts are taught separately, each following the same timeline:

1. Week 1 - lectures on the material (2.5 hours)

2. Week 2, day 1 - pre-test (0.5 hour)

3. Week 2, day 1 - use of tutoring system (1 hour)

4. Week 2, day 2 - post-test (0.5 hour)

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 nine numbers starting from an empty tree. Similarly, problems in the deletion tutor require students to delete all values from an initial tree with nine nodes. The number of questions in each session is listed in Table 4.1.

	Pre-test	**Tutor**	**Post-test**
Insertion	4	20	4
Deletion	7	25 (F2017, S2018) 20 (other semesters)	7

Table 4.1: Number of questions in each session.

Semester	**Application**	**Selection**	**Unrecognized**
Pre F2016	19	62	7
Post F2016	18	37	2
Pre S2017	54	79	13
Post S2017	34	42	6
Pre F2017	4	40	4
Post F2017	8	20	4
Pre S2018	30	45	5
Post S2018	12	25	4

Semester	**Application**	**Selection**	**Unrecognized**
Pre F2016	22	76	9
Post F2016	20	45	6
Pre S2017	50	109	14
Post S2017	33	80	13
Pre F2017	19	53	11
Post F2017	8	51	3
Pre S2018	25	46	5
Post S2018	10	25	2

Table 4.2: Breakdown of student errors for insertion problems (top) and deletion (bottom).

4.1 Assessment of students' test performance

While not the focus of this thesis work, it is useful to first compare the students' performance in the pre and post tests to see if Xhakaj's results about student improvement [46] can be reproduced, using the automated grading algorithm's analysis.

For individual student improvement, I 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} $(M = 1.72, SD = 1.35)$; $t(137) = -8.23,\ p = 2.95 \cdot 10^{-13}$.
- In tree 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.

For cumulative data, Table 4.2 show the total number of errors made in the pre and post tests in each semester across all students. The number of *unrecognized* errors is approximately 10% in the pre-tests and 4% in the post-tests. I analyzed the unrecognized errors by hand and found that in most cases I (the human instructor) was also unable to determine the error type. Many of these errors were generated by students who were "gaming" the system to finish the tests faster by making random selections and then submitting the random answer. The data show that the most common error was committed when selecting the appropriate transformation to apply (from the Selection column). This error is caused by the student either (1) selecting the wrong node as the current node or (2) not correctly recognizing the preconditions for each transformation. The data format provided in the student answers, in the form of raw tree sequences, is insufficient to disambiguate between those two errors.

Comparing the difference between the number of errors in the pre-test and that in the post-test across semesters (Figure 4.1, 4.2), we see that the errors in node selection all decrease; in insertion exercises there is a steady 50% reduction from pre test to post test, whereas the differences vary more in deletion exercises. The number of unrecognized errors also decrease across all semesters, suggesting that students are less likely to game the system once they have finished the tutor and understood the materials; however, due to the grading algorithm being able to recognize more than 90% of the answers, there are not many unrecognized errors across both test conditions.

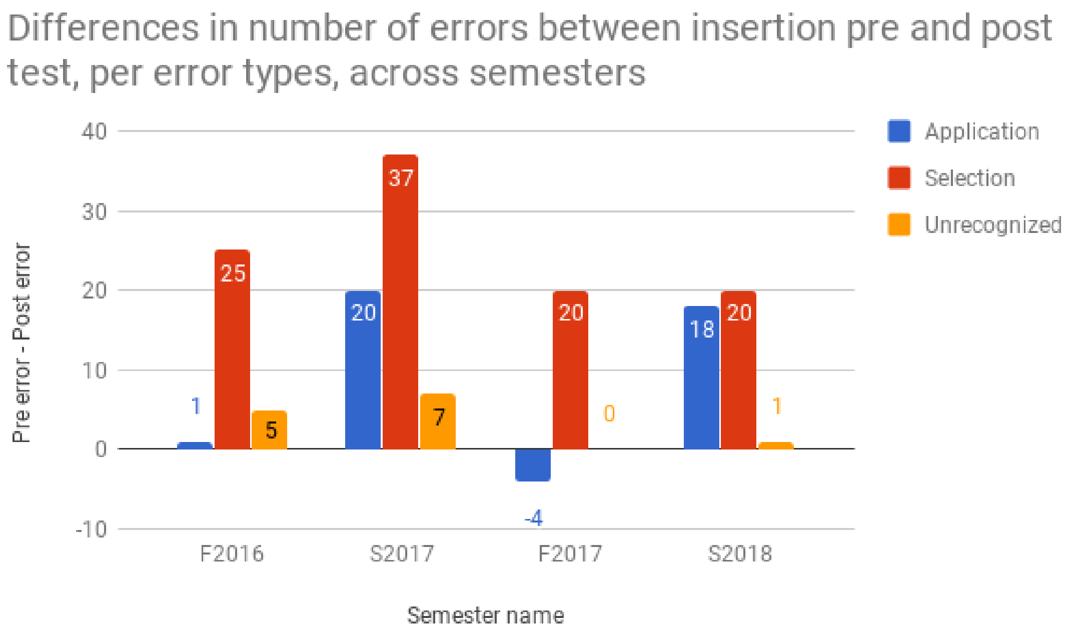


Figure 4.1: Insertion pre-test and post-test error count comparison.

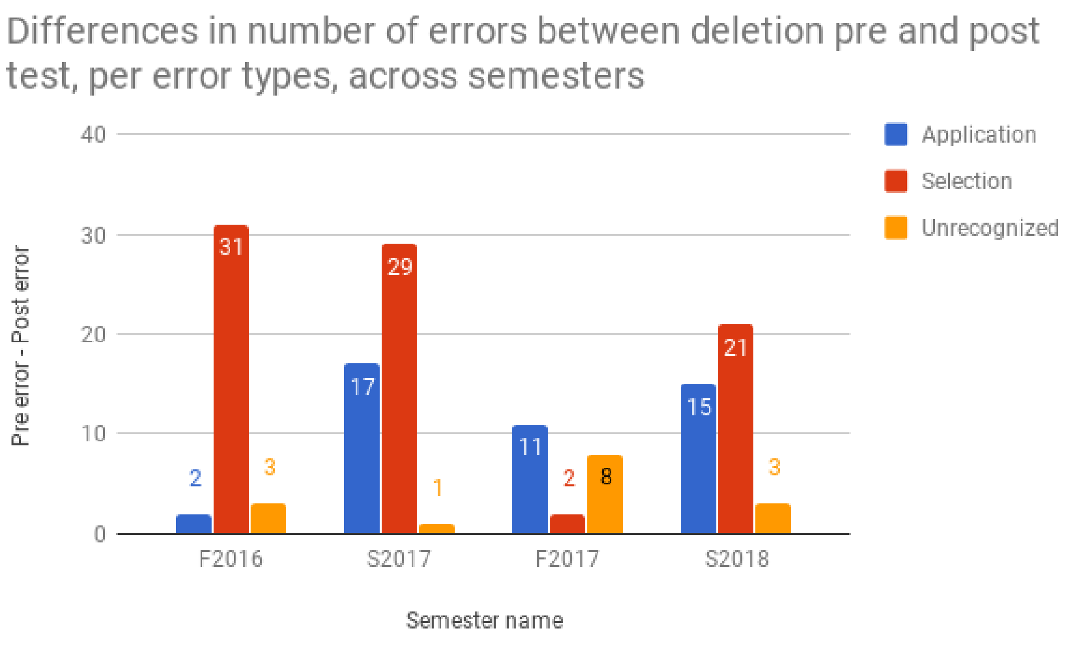


Figure 4.2: Deletion pre-test and post-test error count comparison.

Interestingly, the number of errors in applications do not seem to decrease by much; we only see a difference of 1 in Fall 2016 insertion, 2 in Fall 2016 deletion, and even −4 in Fall 2017 insertion (in this case, the number of error *increases* from pre test to post test). To understand why this is the case, I further broke down the data by the type of transformations (Table 4.3). The largest number of errors were made in applying the *color flip* transformation. This is one of the simpler transformations in that the colors of three nodes (current node, two children) are flipped. Most of the errors occurred due to the students not recognizing the applicable preconditions and selecting the transformation. Note that the grading tool only grades up until the first error so that if students were to make subsequent errors those errors would not be detected. This explains why the errors in *single rotation* and *double rotation* do not appear to decrease significantly (in some cases they increase) between the pre and post test. In the pre-test, because most students forgot about *color flip*, they did not have many opportunities to apply *single rotation* or *double rotation*, resulting in few application errors reported. In the post-test, students did learn to perform *color flip*, which them 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%. Overall these findings are consistent with those reported in Xhakaj's thesis.

Deletion operations are more complex than insertion operations and, as an added level of

Semester	Color Flip	Single Rot	Double Rot	Insertion	Recolor Root	Unrecog
Pre F16	42	19	13	12	2	7
Post F16	15	16	17	6	1	2
Pre S17	67	25	19	30	3	13
Post S17	26	24	19	7	2	6
Pre F17	20	11	5	2	8	4
Post F17	12	6	9	3	1	4
Pre S18	32	15	10	5	7	3
Post S18	18	8	7	2	4	2

Table 4.3: Errors for each type of insertion operation

difficulty, many of the operations share the same name as the insertion operations even though the preconditions and semantics are different. Just like for insertion, the most common error was committed when selecting the appropriate transformation to apply. Table 4.4 shows a breakdown of the data by the type of transformation. The largest number of errors were made in applying the *drop & rotate* transformation. This is a case that is poorly explained and illustrated in the textbook. Using similar analysis, we also show the the difference between the number of errors in the pre-test and post-test, per transformations, across semesters in Figure 4.4.

Semester	Color Flip	Single Rotate	Double Rotate	Deletion	Recolor Root	Switch Value	Drop & Rotate	Unrecog
Pre F16	1	15	19	4	6	11	50	9
Post F16	2	10	10	3	12	3	31	6
Pre S17	3	37	36	2	3	20	72	14
Post S17	1	18	22	2	10	5	63	13
Pre F17	0	14	14	0	1	19	35	11
Post F17	3	5	4	1	12	3	33	3
Pre S18	2	11	13	1	3	10	37	15
Post S18	2	5	5	0	8	0	27	6

Table 4.4: Errors for each type of deletion operation

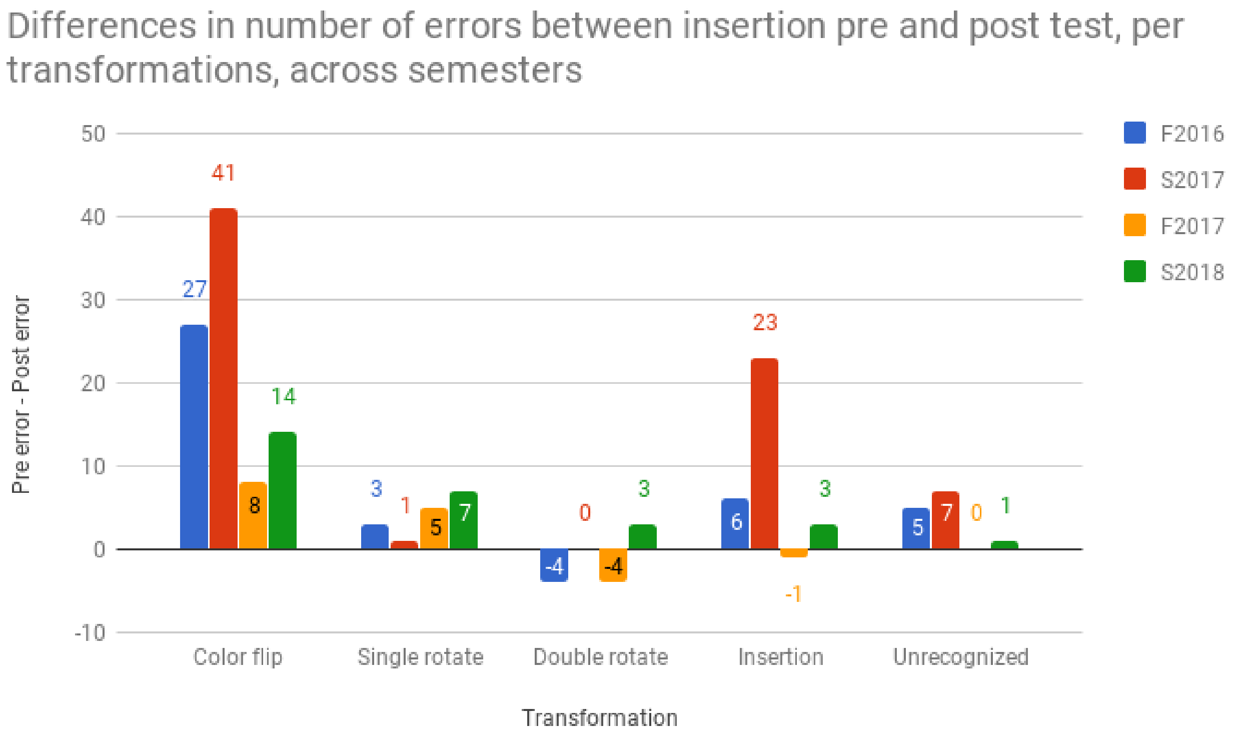


Figure 4.3: Insertion pre-test and post-test error count comparison, per transformations.

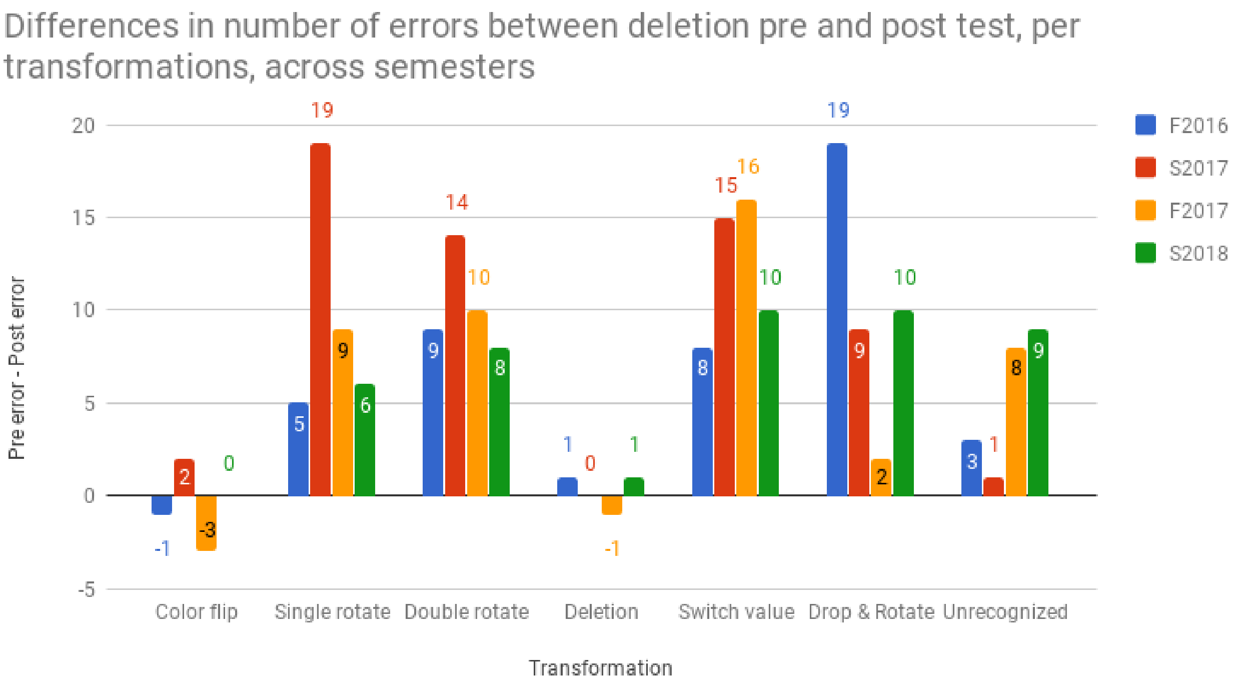


Figure 4.4: Deletion pre-test and post-test error count comparison, per transformations.

4.2 Modeling of students' performance on the tutor

Each time the student submits an answer, the tutoring system attempts to predict whether that answer is correct, using one of the three methods outlined earlier. 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.

4.2.1 Baseline prediction

The data for predicting with fixed probability $p = 0.5$ are shown in Table 4.5. 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. The resulting accuracy also approximates 50% with small variances.

Insertion	**F2016**	**S2017**	**F2017**	**S2018**
Correct/Total	187/372	200/399	184/371	179/375
Accuracy	50%	51%	51%	51%
Stdev Acc	5%	6%	4%	5%
Min Acc	41%	41%	43%	43%
Max Acc	60%	58%	60%	57%

Deletion	**F2016**	**S2017**	**F2017**	**S2018**
Correct/Total	188/383	195/383	267/461	262/472
Accuracy	47%	50%	52%	53%
Stdev Acc	4%	6%	4%	5%
Min Acc	44%	42%	45%	42%
Max Acc	59%	58%	60%	61%

Table 4.5: System's baseline accuracy on the insertion and deletion tutor.

4.2.2 Bayesian model with error contexts

We evaluate the Bayesian student model on both the insertion tutor and deletion tutor (Table 4.6). Note that because there are five more exercises in the deletion tutor of Fall 2017 and Spring

2018, the number of answers submitted (and the number of predictions) in these two semesters are higher than in 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 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.

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

4.2.3 Bayesian model with extended error contexts

Table 4.7 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). A comparison of the three prediction methods' average accuracy (Figure 4.5, 4.6) shows that 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.

I then performed additional analysis in this direction to see whether there is room for improvement and what problem-solving patterns students might have. Table 4.8 breaks down the accuracy in more details; each prediction is categorized as either correct, false positive or false

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

negative. 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, when 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.

Next, I computed the cumulative statistics for each semester. Specifically, I would like to know the transformations involved in the answers that the system can predict accurately and in those that the system cannot. Table 4.9 and Figure 4.7 breaks down this information from Fall 2017 based on the three categories *Correct, False Positive* and *False Negative* mentioned above. Here the tree insertion transformations of interest are *Insert node, Color flip, Single rotate*, and *Double rotate.* Data from the other three semesters are also similar. Table 4.10 and Figure 4.8 present the same kind of data for the deletion tutor in Fall 2017. Here the tree transformations of interest are *Delete node, Color flip, Single rotate, Double rotate, Drop rotate* and *Switch value.*

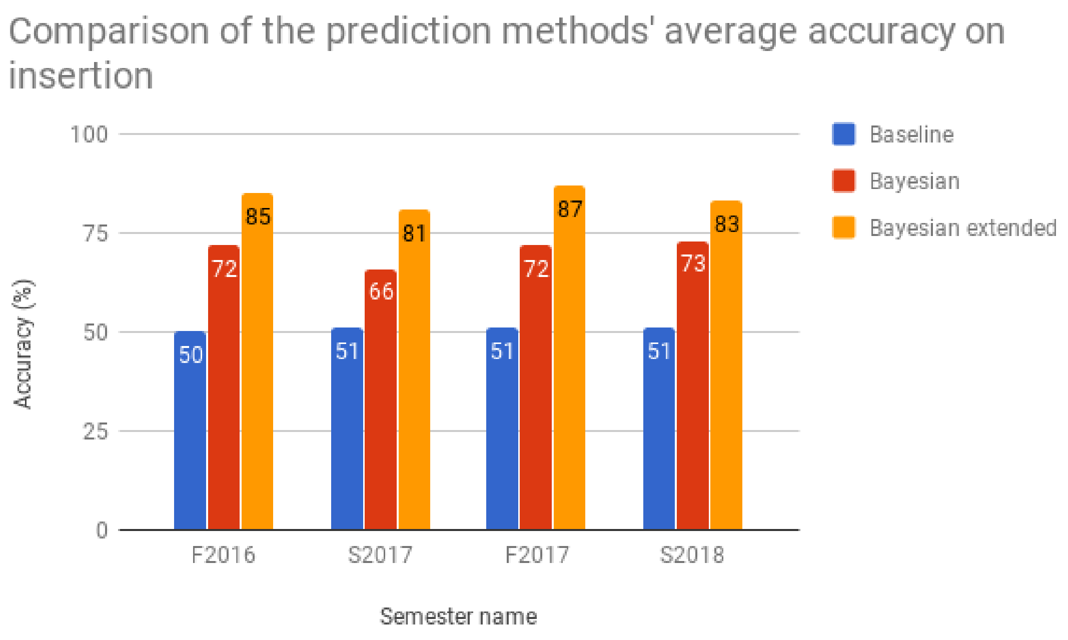


Figure 4.5: Accuracy comparison on the insertion tutor.

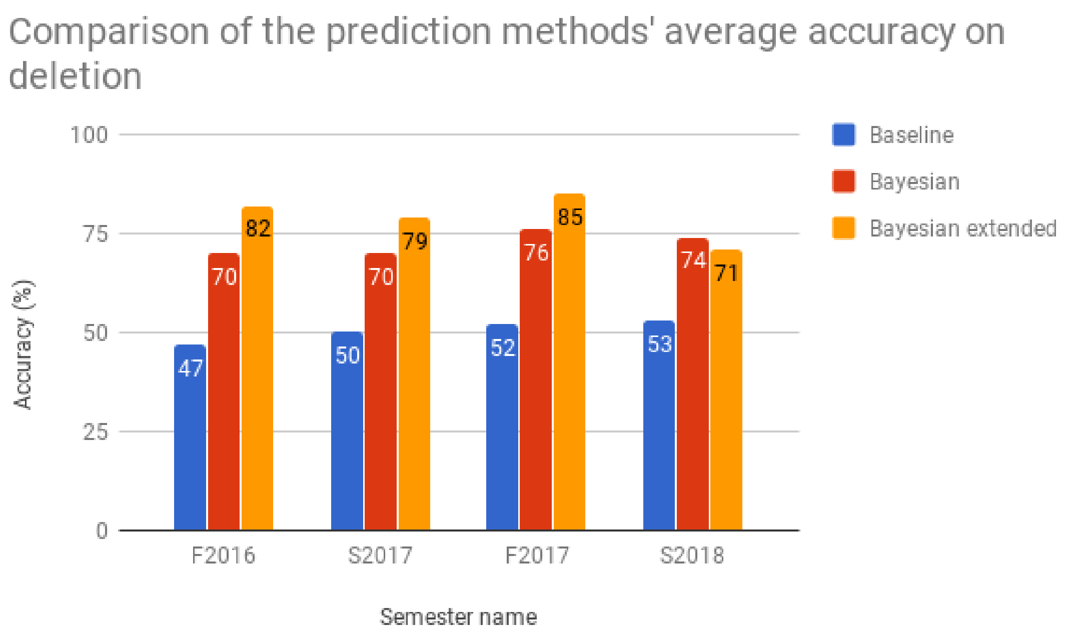


Figure 4.6: Accuracy comparison on the deletion tutor.

I also looked 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 4.11 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

Insertion	**F2016**	**S2017**	**F2017**
Correct	85%	81%	87%
False Positive	11%	14%	10%
False Negative	4%	5%	3%

Deletion	**F2016**	**S2017**	**F2017**
Correct	82%	79%	85%
False Positive	15%	16%	13%
False Negative	3%	5%	2%

Table 4.8: Prediction result categorization on the insertion tutor (top) and deletion tutor (bottom), averaged by students.

	Insert	**Color Flip**	**Single Rotate**	**Double Rotate**
Correct	3353 (90%)	792 (73%)	331 (79%)	348 (80%)
False positive	327 (9%)	229 (21%)	59 (14%)	66 (15%)
False negative	53 (1%)	71 (6%)	31 (7%)	23 (5%)
Total	3733	1092	421	437

Table 4.9: System's prediction result categorization for insertion tutor, cumulative in Fall 2017.

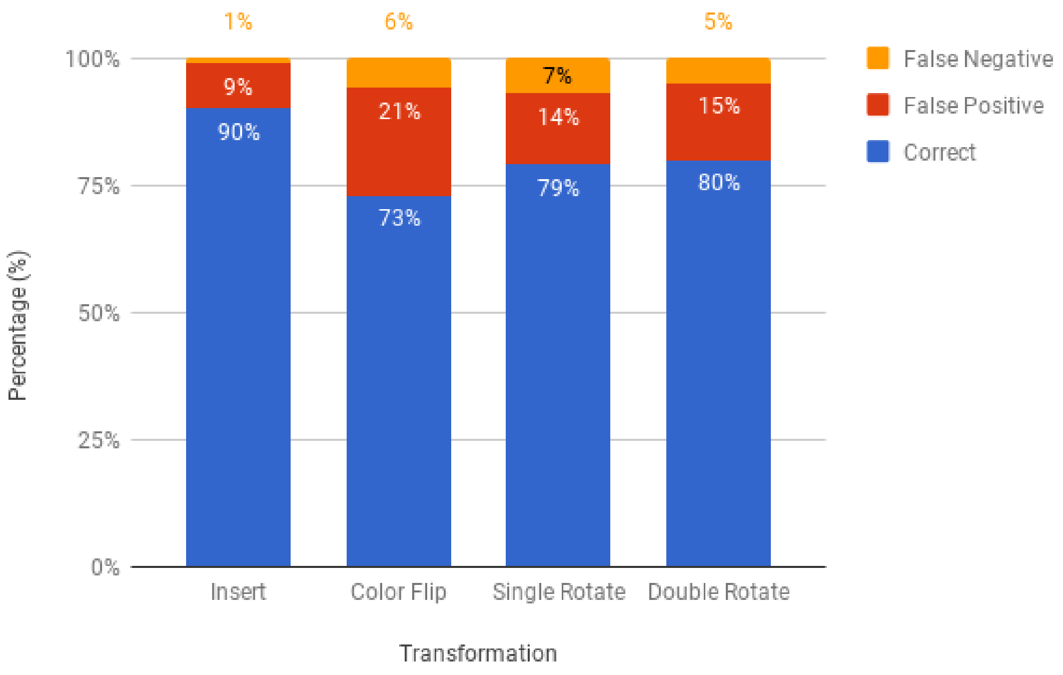


Figure 4.7: Visualization of prediction result categorization for insertion tutor in Fall 2017.

	Delete	**Color Flip**	**Single Rotate**	**Double Rotate**	**Drop Rotate**	**Switch Value**
Correct	3413 (89%)	786 (71%)	341 (74%)	367 (80%)	292 (68%)	795 (95%)
False positive	385 (10%)	243 (22%)	79 (17%)	54 (12%)	101 (24%)	27 (3%)
False negative	52 (1%)	78 (8%)	41 (9%)	33 (7%)	36 (8%)	15 (2%)
Total	3850	1107	461	454	429	837

Table 4.10: System's prediction result categorization for deletion tutor, cumulative in Fall 2017.

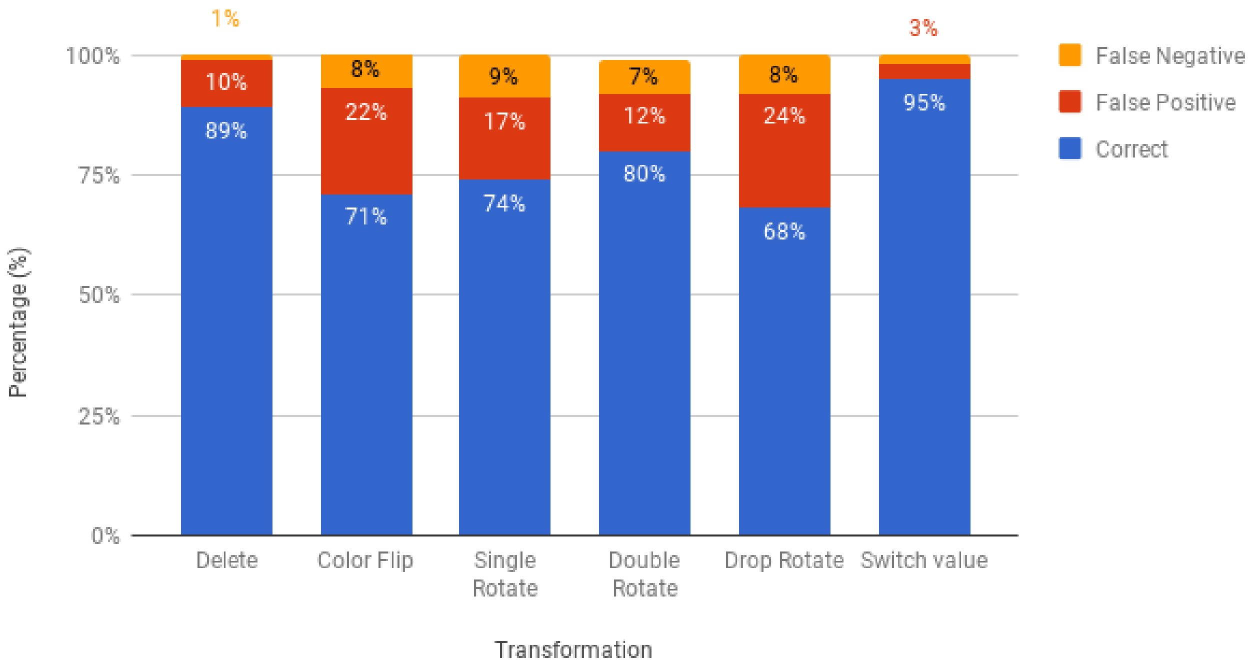


Figure 4.8: Visualization of prediction result categorization for deletion tutor in Fall 2017.

that in most tree insertion problems, the final step is to insert a new node at a leaf node's child; however, sometimes a subsequent rotation at the newly inserted node is also required, which students tend to forget about. 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 case in Table 4.11 (top) represent this case. In general, from my teaching experience, all four cases occur very often, but this is the first time we obtain a relative ranking of their frequencies.

Table 4.11 (bottom) also shows that, in red-black tree deletion, students are most likely to make mistakes in node deletion 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

Transformation	**Previous Transformation**	**Count**
SingleR	Insert	90
DoubleR	Insert	72
SingleR	Cflip	65
DoubleR	Cflip	50

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

Table 4.11: Most common pairs of insertion transformations in students' errors across three semesters in insertion (top) and deletion (bottom).

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).

Chapter 5

Conclusion & Future Works

This thesis work explored the process of constructing an Intelligent Tutoring System called Red-Black Tree Tutor. The system has been used at Lafayette College for four semesters and serves as an effective platform to explore research questions pertaining to teaching and learning in a complicated graphical domain such as balanced binary search tree. In particular,

- The system has successfully replicated the results of Xhakaj's thesis [46], confirming that the granularity approach and the associated tutor interface are effective in improving students' learning gains. It also automates the test procedures by offering an online non-scaffolded testing environment and a grading algorithm that can handle mostly unconstrained inputs in this setting. The algorithm not only assigns a numerical score for each submission but also identifies the first error in 90% of the cases and outputs class-wide statistics comparing students' performance in the pre and post test, with respect to each tree transformation.

- I have shown that a standard Bayesian student model can still be effective in the domain of red-black trees, where a direct mapping from a student answer to its underlying knowledge content is not obvious. An important assumption here is the distinction between understanding of a rule (transformation) in general and understanding of such rule in a specific context. By keeping track of students' errors and their contexts, the model is able to predict students' accuracy throughout the tutoring session with good accuracy ($\approx 70\%$). Extending the error contexts to include the previous transformation, in addition to the current subtree, further improves this accuracy (85%).

- Mining student logs has revealed common patterns about students' problem-solving behaviors. For example, whether students make a mistake on one transformation may depend on the previous transformation (Table 4.11). Certain pairs of transformations, such as (*Insert, Single Rotate*) or (*Switch Value, Delete*) are much more error-prone than others. This information provides valuable feedback to instructors who can then adjust the instructional content to include common errors that students should avoid.

The Red-black Tree Tutor also opens up several exciting future directions. First and foremost, the student model has set up a solid foundation for the implementation of an adaptive tutoring system, which, as outlined in the introduction, is a prominent feature. 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 further 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.

Bibliography

[1] Play framework. `https://www.playframework.com/`.

[2] Aleven, V., Mclaren, B. M., Sewall, J., and Koedinger, K. R. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education 19*, 2 (2009), 105–154.

[3] Aleven, V., Sewall, J., Popescu, O., Xhakaj, F., Chand, D., Baker, R., Wang, Y., Siemens, G., Rosé, C., and Gasevic, D. The beginning of a beautiful friendship? intelligent tutoring systems and moocs. In *International Conference on Artificial Intelligence in Education* (2015), Springer, pp. 525–528.

[4] Andrade, E. L. M., Mercado, C. A. A., and Reynoso, J. M. G. Learning data structures using multimedia-interactive systems. *Communications of the IIMA 8*, 3 (2008), 25.

[5] Bagheri, M. M. Intelligent and adaptive tutoring systems: How to integrate learners. *International Journal of Education 7*, 2 (2015), 1–16.

[6] Ben-Simon, A., Budescu, D. V., and Nevo, B. A comparative study of measures of partial knowledge in multiple-choice tests. *Applied Psychological Measurement 21*, 1 (1997), 65–88.

[7] Bennett, R. E., et al. Toward a framework for constructed-response items.

[8] Bloom, B. S. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher 13*, 6 (1984), 4–16.

[9] Chi, M. T., Siler, S. A., Jeong, H., Yamauchi, T., and Hausmann, R. G. Learning from human tutoring. *Cognitive Science 25*, 4 (2001), 471–533.

[10] Collins, A. Goals and strategies of inquiry teachers. *Advances in instructional psychology 2* (1982), 65–119.

[11] Conati, C. Bayesian student modeling. *Advances in intelligent tutoring systems* (2010), 281–299.

[12] 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.

[13] Corbett, A. T., and Anderson, J. R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction 4*, 4 (1994), 253–278.

[14] Corbett, A. T., Koedinger, K. R., and Anderson, J. R. Intelligent tutoring systems. In *Handbook of Human-Computer Interaction (Second Edition)*. Elsevier, 1997, pp. 849–874.

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

[16] Dean, T., and Kanazawa, K. A model for reasoning about persistence and causation. *Computational intelligence 5*, 2 (1989), 142–150.

[17] Elsom-Cook, M. Student modelling in intelligent tutoring systems. *Artificial Intelligence Review 7*, 3-4 (1993), 227–240.

[18] 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.

[19] Fossati, D., Di Eugenio, B., Brown, C., and Ohlsson, S. Learning linked lists: Experiments with the ilist system. In *International Conference on Intelligent Tutoring Systems* (2008), Springer, pp. 80–89.

[20] Gong, Y. *Student modeling in intelligent tutoring systems*. PhD thesis, Worcester Polytechnic Institute, 2014.

[21] Holland, J., Mitrovic, A., and Martin, B. J-latte: a constraint-based tutor for java.

[22] Kodaganallur, V., Weitz, R. R., and Rosenthal, D. A comparison of model-tracing and constraint-based intelligent tutoring paradigms. *International Journal of Artificial Intelligence in Education 15*, 2 (2005), 117–144.

[23] Koedinger, K. R., Anderson, J. R., Hadley, W. H., and Mark, M. A. Intelligent tutoring goes to school in the big city.

[24] Krishnamoorthy, V., Appasamy, B., and Scaffidi, C. Using intelligent tutors to teach students how apis are used for software engineering in practice. *IEEE Transactions on Education 56*, 3 (2013), 355–363.

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

[26] Liew, C. W., Nguyen, H., and Norton, D. J. Assessing student answers to balanced tree problems. In *International Conference on Artificial Intelligence in Education* (2017), Springer, pp. 532–535.

[27] 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.

[28] Liu, Z., Mostafavi, B., and Barnes, T. Combining worked examples and problem solving in a data-driven logic tutor. In *International Conference on Intelligent Tutoring Systems* (2016), Springer, pp. 347–353.

[29] Lunt, B. M., Ekstrom, J. J., Gorka, S., Hislop, G., Kamali, R., Lawson, E., LeBlanc, R., Miller, J., and Reichgelt, H. Curriculum guidelines for undergraduate degree programs in information technology. *Retrieved March 2*, 2009 (2008).

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

[31] McLaren, B. M., van Gog, T., Ganoe, C., Karabinos, M., and Yaron, D. The efficiency of worked examples compared to erroneous examples, tutored problem solving, and problem solving in computer-based learning environments. *Computers in Human Behavior 55* (2016), 87–99.

[32] Mitrovic, A. An intelligent sql tutor on the web. *International Journal of Artificial Intelligence in Education 13*, 2-4 (2003), 173–197.

[33] Mitrovic, A., Mayo, M., Suraweera, P., and Martin, B. Constraint-based tutors: a success story. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (2001), Springer, pp. 931–940.

[34] Nguyen, H., and Liew, C. W. Building student models in a non-scaffolded testing environment. In *International Conference on Intelligent Tutoring Systems* (2018), Springer.

[35] Nguyen, H., and Liew, C. W. Using student logs to build bayesian models of student knowledge and skills. In *International Conference on Educational Data Mining* (2018), Springer.

[36] Nkambou, R., Mizoguchi, R., and Bourdeau, J. *Advances in intelligent tutoring systems*, vol. 308. Springer Science & Business Media, 2010.

[37] Oracle. Treemap. `https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html`.

[38] Rivers, K., and Koedinger, K. R. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education 27*, 1 (2017), 37–64.

[39] Russell, S. J., and Norvig, P. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[40] Self, J. A. Bypassing the intractable problem of student modelling. *Intelligent tutoring systems: At the crossroads of artificial intelligence and education 41* (1990), 1–26.

[41] VanLehn, K., Niu, Z., Siler, S., and Gertner, A. Student modeling from conventional test data: A bayesian approach without priors. In *Intelligent Tutoring Systems* (1998), Springer, pp. 434–443.

[42] Warendorf, K. Adis-an animated data structure intelligent tutoring system on the www. In *Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on* (1997), vol. 2, IEEE, pp. 944–947.

[43] Weiss, M. A., and Hartman, S. *Data structures and problem solving using Java*, vol. 204. Addison-Wesley Boston, MA, 1998.

[44] Weragama, D., and Reye, J. The php intelligent tutoring system. In *International Conference on Artificial Intelligence in Education* (2013), Springer, pp. 583–586.

[45] Woolf, B. P. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning.* Morgan Kaufmann, 2010.

[46] Xhakaj, F. Intelligent tutors and granularity: A new approach to red black trees. 2015.

[47] Xhakaj, F., and Liew, C. W. A new approach to teaching red black tree. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (2015), ACM, pp. 278–283.

[48] 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.

[49] Zimmerman, D. W., and Williams, R. H. A new look at the influence of guessing on the reliability of multiple-choice tests. *Applied Psychological Measurement 27*, 5 (2003), 357–371.