2017-06-08

Creating Systems and Applying Large-Scale Methods to Improve Student Remediation in Online Tutoring Systems in Real-time and at Scale

Douglas A. Selent
Worcester Polytechnic Institute

Creating Systems and Applying Large-Scale Methods to Improve Student Remediation in Online Tutoring Systems in Real-time and at Scale

Ph.D. Dissertation

Douglas Selent

Advisor: Neil Heffernan

Spring 2017

Worcester Polytechnic Institute

Table of contents

Overview

This dissertation consists of the following topics broken up into chapters.

Chapter 1 discusses the paper, "Refining Learning Progression Models with Data Fitting Techniques: Searching for better fitting Learning Progression Models." (Adeji, Selent, et al. 2014). In this work, we were given a dataset from the University of Kansas with student response data on items that were mapped to skills. Our goal was to see if we could simplify the skill graph, under the assumption that there were too many skills, without losing model accuracy using an iterative greedy merge algorithm. We concluded that initially merging skills was beneficial to model accuracy, but after merging too many skills the model accuracy quickly declined. These results were statistically significant but not practically significant, since the decrease in RMSE was negligible.

Chapter 2 continues from Chapter 1, where we examine the effectiveness of our algorithm introduced in (Adeji, Selent, et al. 2014). Our goal was to see under what conditions our algorithm would work at effectively merging the correct skills in a skill graph. We used synthetic data, where we knew the ground truth values, and generated and evaluated several graphs. Our conclusion was that we cannot attempt to merge skills without correct time series data, since some unrelated skills were highly correlated and therefore were correctly being merged by the algorithm even though the skill were not related. An example of this is having an entire second grade class do problems on Algebra and Calculus, which are both different skills. The two skills will appear the same, since the entire class will answer both types of problems incorrectly.

Chapter 3 is about research related to buggy messages. Chapter 3 explains the machine learning algorithm created to discover the generalized incorrect processes student make. Data from ASSISTments was used where the inputs to the problem and the student's answer was used as input to the machine learning algorithm. The algorithm was effective at automatically learning the incorrect processes. The majority of the incorrect processes tend to be the same or within a group of the top three. Students who make the most common wrong answers often quickly fix their mistake without help and proceed through the rest of the problem set without any more mistakes. Students who make less common mistakes tend to struggle for several problems

before completing the problem set. Students rarely make the same mistake twice and usually the second mistake is another common wrong answer.

Chapter 4 uses the machine learning developed and explained in Chapter 3, to created buggy messages and run a randomized controlled trial to compare the buggy messages to existing tutoring. In this experiment, the control group received the existing content in ASSISTments for two problem sets, and the experimental group received buggy messages in addition to the existing content in ASSISTments. Although buggy messages significantly reduced hint usage, they did not improve learning. This implies that they are just as effective as hints, but not better than hints. A summary of these experiments and the results are reported in (Selent & Heffernan, 2014).

Chapter 5 discusses two more randomized controlled experiments that were run on the same problem set with similar designs. Since it was believed that the machine-learned predicted incorrect processes were accurate, more attention was given to the content of the buggy messages. The original buggy messages were designed to be simple plain-text messages roughly seven words long specifically targeting the student's erroneous logic. In these experiments, I attempted to improve the content of the buggy messages from the previous randomized controlled experiment discusses in Chapter 4. In the first experiment of Chapter 5, I tried to improve the content of the buggy messages by providing the context of the logical flaws, the correction, and the reason for the correction to the student on the tutor screen. Unfortunately these buggy messages did not result in any learning gains. In the second experiment, I attempted to use buggy messages in the form of short 20-30 second videos, which I call video messages. Although this idea was really cool, it also resulted in a null result. Not only was this result null, it failed to beat the weakest control condition as opposed to the strongest control condition supplied in prior experiments. The experiments discussed in Chapter 5 are also reported on in (Selent & Heffernan, 2015).

Chapter 6 presents work where Bohao Li and I compared the predictive power of common wrong answers to standard prediction methods KT and PFA (Li & Selent, 2014). This paper did not use my machine learning algorithm to generalize common wrong answers across multiple problems, but instead just used the common wrong answers for each problem individually. The main result of this work showed that all predictive models performed poorly when compared to the baseline of predicting the majority class.

Chapter 7 demonstrates the feasibility of using Bayesian hypothesis testing on randomized controlled experiments run in online tutoring systems. This is work I did with my student Vijaya Dometti at Rivier University and resulted in a poster (Dommeti & Selent, 2017). In this work we applied the Bayesian A/B hypothesis testing created by Alex Deng to a dataset from twenty-two randomized controlled experiments run inside the ASSISTments online learning platform. We extended the Bayesian A/B testing framework to include confidence intervals on the learned prior probability that an experiment has differences between conditions. We conducted a leave-one-experiment-out cross-validation experiment, where we learned a prior on twenty-one experiments and used that prior to perform the hypothesis test on the last experiment. We concluded that twenty-two experiments are not enough to accurately learn a genuine prior (~200 are required), and both Bayesian and Frequentists methods gave similar results.

Chapter 8 presents the PeerASSIST system for crowdsourcing student work in real-time and redistributing that work to peers in need of assistance. This work was submitted in (Selent & Heffernan, 2017). The infrastructure design is discussed in great detail, as many important decisions were made in this design. A walkthrough of how teachers and students can use this system is provided. Statistics on the usage of the system are provided as well. This chapter ends with discussing the results of a randomized controlled experiment, where crowdsourced hints from peers is compared to no hints (control). Some useful information gained from this data collected is that the majority of student work is low quality, this work did not improve student learning performance, student work is rarely distributed more than 1-2 times, and student work is rarely inappropriate.

Chapter 1: Refining Learning Maps with Data Fitting Techniques: Searching for Better Fitting Learning Maps

Abstract

Learning sciences needs quantitative methods for comparing alternative theories of what students are learning. This study investigated the accuracy of a learning map and its utility to predict student responses. Our data included a learning map detailing a hierarchical prerequisite skill graph and student responses to questions developed specifically to assess the concepts and skills represented in the map. Each question aligned to one skill in the map, and each skill had one or more prerequisite skills. Our research goal was to seek improvements to the knowledge representation in the map using an iterative process. We applied a greedy iterative search algorithm to simplify the learning map by merging nodes together. Each successive merge resulted in a model with one skill less than the previous model. We share the results of the revised model, its reliability and reproducibility, and discuss the face validity of the most significant merges.

Introduction

Cognitive models are used to represent how one's knowledge may be organized. As such, they contain descriptions of component pieces of knowledge and connections among the components to indicate how understanding develops in a specified domain (Gierl et al., 2008). Different authors have described various cognitive models, including learning maps (Pavlik & Koedinger, 2009), learning trajectories (Clements & Samara, 2004), and learning hierarchies (Gagné, 1968). Learning maps use linear sequences of learning goals and are useful for instructional planning (Popham, 2011). A learning trajectory includes a learning goal, a developmental progression defining the levels of thinking students pass through as they work toward the defined goal and a set of learning activities or experiences that assist students in reaching the defined goal (Clements & Samara, 2004). As their name implies, learning hierarchies model prerequisite knowledge components in hierarchies, allowing multiple pathways to extend from one prerequisite skill to multiple learning goals (Gagné, 1968).

The learning map extends the notion of a learning hierarchy by representing domain knowledge as a network of component skills and connections, allowing for multiple paths from prerequisites to learning goals. While multiple paths add complexity to the cognitive model, they allow the learning map to represent the potential learning of a broad range of individuals who may experience difficulties traversing certain pathways due to disabilities or particular learning preferences. As such, the learning map provides a flexible model of learning that is consistent with recent advances in universal design for learning (CAST, 2011).

In the present study we examine a small section of the learning map and investigate the effects of permuting the topology of the hierarchy. Skills and concepts are represented by latent nodes in the learning map. Directed edges represent the prerequisite relationship among latent nodes and also represent the relationship between those nodes and their associated test items. We present a simple method for improving the predictive power of the learning map by combining latent nodes. We report our initial results on the fit improvement, stability of the resulting map, and interpretation of the algorithms chosen node combinations.

This work connects with literature on searching for better fitting cognitive models. Several non-hierarchical cognitive models have been developed to represent the relationship between knowledge components (KCs) in the form of prerequisite skill maps. These cognitive models have been developed to help intelligent tutors, as well as experts, determine student mastery of KCs. A number of technical approaches have been developed to evaluate cognitive models developed by domain experts. One approach is Learning Factors Analysis (LFA), developed by Cen, Koedinger and Junker (Cen et al., 2006) to help the Educational Data Mining (EDM) community evaluate different cognitive models. LFA incorporates a statistical model, item difficulty and a combinatorial search to select the model. Our work is different from the flat Item Response Theory (IRT) models presented in (Van der Linden & Hambleton. 1997) in that IRT presents does not deal in any way with hierarchical relationships between knowledge components.

Rafferty and Yudelson showed that a cognitive model with more fine-grained skills does better at predicting student performance for lower-knowledge students (Rafferty & Yudelson, 2007). A simpler model does better at predicting for students with a higher level of knowledge. This is because students with a lower level of knowledge think that slight differences in the problem are

completely different skills, where a higher knowledge student recognizes that these slight differences are still part of the same skill.

There are several different methods for analyzing skills. Tatsuoka introduced the rule space method for representing and determining how well students understood the underlying skills (or rules as the paper calls it) for test items (Tatsuoka, 1983). Additionally, the method is used to identify any erroneous classification or misconceptions of students in responding to test items. Barnes utilized the Q-matrix method from Tatsuoka's rule space method to organize combinations of skills into distinct latent classes and assign students to latent classes based on level of mastery (Barnes, 2005). Additive Factor Models (AFM) also utilize the Q-matrix but with a multiple logistic regression model which predicts student performance based on a number of factors, primarily the number of opportunities a students has to demonstrate a particular skill. Cen reported that AFM did not accurately predict items involving conjunctive skills and hence introduced the Conjunctive Factor Model (CFM) to improve predictions in this area (Cen, 2009). In addition to latent skill cognitive models, item to item knowledge structures have also been learned from empirical data using Bayesian Network structure learning and partial order knowledge structures (Desmarais & Gagnon, 2005).

Our approach to simple merging of skills was inspired by Learning Factors Analysis, which uses a combinatorial search to determine which model best fits student data. The combinatorial search consists of three different types of operations: splitting, merging or adding existing KCs. Splits occur when a knowledge component is determined to be composed of more than one skill, and hence splits into multiple skills. One or more skills are merged if they are determined to be inseparable skills, given student data. The add operation involves the inclusion of a completely new skill to the original map (CAST, 2011).

Other researchers have tried to extend LFA to other subject domains. Leszczenski and Beck introduced a scalable application of the LFA framework in the context of reading knowledge transfer (Leszczenski & Beck, 2007). The problem with this approach is that the search was unstable and could give different results each time the search was run. Instead of determining a student model given an initial human generated model, Li, Cohen, Noboru, and Koedinger proposed a method for automatically generating the KCs from student responses to individual items. Although their method resulted in the best fit among the other candidates, it may not generalize for models with less coarse grained KCs (Li et al, 2011).

Other models have focused on the determination of a student's knowledge of certain skills. Logistic regression has been used to trace multiple sub-skills of a given skill (Xu & Mostow, 2011). Pavlik, Cen, and Koedinger proposed a method for automatically deriving a cognitive model by generating a Q-matrix, which provides a representation of the KCs required for each test item (Pavlik, Cen, & Koedinger, 2009).

In this work we follow the process described by Cen, Koedinger and Junker (Cen et al., 2006). This technique can be used to analyze hypothesized learning maps and consider whether small improvements to the model result in a better fit to the data. In this method two different approaches were studied to determine the best skill map from an initial graph. Cen, Koedinger, and Junker suggested three types of operations, i.e., merges, splits, and adds (Cen et al., 2006). However, in this study, we used only merge operations given the already highly granular quality of our initial, subject matter expert derived learning map.

Initial Learning Map

This study examined a section of the learning map containing 15 concepts and skills related to understanding integers. The map was developed using mathematics educational literature describing how students learn to understand and operate with integers. The set of integers includes the whole numbers and their opposites, presenting many students their first exposure to negative numbers (Van de Walle et al. 2014). Although many students have prior knowledge of negative values within contexts such as debt or temperatures below freezing, they often struggle when first learning to work with negative numbers. Proficiency with integers includes understanding opposite numbers, comparing integers, representing integers on number lines and graphs, and using integers in real world problem contexts. The learning map shown in Figure 1.1 illustrates the component concepts and skills that comprise such understanding. This map suggests that students should learn to identify opposite numbers (M-1104) and integers (M-1289) in preparation for comparing and ordering integers (M-1133, M-1135, M-1140) as well as representing integers on number lines (M-1118, M-1120, M-1108, M-1126) and coordinate planes (M-1122, M-1124). Because integers challenge the initial counting strategies students learned for positive numbers, it is beneficial for students to work with integers in real-world contexts (M-1106, M-1105, M-1127, M-1128) (Van de Walle et al. 2014).

The data for this study was gathered from student responses to 25 test items aligned to the 15 skills shown in the learning map in Figure 1.1. All of the test items were multiple choice questions, with four answer options per question. Each skill was assessed by one or more items. As part of the test development process, subject matter experts confirmed the alignment of each item to its associated skill, meaning that the item was judged by experts to evoke the intended skill. Therefore, when a student answered a test item correctly, we assumed in this study that the student had mastered the skill associated with that test item. Furthermore, due to the hierarchical structure of the learning map, items associated with skills lower in the learning map were assumed to be more difficult, i.e., require more skills, than items associated with skills higher in the learning map.

In addition to the graph, we utilized a data-set containing the responses of 2,846 students answering the same sequence of 25 items in the learning map. All the students were chosen from middle schools in a mid-western state from grades 6 (8%), 7 (49%), 8 (39%) and 10 (4%). The students' responses were dichotomous, '1' for correct and '0' otherwise. Students did not receive any feedback on their responses and therefore did not know if they got the items correct or not. All items were presented in the same order to all of the students. A complete list of these items can be found in Appendix B.

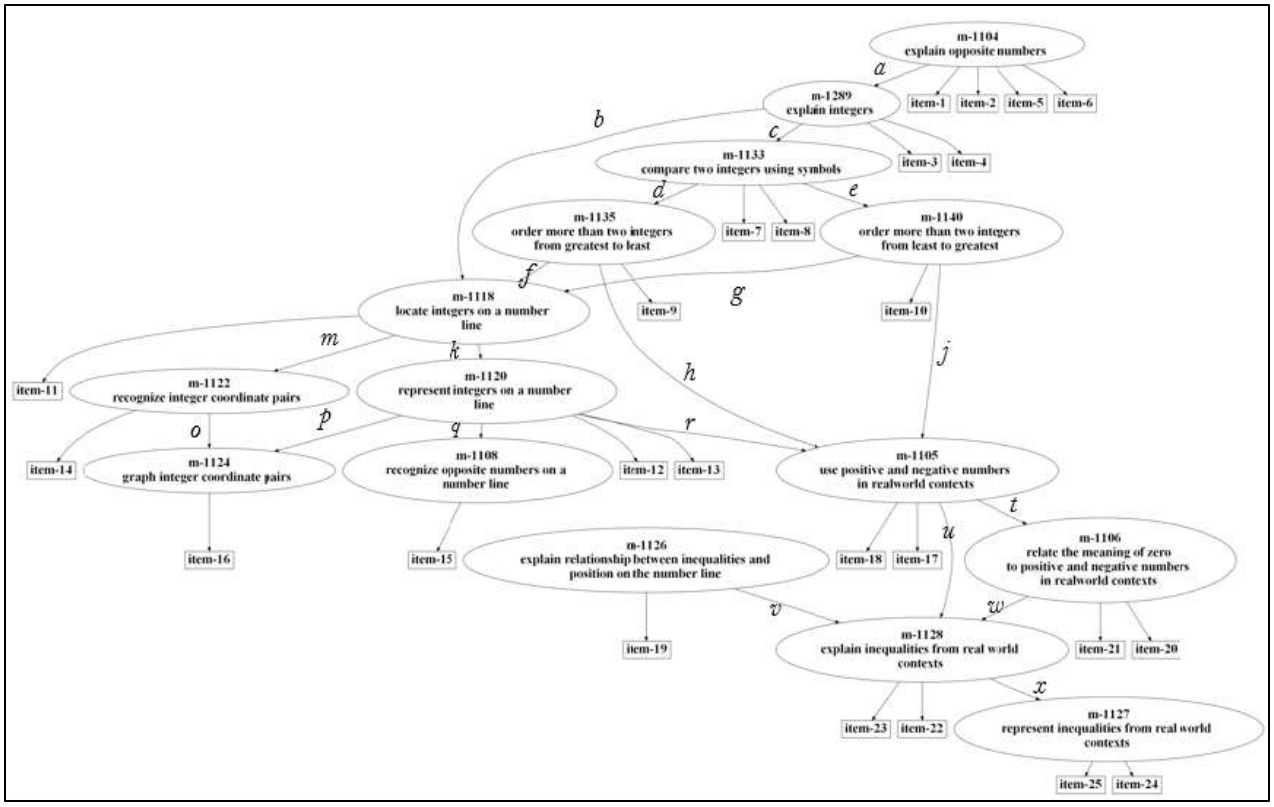


Figure 1.1. The initial learning map that researchers created. Each ellipse represents a "skill" and each rectangle represents a test item. For easy reference, the links are labeled. The labels do not have any specific meaning.

Modeling Technique: How we create a Bayesian Network for the Learning Progression

Given Figure 1.1, there are many ways you could attempt to fit a Bayesian network to such a model. In this section we lay out a few of the issues and the choices we made. To apply a Bayesian modeling approach, we treat each of the skill nodes at latent Boolean variables. Each question is an observable node where we know for each student which questions they answered correctly or incorrectly. For skills without parent skills, such as skill m-1104, the Bayes net will learn the best fitting estimate of that skills knowledge across all the student data. It will learn this and all other parameter using Expectation Maximization, which is the standard way of fitting Bayes nets. We use the Kevin Murphy Bayes Net Toolkit in Matlab to fit all models. To fit skill nodes that have parents, we model such conditional probability tables as latent variables that EM will attempt to fit. For example, to model skill m-1118, we need to learn four values for the conditions. The first row in Table 1.1 shows the probability the student know skill m-1118 given they do not know either of the parent skills. The value of 0.2 is listed here, which was learned through running the expectation maximization algorithm. The second row represents the

probability that the student knows skill m-1118 given that they do not know skill m-1135 but do know skill m-1140. The values in the third column are examples of values that EM will learn. Expectation Maximization starts the fitting process with guesses for values, and then iteratively these values converge to new values. These starting values are the Bayesian "Priors" of the network. Ideally, we would like our approach to not be sensitive to the initial priors. For the mode in Figure 1.1, we have not yet shown that this is true of the techniques we use in this work, but we have done so in similar work. In (Pardos & Heffernan. 2010) we conducted a simulation study were set the gold standard ground truth values and looked to see if regardless of what initial value we started with, EM would learn back the ground truth values.

m-1135	m-1140	m-1118
0	0	0.2
0	1	0.4
1	0	0.5
1	1	0.8

Table 1.1: The conditional probability table that models the probability of knowing skill m-1118 given the probability of knowing m-1135 and m-1140.

While we don't know the real values in the third column, we are assuming that the probably of the last row should be higher than the probability of any of the other rows. The probability that a student knows skill m-1118 given that a student knows both of the parent skills should be higher than the probability of a student knowing skill m-1118 given that the student does not know both of the parent skills. If not knowing the prerequisite skills increases the chance of knowing the post-requisite skill, then the model itself is likely to be invalid and the arcs between those skills should be deleted to get a better fitting model. In this work we do not consider removing arcs and only look at merging skills, although there is no reason not to look at removing arcs as future work.

In modeling it is often a good idea to think about the number of parameters the model requires. Let's look at the number of parameters for the model in Figure 1.1. Given that there are 15 circles, we have to split them out depending upon how many nodes that have zero, one, two or three parents, as nodes that have different numbers of parents will require a different number of parameters. As a general rule, if you have more parents you will require more parameters. If you have no parents you learn one parameter. If you have one parent skill, you will learn two numbers, one number for the probability that you know the post-requisite skill given you know the parent skills, and a second parameter for the probably that you know the post-requisite skill given that you do not know the parent skill. Table 1.1 showed that if you have two parents you need 4 parameters. If you have three parents (m-1105 is the only skill that has three parents in Figure 1.1) you will need 8 parameters. Figure 1.1 has two skills with one parent (m-1104 and m-1126), 9 skills with one parent, three skills with three parents (m-1118, m-1124 and m-1128) and one skill with three parents (m-1105). This results in 2*1+9*2+3*4+1*8=40 parameters. Additionally we modeled each of the 25 question as a Boolean variable (we learned a probably for each questions dependent upon whether they knew the skill.)

Merge Operation

In all of the experiments our sole manipulation of the map was to merge latent nodes. A merge operation occurred when two skills adjacent to each other in the map were combined into one skill. Items from both skills that were merged were reattached to the new single skill. The prerequisites of the constituent skills became prerequisites of the merged skill and the same applied to the post-requisites. An example merge operation on a section of the skill map is shown below. The skill maps before, during, and after the merge operation are shown in Figures 1.2, 1.3, and 1.4. M1289 and M1133 are the skills that were merged into a single skill, named "M1289XM1133". Note that the names of the skill hold no meaning of their own, just as the labels of the arcs between the skills.

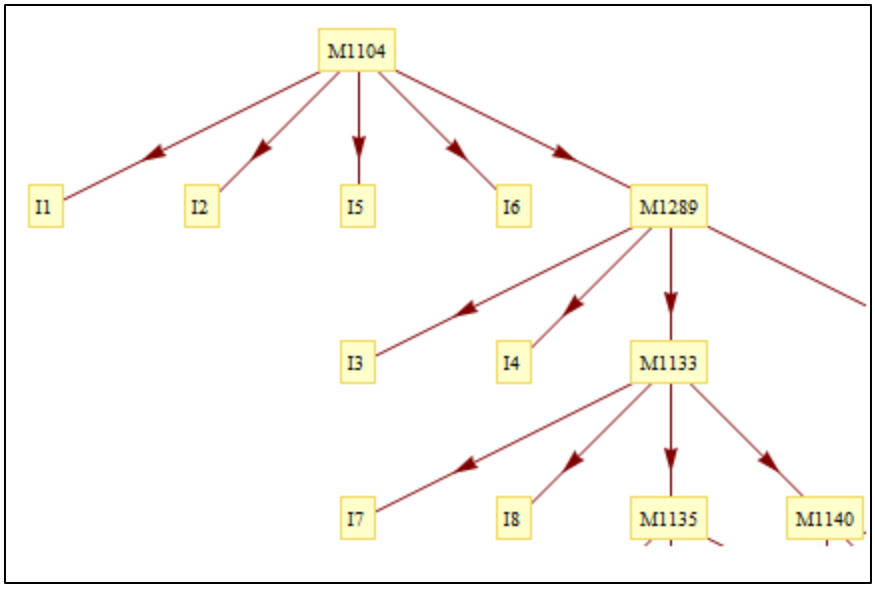


Figure 1.2: What part of the learning progression looked like before the merge operation.

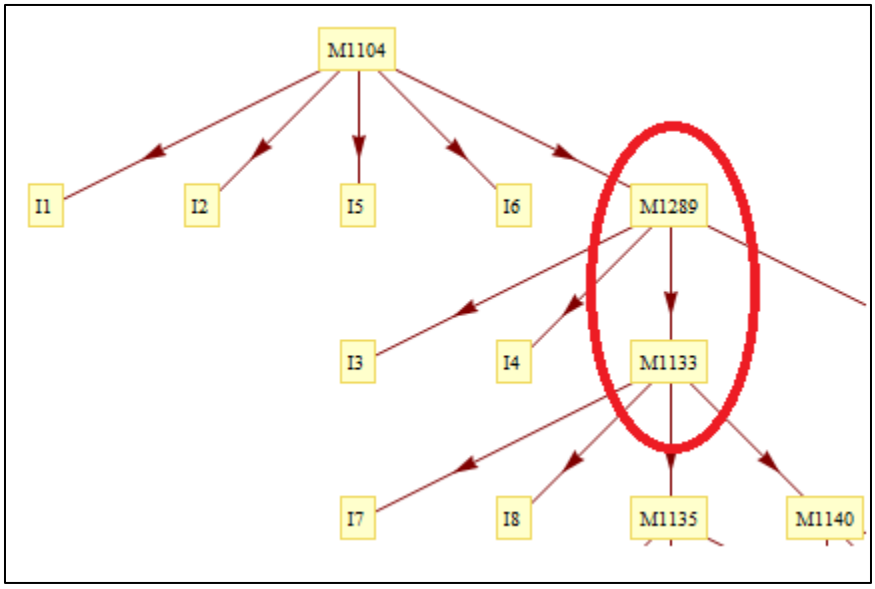


Figure 1.3: Showing two skills that are going to get merged together. Note that I3 & I4 and I7 and I8 will all be grouped together.

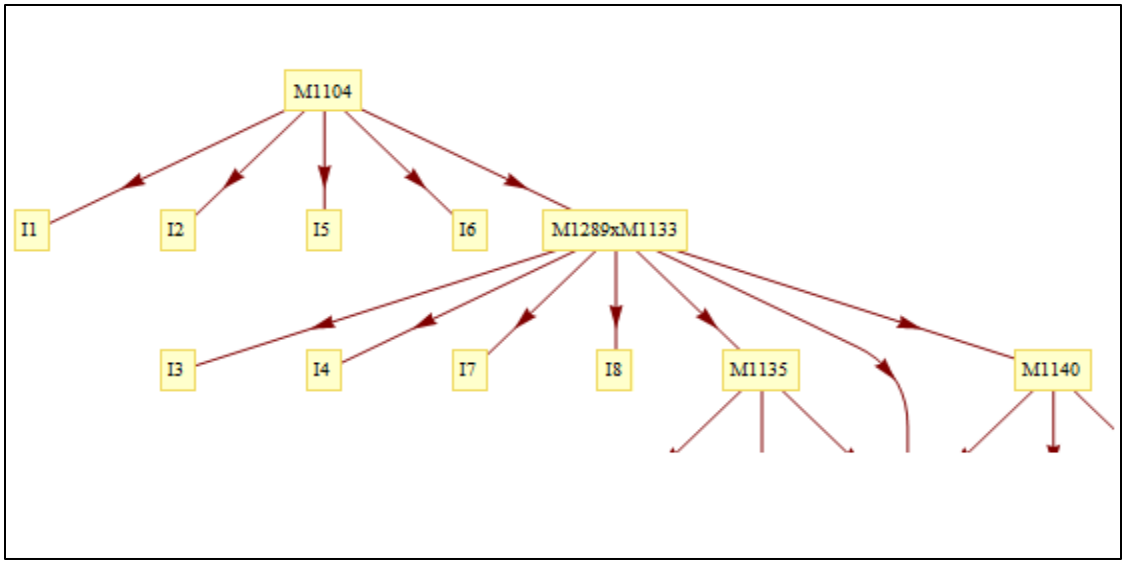


Figure 1.4: After the merge. Note that M1289 and M1133 now have all the questions of each of them mapped to the same node.

The code that does the merging, functions at the item level and not the skill level. The result is conceptually the same. Tables 1.2 - 1.5 show the same merge operation done in Figures 1.2-1.4, but shown as how the merge is done programmatically instead of conceptually. In the merge example skills M1289 and M1133 are being merged. Cells highlighted in blue show all items in skills that are parent skills of the items to be merged. In this example skill M1104 is a parent skill of M1289 and contains the items 1, 2, 5, and 6. Therefore those rows that contain elements that have a '1' are highlighted in blue. This represents that those items are parent items of items 3 and 4. Skill M1289 is also a parent as well as one of the skills being merged. Since skill M1289 is a parent skill of skill M-1133, those cells are also highlighted in blue. Skills M1135, M1140, and M1118 are child skills of one of the skills that are going to be merged (M1289 and M1133). Items in these skills are items 9, 10, and 11. Cells highlighted in red show the rows of the merged skills that have the items of the child skill.

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	1	0	0	1
4	0	0	0	0	0	0	1	1	0	0	1
5	0	0	1	1	0	0	0	0	0	0	0
6	0	0	1	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	0	0	1	1	0
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0

Table 1.2: Merge Example: Item x Item Matrix Before Merging

In order to merge skills M1289 and M1133, the items in both of those skills must have the same parent items and child items. Every item that is in a parent skill of skill M1289 must be a parent item of every item in skill M1133. The same is true for items that are in a parent skill of M1133. The matrix shown in Table 1.3 shows the updated parent cells in green. Since M1104 is now a parent of the merged skill, items 1, 2, 5, and 6 are parents of items 7 and 8 now. In addition since skill M1289 is merged into the same skill as M1133, the items in skill M1289 are no longer parents of the items in skill M1133.

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	1	1	0	0	1	1	0	0	0
2	0	0	1	1	0	0	1	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	0	0	0	1
5	0	0	1	1	0	0	1	1	0	0	0
6	0	0	1	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	0	0	1	1	0
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0

Table 1.3: Merge Example: Item x Item Matrix Parents merged

The next step in the merge is to merge the children of the merged skills. This is similar to merging the parents of the merged skills. All items in children of M-1289 and M-1133 must be children of items of all items in both M-1289 and M-1133. The pink cells in Table 1.4 show the changed after merging the children. Items 3, 4, 7, and 8 now all have the same rows.

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	1	1	0	0	1	1	0	0	0
2	0	0	1	1	0	0	1	1	0	0	0
3	0	0	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	0	0	0	1	1	1
5	0	0	1	1	0	0	1	1	0	0	0
6	0	0	1	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	1	1	1
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0

Table 1.4. Merge Example: Item x Item Matrix Children merged

The final matrix shown below shows that all items merged skill M1289XM1133 have the same row entries. All items in the merged skill have the same children, which was a combination of the children from each skill. All items in the merged skill also have the same parents, which was a combination of the parent skills from each skill. Parent items are highlighted in blue, and child items are highlighted in red.

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	1	1	0	0	1	1	0	0	0
2	0	0	1	1	0	0	1	1	0	0	0
3	0	0	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	0	0	0	1	1	1
5	0	0	1	1	0	0	1	1	0	0	0
6	0	0	1	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	1	1	1
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0

Table 1.5: Merge Example: Item x Item Matrix After Merge

Evaluation Procedure

To evaluate the models we used per student per item cross validation with 5 student folds and 3 item folds. Our student and item folds were chosen randomly for our evaluation; however each item fold consisted of the same random partition of items. Typically cross-validation is done by testing on one fold and training on the others. In this experiment we did the opposite by training on one fold and testing on the other folds. We did this to see if we could make good predictions given a limited amount of student responses, which will likely occur in reality. An example of how our cross-validation works is shown in Figure 1.5. The item folds are colored with

background colors of green, red, and blue to indicate which item fold the data belongs to. Similarly the student numbers are also colored to indicate which student fold a student belongs to. The data used to train fold 1 is has a black border around the cells as an example of what one training fold will look like on a subset of the data.

Student	Item-1	Item-2	Item-3	Item-4	Item-5	Item-6	Item-7	Item-8	Item-9
S1	0	0	1	0	0	1	1	1	1
S2	0	0	0	1	0	1	1	1	1
S3	0	0	1	1	1	0	0	0	1
S4	0	1	1	0	0	1	0	1	0
S5	1	0	0	1	0	1	0	0	1
S6	1	0	0	0	0	0	0	0	1
S7	1	0	1	1	0	1	1	1	1
S8	0	0	0	0	0	1	1	1	0
S9	0	0	1	0	1	0	0	0	1
S10	0	0	1	1	1	1	1	1	1
S11	0	0	0	0	0	1	0	0	1
S12	1	0	1	0	0	1	0	1	0
S13	0	1	1	0	1	1	1	1	1
S14	0	0	1	0	0	1	0	0	1
S15	0	1	0	1	0	1	1	1	0

Figure 1.5: Cross-Validation Example. This figure shows the student and item folds color-coded and shows what a single student + item training fold will look like with a black border around the cells containing the training data.

The evaluation metrics we used were accuracy, Root Mean Square Error (RMSE), and Area Under the Curve (AUC). Since the data is binary, where '0' represents an incorrect answer and '1' represents a correct answer, accuracy is calculated by the number of correct predictions (where the model predicts the same value as the actual data) divided by the total number of predictions.

RMSE (Root Mean Squared Error) is calculated by squaring the differences of each actual value and predicted value then finding the average value of the differences. Taking the square root of the average will give the RMSE value for the model. A lower RMSE means the model is performing with a higher accuracy.

AUC (Area Under the Curve) is a measure of the accuracy of the model under consideration. AUC values range from 0 to 1 with values closer to 1 indicating a higher level of accuracy in the models predictions. Some authors (Lobo et al. 2008) have a number of reservations about the use

of AUC as the sole measure to determine the level of accuracy of a model; however, this is not a subject for discussion in this work. It is one of the reasons the authors decided to include RMSE in the determination of best model. A higher AUC means a better accuracy.

The RSME and AUC metrics were in agreement for which models were the best, where accuracy had selected different models. To check the reliability of the differences between the models, we conducted paired-student t-tests of the predictions between the models.

Experiment 1: Iterative Search

Figure 1.6 shows a graph of the results from the iterative search and Table 1.6 shows a table of the results. The search started at iteration 0, which was the initial skill map consisting of 15 skills before any merges were applied to it. The search ended at iteration 14, which is a graph consisting of just one skill with all the items attached to that one skill. The best models from each iteration are shown in Table 1.6. We recorded AUC, RMSE, accuracy, AIC, and BIC metrics, although we only used RMSE to choose the best models at each iteration and to guide our search. Ultimately, we chose RMSE as the deciding metric. The other metrics were typically in agreement with RMSE when comparing models. As the number of iterations increases, the number of skills decrease as skills are merged. Figure 1.6 shows that the initial graph of 16 skills can be decreased to a graph consisting of 11 skills for the best performance.

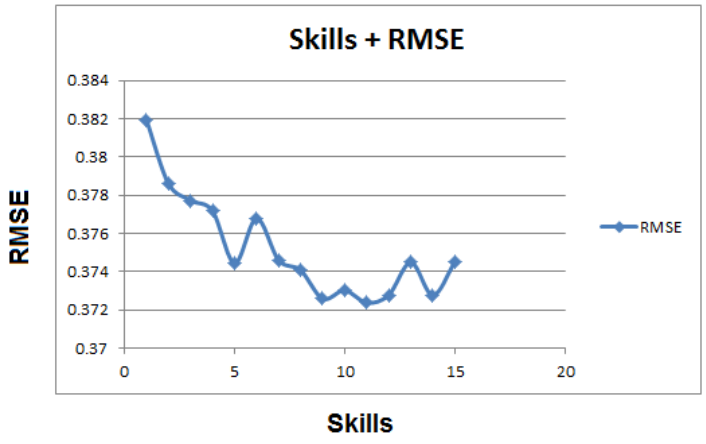


Figure 1.6: Performance for each number of skills. Each merge operation reduces the number of skill by 1. After iteration 1, there are a total of 15 skills. After iteration 15, there is only one skill.

Iteration Number	Skills	AUC	RMSE	Accuracy	AIC	BIC
0	15	0.81843	0.37451	0.79813	539988.22937	540884.62767
1	14	0.82075	0.37277	0.80087	533428.45840	534306.56286
2	13	0.8192	0.37449	0.79769	538538.26208	539398.07270
3	12	0.82086	0.37279	0.80127	536138.56542	536943.49450
4	11	0.82137	0.37238	0.80185	533921.04780	534671.09536
5	10	0.82185	0.37303	0.79951	536019.94810	536733.40797
6	9	0.82126	0.37259	0.80193	535269.73527	535946.60746
7	8	0.81906	0.37408	0.79854	539371.24687	539993.23753
8	7	0.81767	0.37461	0.79787	538287.05902	538872.46199
9	6	0.81526	0.37679	0.79341	547561.78928	548128.89840
10	5	0.81912	0.37445	0.79758	539138.56159	539678.22996
11	4	0.81557	0.37718	0.79317	547274.26056	547795.63508
12	3	0.81471	0.37773	0.79038	549777.55035	550280.63103
13	2	0.81241	0.37861	0.78983	552058.20873	552542.99557
14	1	0.80576	0.38194	0.78756	558689.91568	559156.40867

Table 1.6. Accuracy results after each merge operation of the best model for each number of skills.

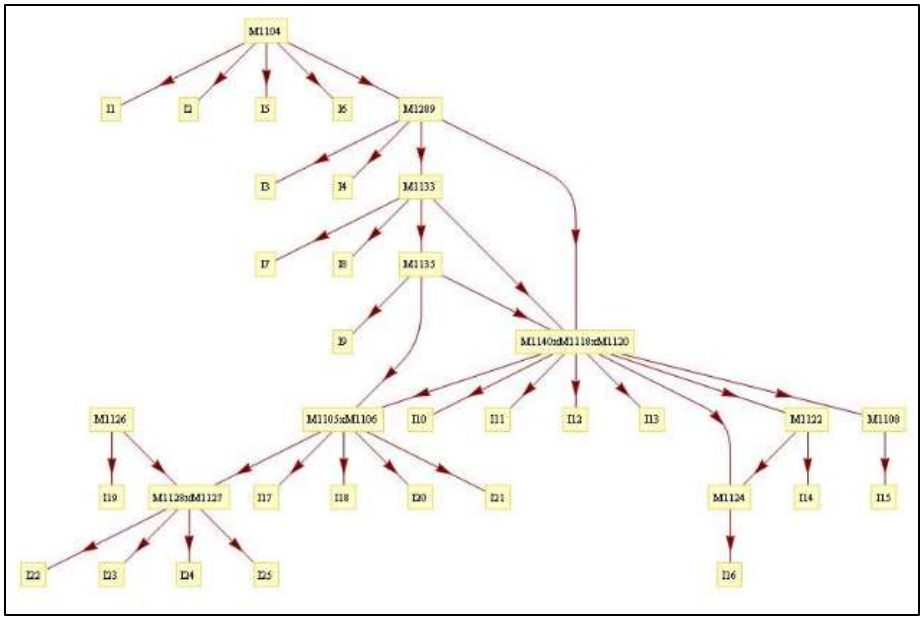


Figure 1.7: The best performing skill graph. This skill graph contains a total of 11 skills as a result of four merge operations.

The results in Figure 1.6 show that the best RMSE obtained was from the 11-skill map at iteration 4 with an RMSE of 0.372. This is slightly better than the original skill map with RMSE of 0.375. The 11-skill map shown in Figure 1.7 has a small but significant improvement ($p < 0.01$) from the original skill map. We calculated our significance by using a 2-tailed paired t-test of the errors for each prediction. The effect size was negligible (0.01). The graph also shows that models consisting of between 9 and 12 skills have similar RMSE values and are alternative choices for a best model depending on the level of skill granularity desired. Those models are also significantly better than the original model. A significance table is shown in Table 1.7 for skill graph consisting between 15-8 skills inclusive.

Skill model	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
15	N/A	5.16E-79	7.74E-12	4.81E-22	6.22E-68	7.37E-59	1.59E-30	0.23
14	5.16E-79	N/A	1.60E-44	2.64E-23	0.05	1.76E-22	1.31E-09	2.84E-51
13	7.74E-12	1.60E-44	N/A	5.29E-09	5.36E-39	4.34E-21	6.88E-15	0.05
12	4.81E-22	2.64E-23	5.29E-09	N/A	1.14E-33	0.77	0	6.10E-25
11	6.22E-68	0.05	5.36E-39	1.14E-33	N/A	2.96E-13	1.35E-06	1.32E-56
10	7.37E-59	1.76E-22	4.34E-21	0.77	2.96E-13	N/A	0.02	1.04E-15
9	1.59E-30	1.31E-09	6.88E-15	0	1.35E-06	0.02	N/A	1.80E-47
8	0.237	2.84E-51	0.05	6.10E-25	1.32E-56	1.04E-15	1.80E-47	N/A

Table 1.7: P-values (Skill models 15-8) truncated to 2 decimal places from the paired T-Test of the absolute errors from the models

In addition to looking at which model predicted the best, we also looked at which skills were being merged throughout our iterative search to see if we could find any general trends. A list of the merges is shown below. The individual skills are represented as their original numbering and a merged skill is represented by the number of each skill concatenated with an 'x'. The numbering is in topological order, meaning that the skill highest up on the skill graph will be listed first for a merged skill. The first merge is between skill 1128 and 1127. Since skill 1128 is a parent of skill 1127, it is listed first in the combined skill name 1128x1127.

Merge 1: 1128x1127

Merge 2: 1140x1118

Merge 3: 1140x1118x1120

Merge 4: 1105x1106

Merge 5: 1122x1124

Merge 6: 1140x1118x1120x1105x1106

Merge 7: 1135x1140x1118x1120x1105x1106

Merge 8: 1289x1233

Merge 9: 1135x1140x1118x1120x1105x1106x1108

Merge 10: 1126x1128x1127

Merge 11: 1135x1140x1118x1120x1105x1106x1108x1122x1124
Merge 12: 1104x1289x1233
Merge 13: 1135x1140x1118x1120x1105x1106x1108x1122x1124x1126x1128x1127
Merge 14:
1104x1289x1233x1135x1140x1118x1120x1105x1106x1108x1122x1124x1126x1128x1127

One observation is that the skills that are chosen to be merged in the first few iterations tend to be near the bottom of the skill graph. This suggests that the skills near the top of the skill graph are really separate skills compared to the skills near the bottom of the skill graph, which is the same observation made in experiment 2. Since the skills near the bottom are merged first based on best RMSE, those skills are better predicted with one skill parameter. Therefore those skills are not really that distinct since they are better modeled with one skill parameter compared to skills near the top of the skill graph.

The last skills to be merged are more likely to be distinct skills from the other skills. Our last merge was the merge between the skill group of 1104x1289x1233 and the rest of the skills in the skill graph. Since this was the last merge chosen resulting in the worst RMSE, it is likely that the group of skills 1104x1289x1233 is a separate group from the rest of the skills.

We believe the structure of the initial skill graph also has influence on the distinct skill groups. Since our merge operation only merges adjacent skills, it takes several merges for a skill at the top of the graph to merge with a skill at the bottom of the graph. A separation will naturally occur between the different levels of the graph. This implies that the original skill graph would need to be somewhat correct in terms of network topology. If non-adjacent skills were in fact the same or similar skills, there would be no easy way for our iterative algorithm to merge them, because our merge operation can only merge adjacent skills. Three distinct skill groups can be seen after iteration 12 in a 3-skill graph shown in Figure 1.8. These skills show the influence of the original network topology. The skills tend to group by their locations in the original network. The topmost skill group consists of the skills at the top of the graph, the middle group of skills consists of the skills in the middle of the graph, and the bottommost group of skills consists of the skills at the bottom of the original skill graph.

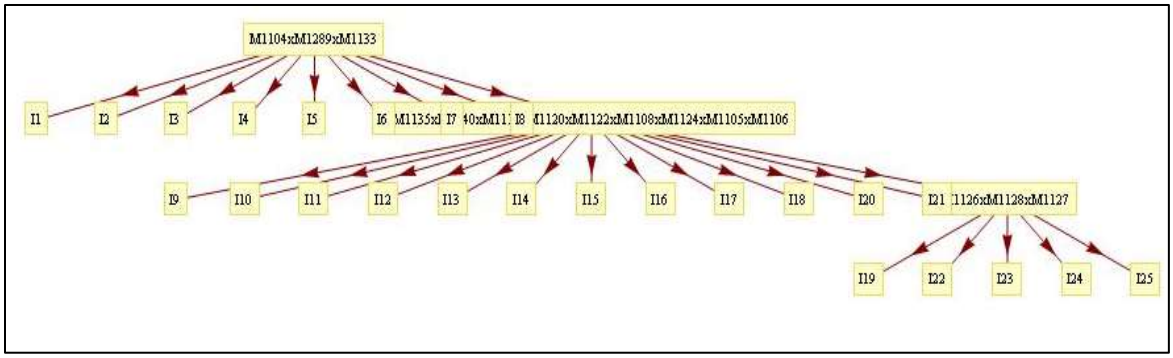


Figure 1.8: Skill Graph after Merge 12 (3 skills)

An additional observation is that some of the skills tend to merge by pairing up with one and only one adjacent skill before RMSE starts to decline. Before merge 5, the merges are all pairwise with the exception of merge 3. After merge 5, the skills tend to keep merging into the same skill. The graph generated after merge 4 corresponds to the best skill graph. This suggests that the adjacent skills tend to be similar skills. It also suggests that skills M-1140 and M-1120 are similar although they are not adjacent. This is a stronger relationship for several reasons. Firstly, the merges that culminated in the merger of M-1140, M-1118, and M-1120 all took place before the best skill graph was reached. This indicates that those three skills give better predictive performance when represented as one skill. Secondly, this was the first 3-skill group to be merged and the only 3-skill group in the best model before RMSE declines. Lastly the three skills took two iterations of the search algorithm to merge together because skills M-1140 and M-1120 were not adjacent skills. Despite the initial graph topology our search decided to merge these three skills. The combination of all these factors give stronger reasoning that the three skills, M-1140, M-1118, and M-1120 are not really distinct skills.

Experiment 2: Stability Experiment

In the previous experiment, every model was evaluated once and only once, which lead to the question of whether or not our results were stable. Our model evaluation used the Expectation Maximization (EM) algorithm, which is known to be affected by the starting value. For our original experiment we chose our starting points for EM randomly and only evaluated each model once. The authors, in earlier research, found that the starting point of the EM algorithm could make a difference in the converged value. In general the EM algorithm does converge to

the correct value, but there are cases where it can converge to incorrect values or to the "opposite" value. Considering the range to be between 0-1, if the actual true value of a parameter was 0.3, EM could converge to (1 - 0.3) = 0.7 instead if the initial starting point was too far from the true value.

Our question was: if we were to run the iterative search experiment several times would we end up with the same results using different starting values for EM. Since it takes several hours just to evaluate a single model, running the entire search consisting of over 100 models to evaluate would take too long. The purpose of this experiment was to evaluate just the first iteration of the search ten times to see if the results converged to a single best graph.

For the first iteration of the algorithm there were sixteen possible merges that could happen. For each of these possible merges we evaluated the resulting model ten times. The evaluation used was the same evaluation as the iterative search experiment for which we tested stability. For each of the ten runs we set the random seed in MatLab to correspond to the run number. This gave us a different set of random numbers for each run of the 16 possible merges, where each merge got the same random seed within a run. Manually setting the random seed also meant our results for the stability experiment would be reproducible

Results and Analysis

After evaluating all sixteen models from the first iteration ten times we kept a count of how many times a model was the best model and how many times a model was in the top three best models. RMSE was used to choose the best models since it was used to determine the best model in the iterative search experiment. The results are shown in Figure 1.9.

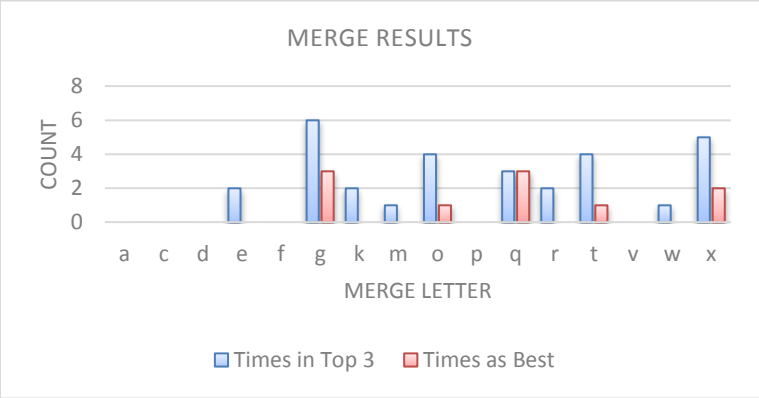


Figure 1.9: The results of stability experiment for the counts of the best models.

Merge ‘g’ was in the top 3 the most times (6) and was also the best model the most times (3). Merge ‘x’ and merge ‘q’ also did well. Merge ‘x’ was in the top 3, five times and was the best model two times. Merge ‘q’ was in the top 3, three times and was the best model three times. Merges ‘t’ and ‘o’ also did well. The general observation was that there was separation between good and bad merges but the best merge was not stable and did not converge.

We compared the graphs to our original iterative search experiment. In the original iterative search, the first two skills that were merged were skills M-1128 and M-1127, corresponding to merge ‘x’ in our stability experiment. The second two skills that were merged in the iterative search experiment were skills M-1140 and M-1118, corresponding to merge ‘g’ in the stability experiment. Both merges ‘x’ and ‘g’ were the best two graphs in the stability experiment. Although merge ‘g’ did slightly better in the stability experiment, the order in which the merges took place did not matter. The best model in the iterative search took place after four merges, which included merges ‘x’ and ‘g’. Although we could not run the stability experiment ten times for all possible merges and merge paths, we believe that it has a decent chance to converge to the same best model, which occurred after the fourth merge in the iterative search.

Discussion

When analyzing each merge, we considered the skills or concepts described by the affected skills as well as the test items associated to those skills. The descriptions below discuss the three groups of skills merged in experiment 1 and shown in the best model skill map (Figure1.7). The two additional pairs of skills merged in experiment 2 are also discussed. In each case, the merges point to commonalities in the skills themselves or among the test items used to assess different skills.

Merge ‘x’ affected skills M-1127 and M-1128. These skills represent “the abilities to represent inequalities from real world contexts” and “explain inequalities from real-world contexts”, respectively. The test items associated with these skills required students to read problems and identify inequality statements that matched the problems. In this case the test items did not distinguish between two unique skills, i.e., representing a problem or explaining a problem, as was suggested by the two skills.

Merges ‘g’ and ‘k’ affected skills M-1118, M-1140 and M-1120. These skills represent the abilities to “locate integers on a number line”, “represent integers on a number line”, and “order integers from least to greatest”, respectively. The test items associated with these skills required students to select lists of correctly ordered integers or identify the correct number line graph of a particular integer. In this case the test items did not adequately distinguish between locating and representing integers on a number line (i.e., M-1118 and M-1120) because all of the items were multiple-choice, and none provided students the opportunity to construct their own number line representations of integers. The inclusion of ordering integers from least to greatest (i.e., M-1140) with the other two skills is possibly due to the fact that using a number line is inherently, cognitively connected to ordering numbers from least to greatest.

Merge ‘t’ affected skills M-1105 and M-1106. These skills represent the abilities to “use positive and negative numbers in real-world contexts” and “relate the meaning of zero to positive and negative numbers in real-world contexts”, respectively. The test items associated with these skills required students to interpret problems involving integers and choose integer answers or verbal statements about integers. Two of the four test items included references to zero either as freezing point or sea level. In this case the items were designed to distinguish between the two skills, i.e., using integers and relating integers to zero. However, the relationship between zero and positive or negative numbers is so critical for understanding integers, that it is likely one cannot compare integers without considering their values in relation to zero.

Merge ‘q’ affected skills M-1120 and M-1108. These skills represent the abilities to “represent integers on a number line” and “recognize opposite numbers on a number line”, respectively. The test items associated with these skills required students to identify the correct number line graph of a particular integer or the opposite of a given integer. In this case, the two skills are inherently connected by the very definition of an integer as the opposite of a whole number. Consequently, it is likely that once students understand the definitions of integers and opposites and can use a number line, the act of graphing an integer is the same as graphing an opposite.

Merge ‘o’ affected skills M-1122 and M-1124. These skills represent the abilities to “recognize integer coordinate pairs” and “graph integer coordinate pairs”, respectively. The test items associated with these skills required students to identify the graph of a given integer ordered pair or to select the description of how to graph a given ordered pair on a coordinate plane. In this case, the items did not clearly distinguish between the two skills because the items associated

with recognizing integer coordinate pairs included graphs. Furthermore, the skills themselves are difficult to distinguish in a practical sense because when students learn to graph integer ordered pairs, they routinely associate the numerical representation (i.e., the ordered pair) with its graphical representation (i.e., the point graphed in the coordinate plane).

An additional observation is that some of the skills tended to merge by pairing up with one and only one adjacent skill before RMSE started to decline. Before merge 't', the merges were all pairwise with the exception of merge 'm'. After merge 't', the skills tended to keep merging into the same skill. The best skill map was generated after merge 'r', suggesting that adjacent skills tended to be similar skills and skills M-1140 and M-1120 were similar although they were not adjacent. This was a stronger relationship for several reasons. Firstly, the merges that culminated in the merger of M-1140, M-1118, and M-1120 all took place before the best skill map was reached. This indicated that those three skills give better predictive performance when represented as one skill. Secondly, this was the first and only 3-skill group to be merged in the best model before RMSE declines. Lastly the three skills took two iterations of the search algorithm to merge together because skills M-1140 and M-1120 were not adjacent skills. Despite the initial graph topology, our search decided to merge these three skills. The combination of all these factors provided strong reasoning that the three skills M-1140, M-1118, and M-1120 were not really distinct skills.

Synthetic Data Experiment

Our goal in this experiment is to see if it is possible to recover the true skill graph by using synthetic data where we know what the true skill graph is. We want to see if the result from our algorithm on the real learning map provided by the Kansas team could potentially be correct. In our earlier work (Adjei, 2014) we analyzed how several different parameters such as the guess/slip rate, number of fake skills, number of students, and number of items affected our algorithms performance on synthetic data. In that work, all these parameters were tested at a small scale with an original skill graph only containing three skills. In this experiment we scaled our experiment up to be more similar to the learning map generated by the Kansas team. We randomly chose a skill graph shown in Figure 1.10. This graph consists of seven real skills and one fake skill created from skill 4. Each skill has between 1-4 items. We chose 500 students to generate item data for and ran our search ten times with different random seeds, which affect the

EM algorithm. We also fixed our starting values of EM to be in a reasonable range (Guess: 0 – 0.4, Slip: 0 – 0.2, Skills: 0.3 – 0.7), because work reported in (Baker et al. 2008) shows that degenerate starting points for EM can cause the algorithm to fail to learn back the correct values. Out of the ten runs we learned back the correct graph 40% of the time with an average RMSE of 0.33 graphs that were correctly learned back and graphs that were not correctly learned back. We looked into why our algorithm only learned back the correct graph 40% of the time. We noticed that for all six times we failed to learn back the correct graph, we learned back a graph where skills 6 and 7 were merged. We examined the original ground truth item data to see why our algorithm chose to merge skills 6 and 7. Skill 6 had items 16 and 17. Skill 7 had items 18 and 19. By taking the absolute value of the difference between all combinations of those items the reason of why those skills were merged became clear. Table 1.8 shows the absolute error between the four items. It shows that items 16 and 17 had a lower error rate with items 18 and 19, than items 18 and 19 had with each other. This is what caused those two skills to be merged.

The next question is why items in skill 6 were better correlated with items from another skill. One would think that items from the same skill should have a higher correlation. We looked at the probability that a student knows skill 7. We set our skill conditional probability tables to take on a random value between 0.3 and 0.7, on whether or not a student knows the skill given that the student knows all of the prerequisite skills. We found that the probability that a student knows skill 7 was 0.046 or 23/500. This is because skill 7 is at the bottom of the skill graph and not many students were able to get all the ancestor skills correct; thus we identified the source of our algorithms poor performance.

In the future we will require our algorithm to have enough variation in the student response data to consider something as a skill. When nearly all the students know two adjacent skills, there is no detectable difference between the skills and therefore those skills will likely be merged. An example of this is when a college student is tested on Algebra and Calculus, which are both known by college students. When nearly all the students do not know two adjacent skills there is also no detectable difference. An example would be a young child being tested on Algebra and Calculus. Clearly Algebra and Calculus are two separate skills, but in both examples they would appear as the same skill.

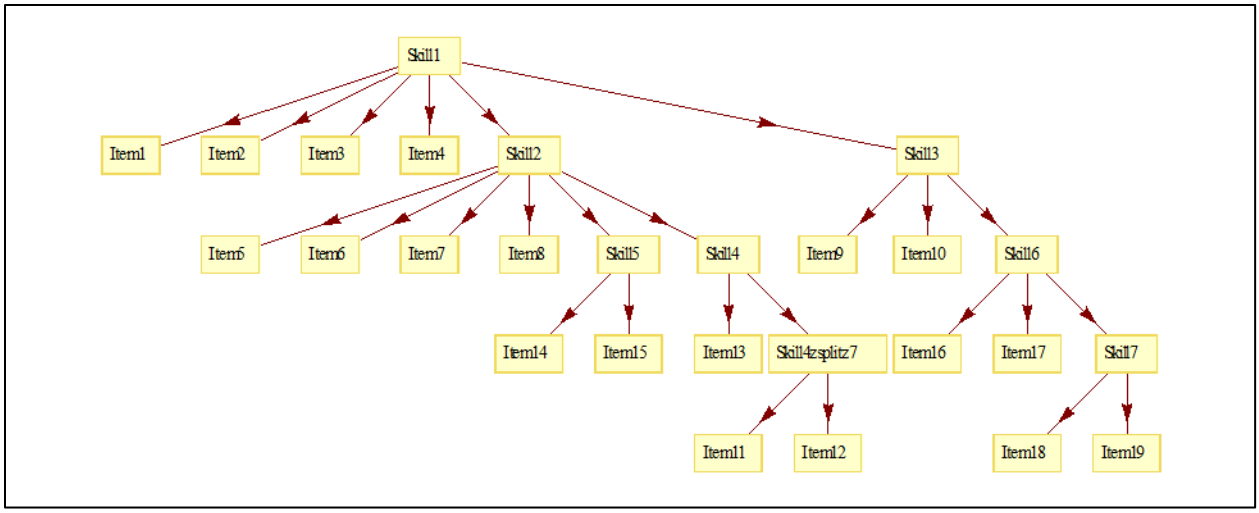


Figure 1.10: This figure shows our synthetically created skill graph with one fake skill added. The fake skill was created from taking items 11 and 12 from skill 4 and adding them to the fake skill (Skill4zsplitz7).

		Skill 6		**Skill 7**	
	Items/Items	**16**	**17**	**18**	**19**
Skill 6	**16**	0	0.16	0.232	0.232
	17	0.16	0	0.196	0.186
Skill 7	**18**	0.232	0.196	0	0.252
	19	0.232	0.186	0.252	0

Table 1.8: Absolute Difference between Items

Contributions

The main contribution of this work is the provision of a greedy algorithm that simplifies learning maps. With both real and simulated data, we showed that this simplification is possible without losing the predictive powers of the learning map. Even though this simplification could be done by hand, this algorithm will be useful in situations where the learning map is large.

This work presents a set of initial experiments in this novel area of EDM. Instead of focusing all our attention on the flat IRT model, the community needs to pay a closer attention to and explore models that deal with hierarchical relationships between knowledge components. These studies and contributions thereof can assist domain experts to produce better fitting models which should impact student learning positively.

Conclusions and Future Work

In this work we provided a search algorithm to reduce the complexity of a given learning map, while improving its fit to real student data. Since merging skills increased accuracy, these results suggest that the original skill map was too fine-grained (given the number of questions per skill and the number of students who took the test.). In some cases the test items did not adequately distinguish between the skills that were merged; hence such skills were merged. The results of algorithms like this can help the content experts who are creating skill maps and test items to either reconsider thinking of two skills as separate, or prompt them to write different test items to better distinguish between students that have mastered one of the skills but not the other skill. In this work, the team that created the learning map expected item 11 was a prerequisite for items 12 and 13, but our stability results suggested that of all the arcs, this arc was the least supported by the data (see Figure 1.1, arc "g"). In fact, due to this work, we asked an unbiased teacher who did know what our mapping was, to create a hierarchy between items 11, 12 and 13. Surprisingly, she suggested that 12 and 13 were prerequisites to item 11, suggesting that the arc should point in the exact opposite direction. This may indicate that our method may be helpful in using the data to suggest places in the skill graph that need more attention and refinement.

We can relate this work to our other work. Heffernan's ASSISTments project is a project that is attempting to track and improve students' knowledge across middle school mathematics. About a decade ago we had a learning map with over 300 skills but we now have reduced that complexity to 147 skills. Curriculum designers will correctly be thinking about the subtle ways in which problems are different from one another, which cause them to want to add skills to the skill maps to make more subtle distinctions between questions. However, if you also want to use the hierarchy to track knowledge, having more skills creates complexity, as few questions for each skill make fitting quantitative models harder.

All of the work we have done only involved a very small number of questions per skill. This naturally would cause us to think that many merges would be necessary, but if we had a large number of questions, and added all those students' responses to that large number of questions, we could probably justify more complicated models.

In our experiments we examined the effects of merging skills on an existing learning map. There are many other ways we could have used the existing map to create alternatives. For instance, Cen, Koedinger and Junker have explored ways of splitting skills or adding new skills, but all of

those make more complicated models (Koedinger & Junker, 2006). What was not examined were the split, and add operations. Possible future work could examine those operations to see if a better model can be obtained with them. Additionally, we can run more experiments with synthetic data to see what percentage of correct and incorrect responses are required for our algorithm to learn back the correct graph.

Chapter 2: Refining Learning Maps with Data Fitting Techniques: What Factors Matter?

Abstract

Cognitive models/Learning maps (skill graphs) have been identified to be possible to improve using some data mining techniques. However the factors that effect this improvements/refining process are not so clear. In earlier work we presented a method for improving these cognitive models. The purpose of this work is to present the factors to consider when using our initial algorithm to refine learning maps. We present a simulation study that shows how important each of the factors is for this refinement process.

Introduction

Learning maps have been used as a tool to depict the set of skills in a cognitive domain and the relationship between these skills. A number of studies have been conducted to find and represent the relationships between the skills (Embretson, 1998; Mislevy et al., 2002; Tatsuoka, 1983; Tatsuoka, 1995). Tatsuoka introduced the Rule-space method for identifying skills/knowledge components in a given cognitive domain whereas (Leighton, 2004) present another approach called the Attribute Hierarchy Method (AHM). The rule-space method (RSM) does not present the relationship of the skills/knowledge components as a hierarchy. However AHM, which is a variation of Tatsuoka's RSM, considers the hierarchical relationship between the components (Tatsuoka, 1995). Gierl (Gierl et al., 2007) used the AHM approach to make inferences of students' cognitive assessment. None of these approaches dealt with methods for improving the item response models developed using the methods proposed. The Learning Factors Analysis (LFA) method by (Koedinger & Junker, 2006) was introduced to deal with this problem. In that paper three different operations for improving the predictive abilities of learning maps or cognitive models are introduced. In (Adjei, Selent, et al., 2014) an attempt was made to solve this problem by presenting the results of a number of experiments that showed that learning maps can be refined using just one (the merge operation) of the three possible of the LFA method. It was shown that there were significant improvements in RMSE for the best model chosen, starting off with a pre-defined learning map.

We realize that, to generalize the method for refining learning maps, there are a number of questions that still need to be answered. These include: "What are the factors that can determine when a model can be best refined?" and "Do the number of skills, the number of items per skills, the number of levels in the skill hierarchy and the number of data points have any effect in determining the best refined model?" Whilst the LFA methods use a set of factors to determine whether to merge, add or split skills to generate better models from an existing one, all the factors used are based on expert knowledge and are independent of data. In order to answer the above questions, we present a number of simulation experiments.

Problem Statement

The LFA model uses three operations (splits, merges and adds) to refine knowledge components. In each of the operations, learning factors were included in the model refinement process. These factors did not include the number of skills in the model, the levels in the hierarchy of skills in the model, the number of items per skill and the guess and slip parameter values for the items. We hypothesize that these factors are important in generating an optimal model from a given learning map (pre-requisite skill hierarchy). Hence we set out in this work to present a series of experiments that help in determining the impact of the above mentioned factors in refining a given learning map or cognitive model.

Methodology

To be able to answer the research questions, we started off with a 3-skill graph. We inserted a fake skill at different locations of the graph and run our evaluation code to determine when the original skill-graph is learned back and what factors determine when this occurs. We examine the following factors and determine which of these factors have the most impact on using the greedy algorithm presented in the earlier work to refine a given model: guess and slip parameter values, the number of levels in the skill graph hierarchy and the number of data points (i.e. students and items). For each randomly chosen skill graph we generate a set of simulated data, one each for the number of student and item pairs used. We then evaluate the models using Expectation Maximization to determine the factors that have the most impact. The section presents the random graph generation, Bayesian network creation, fake skill creation and the evaluation code.

Random Skill Hierarchy Generation

To generate a skill graph randomly we start by choosing a random skill hierarchy. Our algorithm to generate the skill hierarchy takes a range of skills and a graph depth as input parameters. The output of the algorithm is a valid skill hierarchy where the number of vertices is within the skill range and the number of levels is within the depth range. We order our vertices from 1 to N and use the constraint that a vertex cannot have a directed edge pointing to a smaller numbered vertex. We also enforce the constraint that a vertex cannot have any self-edges.

To generate a random graph we choose a random number within the range of possible graphs. We then convert this number to binary form and add the correct number of leading zero's (we know the number of skills from the random number chosen). Then we simply insert the bits of the binary number into the varying spots of the matrix form of the graph in order.

The result is a directed acyclic graph with no self-edges. It will not necessarily be connected. The final step is to check if the graph is connected. If the graph is connected, we keep it; otherwise we discard it and repeat the generation process. This method allows us to instantly generate valid graphs. An example is shown in Table 2.1 and Figure 2.1 for a graph with three skills.

Vertex / Vertex	**1**	**2**	**3**
1	**X (0)**	**1**	**0**
2	**Y (0)**	**X (0)**	**1**
3	**Y (0)**	**Y (0)**	**X (0)**

Table 2.1: Example matrix generated by the random number 5. A 'Y' represents that this cell is ignored because it must be a zero since a vertex cannot have directed edges pointing to vertices with larger numbers. An 'X' represents that this cell is ignored because it must be a zero since a vertex cannot have self-edges.

Figure 2.1: Example Graph Generated

Bayesian Network Creation

The Bayesian network used for the analysis was generated from the skill graph selected from the previous step. To generate the items for the skills an item range is specified. A random number of items are chosen within the item range for each skill. In our experiments we restricted our range to be a single value so all skills will have an equal number of items. We set our Bayesian network up like knowledge tracing, where every skill has one or more items and every item has a guess and slip node (Corbett & Anderson. 1994). An item must belong to exactly one skill. The skill nodes are latent nodes since we cannot observe whether or not a student knows the skill. Each item node is an observable node, which is a '1' if the student answered the item correctly and a '0' if the student did not answer the item correctly. Both the guess and slip nodes are also latent nodes representing whether or not the student guessed or slipped on the item. A student is considered to have guessed when the student answered correctly but did not know the skill. A student is considered to have slipped when the student answered incorrectly but knew the skill. Using the previous skill graph example we added the item, guess, and slip nodes to the graph. The final step to create the Bayesian network is to create the conditional probability tables (CPT) for the nodes. For our experiments we defined each skill node as AND nodes. This means that a student can only know a post-requisite skill if the student knows all of the prerequisite skills. Therefore if a student does not know one of the prerequisite skills then the student cannot know the post-requisite skill. If the student does know all the prerequisite skills (or there are no prerequisite skills), we pick a random probability that the student will know the post-requisite skill between 0.3 – 0.7. Our guess and slip parameters have varying probabilities since that was one of the parameters we experimented with. All the item nodes have a deterministic (0% chance or 100% chance of correctness) CPT based off of the skill, guess, and slip nodes (which or not deterministic).

Fake Skill Creation

We exported our Bayesian network to Matlab and used Kevin Murphy's Bayes Net Toolkit to generate the ground truth data. Once the ground truth data was generated, we randomly generated "fake" skills from the original graph. A fake skill is generated by first randomly choosing a real skill. Once a real skill is chosen, a random number of items are chosen from the real skill. These items are then detached from the real skill and attached to the fake skill. The

fake skill is then randomly chosen to be either a parent or a child of the real skill. Figure 2.2 shows the creation of a fake skill. The left graph shows the original graph consisting of one skill (Skill_1), which has two items associated with it (Item 1 and Item2). Each item has a corresponding latent guess and slip nodes to represent the probability of guessing and slipping for that item. In Figure 2.2, Item 2 on the left graph is highlighted indicating that Item 2 was chosen to be associated with a fake skill. The graph on the right shows the creation of the fake skill (FakeSkill_Split1), which is a child of the original skill (Skill_1). The graph on the right now shows Item 2 associated with the fake skill instead of the real skill which it truly belongs to. This is an example of how fake skills were created for our simulated experiments.

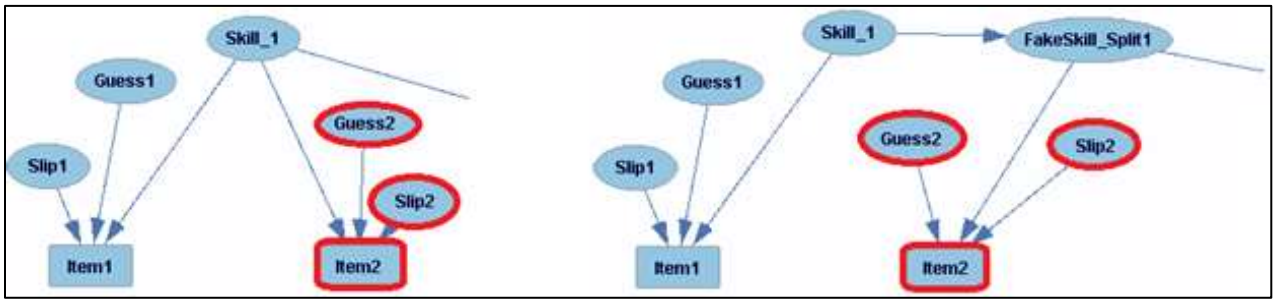


Figure 2.2: Creation of Fake Skill. The left skill graph shows the original skill graph before the creation of the fake skill. The skill graph on the right shows the skill graph after the creation of the fake skill. The fake skill was created from Skill_1 where Item 2 was removed from Skill_1 and attached to the fake skill.

Evaluation

In order to evaluate our Bayesian Network we used a similar process as done in (Adjei, Selent, et al., 2014). We use Expectation Maximization (EM) to learn parameters and fit our model. To evaluate our model we used per student per item cross validation with 5 student folds and 3 item folds. Our student and item folds were chosen randomly for our evaluation. In (Adjei, Selent, et al., 2014) the item folds were chosen randomly but kept the same for each student. The only difference between the evaluation in and this experiment is that each student is assigned a different set of random item folds instead of all students having the same set of random item folds.

Experiment 1

In this first experiment, we started with a set of 3-skill graphs. For each of the graphs, we insert a fake skill. We define a fake skill as one that is broken off of an existing skill. The fake skill has a random number of items chosen from the original skill and the fake skill is either a pre-requisite or post-requisite of the original skill. If the fake skill is a pre-requisite of the original skill, all the previous pre-requisites of the original skill become the pre-requisites of the new fake skill and the original skill becomes the post-requisite of the fake skill. The whole idea is to figure out if this fake skill will be easily identified and merged with the skill from which it was created from. This is to validate our merge operations and to determine what factors influence the determination of a better skill-model /skill map than the original.

We analyzed the results of the experiment and looked at how the number of students, number of items, guess/slip values, and the number of fake skills impacted RMSE of our predictions and the percent of correct graphs learned back. Figure 2.3 shows the relationship between the probability a student guesses/slipped and the RMSE as well as the percent of the correct skill graph being learned back. We paired guess and slip values to lower the number of variables in our experiment. Our guess/slip pairings are as follows {(0, 0), (0.1, 0.08), (0.3, 0.16), (0.5, 0.25)}. It shows that the higher chance the student has to guess the answer the less accurate and harder it is to learn back the true original graph. The percent of graphs learned back with a guess/slip probability of 0 is significantly better than the percent of graphs learned back with a guess probability of .5 ($p < .001$). A realistic guess probability is around 0.14 calculated in (Pardos & Heffernan, 2010). At this point the percentage of graphs learned is somewhere between 0.25 and 0.33. These are not great percentages to learn back a correct graph under realistic guess and slip values. Not much can be done to lower the guess probability on typical questions middle school math students would see. However more student data can be used to increase model performance.

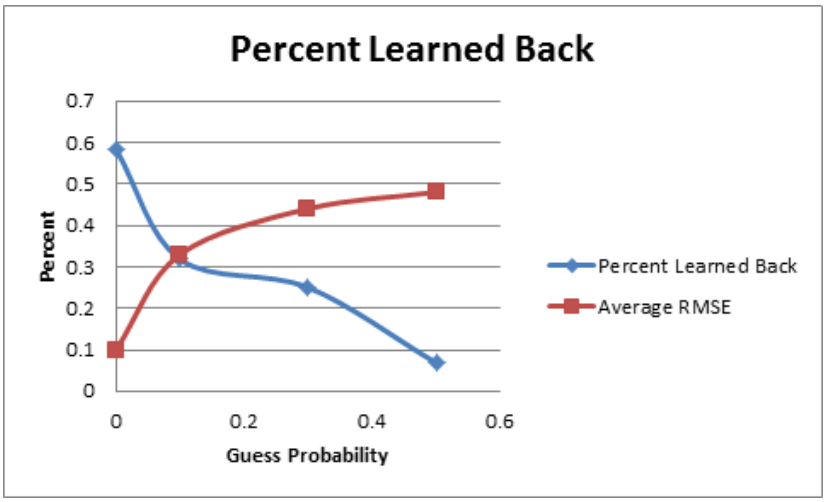


Figure 2.3: Effect of guess/slip on learning back the original graph.

The guess/slip probability is the biggest factor that affects model accuracy followed by the number of students. Table 2.2 shows how both the guess/slip probability and the number of students affects the percentage of correct graphs learned back and average RMSE. A cell is broken up into two columns where the first column in the cell is the percentage of correct graphs learned back and the second column in the cell in the average RMSE value.

	Students							
Guess	**50**		**100**		**150**		**200**	
	P_{LB}	**RMSE**	**P_{LB}**	**RMSE**	**P_{LB}**	**RMSE**	**P_{LB}**	**RMSE**
0	0.33	0.09	0.64	0.08	0.7	0.1	0.67	0.02
0.1	0.25	0.36	0.33	0.33	0.33	0.32	0.38	0.31
0.3	0.25	0.46	0.33	0.44	0.08	0.44	0.38	0.43
0.5	0.08	0.49	0.08	0.48	0.08	0.48	0	0.46

Table 2.2: Student/Guess Impact on Evaluation

For a guess probability of 0.3, the percentage of correct graphs (P_{LB}) increases from 25% for 50 students to 38% for 200 students (p = 0.2). This shows that under a realistic worst case guess probability, increasing the number of students can increase the percentage of correct skill graphs learned back. The number of fake skills seems to have little effect on RMSE, however a large effect on learning back the correct graph. With more than one fake skill the percentage of correct graphs drops significantly from 0.6 for 1 fake skill to 0 for 3 fake skills ($p < .002$) for guess values of 0.1. This can be seen for the three points with an x-axis value of 0.1 in Figure 2.4. We excluded the number of items from our analysis since there were no strong trends.

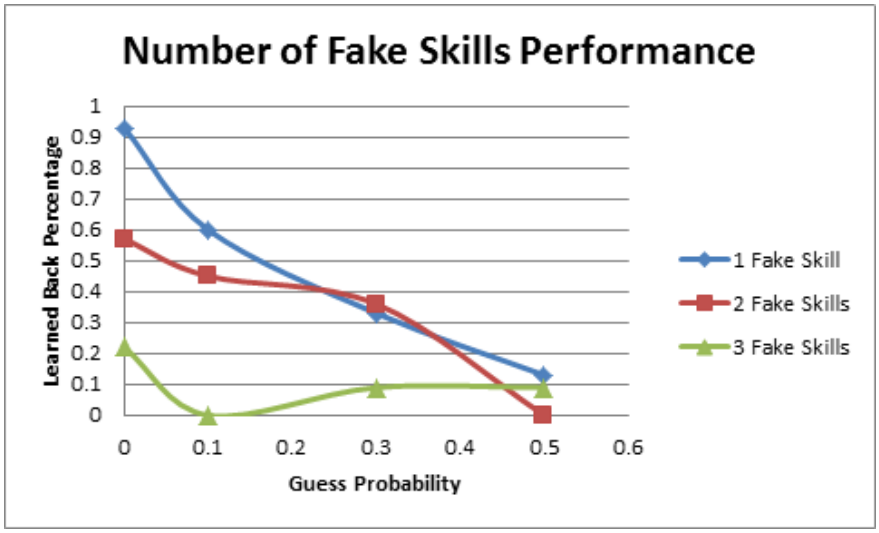


Figure 2.4: Effect of Number of Fake Skills on model improvements

Experiment 2

In experiment 1 multiple randomly chosen graphs were used as the ground truth. In this experiment we chose to try each possible 3-skill graph to see if the graph structure had an effect on whether or not the correct skill graph was learned back. The methodology was the same as experiment 1 except instead of randomly choosing graphs we ran each of the four graphs for each possible number of students and items per skill. Figure 2.5 shows all four possible 3-skill graphs. After determining that the major factor impacting performance were guess/slip values, a reasonable pair of values were chosen for the guess and slip values (guess=0.1 and slip=0.08). Additionally we fixed the number of fake skills to one in order to reduce the variability of the factors.

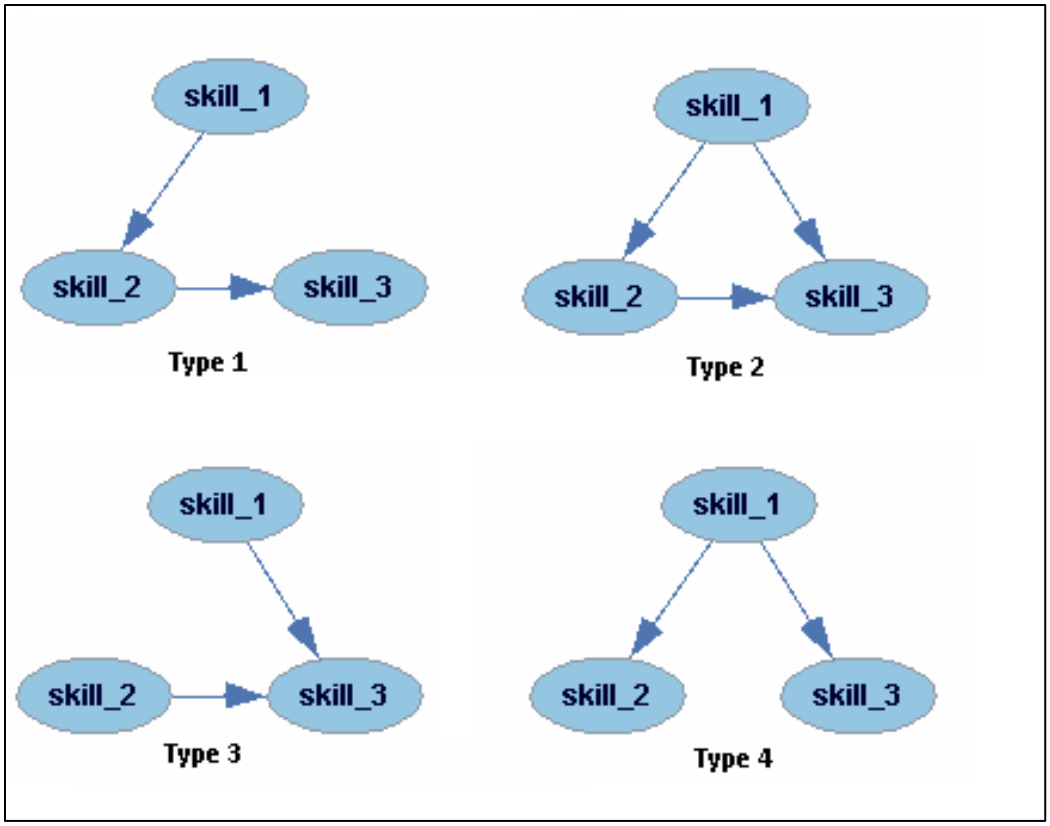


Figure 2.5: Different Graph types for experiment 2

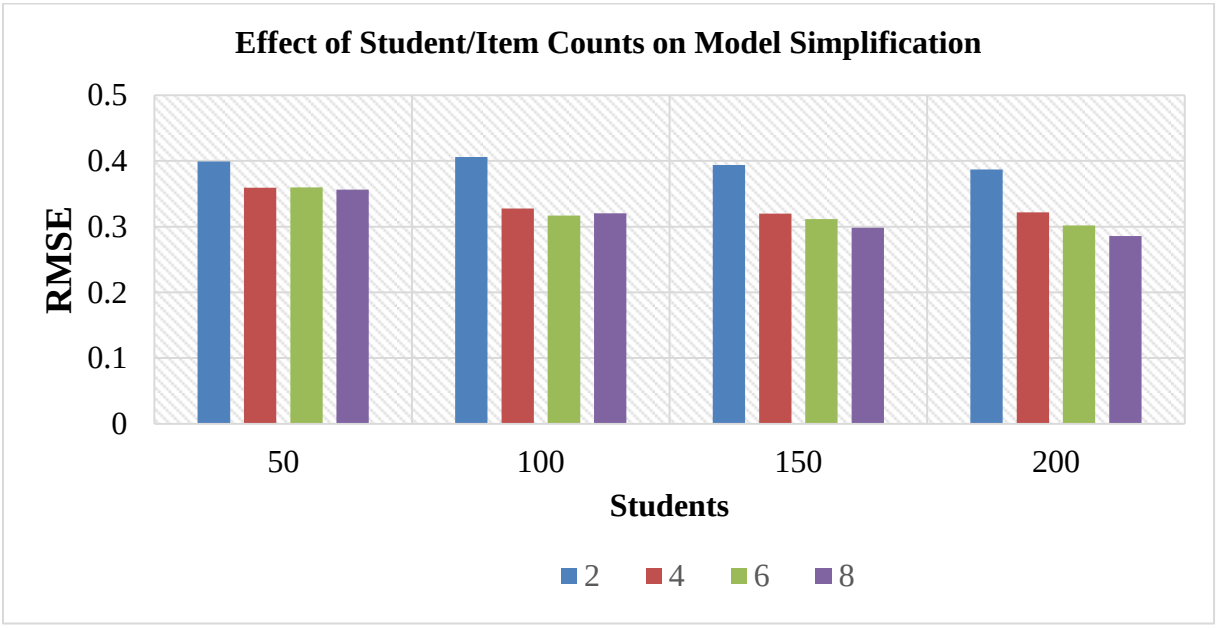


Figure 2.6: Effect of students/items on the model simplification.

The general observation from this experiment is clear from Figure 2.6 above. As the number of data points increases, the level of accuracy in recovering the original graph increases. This is in spite of the fact that the location of the fake skill was not fixed. Moreover, for any given number of students, an increase in the number of items results in a slight decrease in RMSE and hence better chance of learning back the original graph. This experiment shows that the data points (i.e. student and item numbers) have an impact on improving on the determination of the best model from a given model.

Experiment 3

In this experiment we fixed all variables except for the number of students and the number of items per skill. We wanted to see how stable our search was and how well it performed for a small example with reasonable parameter values. We fixed guess at 0.10 and slip at 0.08 with three skills and one fake skill. For the fake skill we took the first half of items from the original skill. We ran our algorithm for 50, 100, 150, and 200 students for 2 and 8 items per skill. For each pair of parameters we ran the experiment 10 times with different random seeds and took an average of the number times the correct graph was learned back. Figure 2.7 shows the results of this experiment. We found that the results are very stable for graphs that had two items per skill. The results were less stable for graphs with eight items per skill although the percent of graphs learned back was much batter. The graphs that had two items per skill were learned back correctly 8% of the time, where graphs with eight items per skill were learned back correctly 43% of the time, which is a significant improvement (n=40, p<.001).

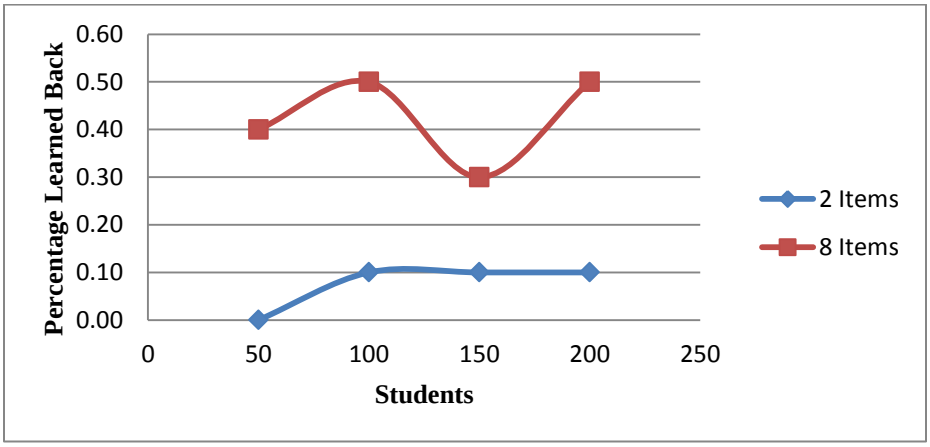


Figure 2.7: Percent of graphs learned back for student ranges 50-200 and 2+8 items per skill.

Experiment 4

We ran experiment 4 to confirm that the number of students has an impact on the recoverability of the original graph, fixing all other parameters at reasonable values and varying the number of students. For this experiment, guess and slip values were set at 0.1 and 0.08 respectively. We used graph type 4 (Figure. 2.6), set the number of items to 4 and fake skills at 1, varying the location of the fake skill. The student numbers were varied from 10 to 100. For each student number, the evaluation was run 10 times. The results, in Figure 2.8, show that as we intuitively assumed, the number of students has a huge impact on the algorithm's ability to learn back the true graph. The results show that as the number of students increases the probability of a skill graph being learned back increases whiles at the same time the RMSE reduces. These results, we found, are significant with p-values below 0.01. This finding confirms that student numbers is an important factor that needs to be considered when refining learning maps.

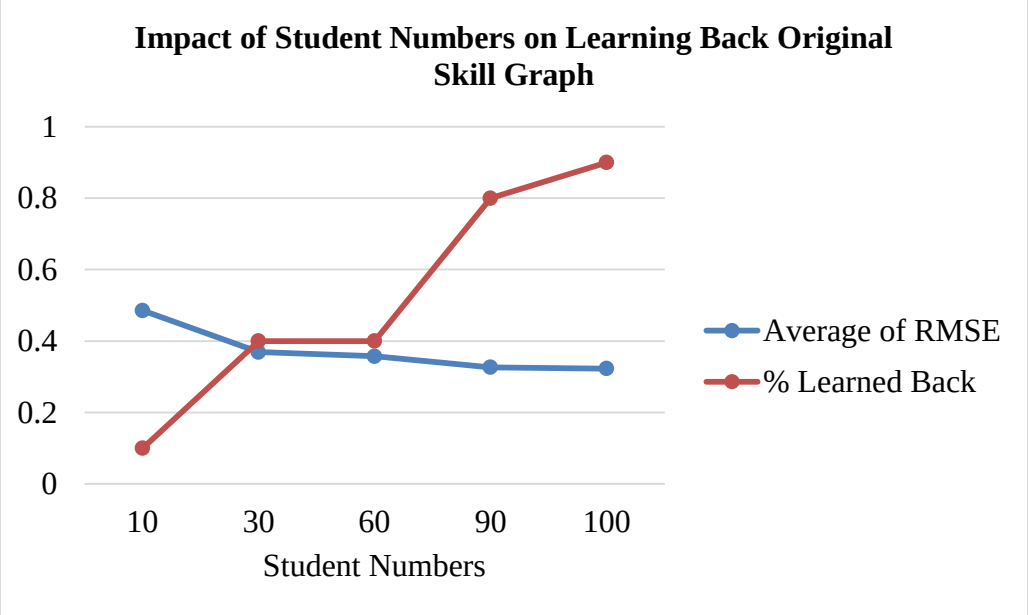


Figure 2.8: Impact of Student Numbers

Conclusion

Many learning maps/cognitive models are built from expert knowledge. With the production of lots of educational data on student performance, it has become imperative to find data centered methods of improving upon these expert-designed learning maps. In our earlier studies we designed and presented an algorithm for simplifying/improving the predictive accuracy of these models. In this work we have presented a number of factors that influence the data centered model improvement process we initially published. We have shown with our simulation studies that the guess/slip values, number of items per skill, the number of students and the number of fake skills in the graph affect the simplification of the skill models. We also explored many parameters to see how much data is needed to recover the true learning maps. For future work we plan to continue to evaluate our algorithm on larger examples to see how well our algorithm can scale up and test it on well-known real data sets.

Chapter 3: Machine Learning Student Incorrect Processes

Abstract

The goal of this work is to look at previous forms of help in tutoring systems and try to improve them. More specifically "buggy" or "incorrect" messages are improved and compared against hints. A machine-learning algorithm is developed to determine exactly what the students' mistakes were on a given problem given the student's answer and the input to the problem. The results of the machine-learning algorithm are used to generate buggy messages to display to the students. This algorithm is evaluated from data from the ASSISTments system and is compared to hints in the ASSISTment system. This chapter introduces the machine learning algorithm used to derive student incorrect processes. The following chapters discuss several randomized controlled experiments run based on the buggy messages generated from the machine-learned incorrect processes.

Introduction

ASSISTments is a freely available online learning platform mostly used for mathematics between grades 4-12 in the United States with a large number of students located in or near Massachusetts. ASSISTments combines assisting students with instruction and assessing student abilities at the same time (Razzaq et al, 2005). Students use ASSISTments on classwork and homework, which may be done with or without the use of a paper copy. Instant feedback is typically provided to the students upon answering problems, so that students will immediately know if they have answered the problem correctly. Students are not allowed to skip any problems and must answer correctly to continue on to the next problem. If a student does not answer the problem correctly on their first try, the problem is marked as incorrect. Almost all problems in ASSISTments offer the student the option to click on a hint button, which either offers additional help to the student or just gives the student the answer to the problem so that he or she may continue.

In many tutoring systems students are often given the option to ask for help. This can be seen in many forms such as scaffold questions (Razzaq & Heffernan, 2010), webpages (Gong et al, 2012), videos (Ostrow & Heffernan, 2014) and hints among various tutoring systems. In the

ASSISTments tutoring system hints are the most common form of help and are what will be analyzed. A hint in the ASSISTments system appears as a button to the lower right of the text field where a student enters their answer. When a student clicks on the hint button they immediately get the question marked wrong and are given a "hint". Most of the time there is only one hint that just contains the answer to the question. The purpose of these hints is so the student can continue with the assignment, because a student cannot skip questions in ASSISTments. ASSISTments data was examined for the 2012-2013 school year. Some basic statistics are shown below on the data in Table 3.1.

	Counts	**Percentage**
Total Problems	30,143	100 %
Problems answered correctly	23,283	77 %
Problems answered incorrectly	3,173	11 %
Problems started but not answered	3,687	12 %
Problems where hints were used	1,879	6 %

Table 3.1. Summary statistics of ASSISTments data for the 2012-2013 school year

Out of the 30,143 problems a surprising number of students answered the problems correctly. We believe a selection bias occurs in the ASSISTments system, since many students who struggle do not do their homework. Therefore there is no data in the system for those students who struggle and the data has a bias in favor of better students and a higher percentage correct. The group of students who answered incorrectly is the group we will be focusing on since they are the students that can benefit from a tutor. We also think this group includes more students than the data suggests since it is unlikely a student will always answer all the questions correctly on every assignment. Whenever a student answers a question incorrectly is where the tutor can help. On the same dataset hints were analyzed to see whether or not they were beneficial to the students who use them.

Table 3.1 shows that roughly 60% of students who answer incorrectly use a hint sometime during the problem, although only 6 % of the time hints are used at all. Figure 3.1 shows the number of attempts it takes a student to complete problems for those students who do not answer correctly on the first try and use hints. Figure 3.2 shows the same information for students who do not use hints.

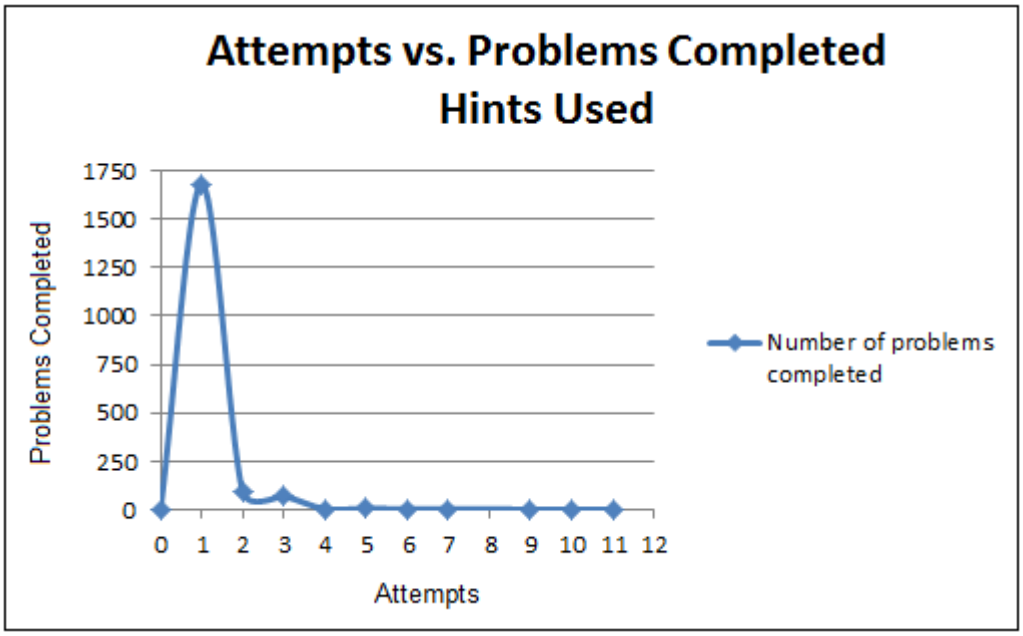


Figure 3.1: The number of attempts it takes to complete problems given that the problem was answered incorrectly and a hint was used at least once.

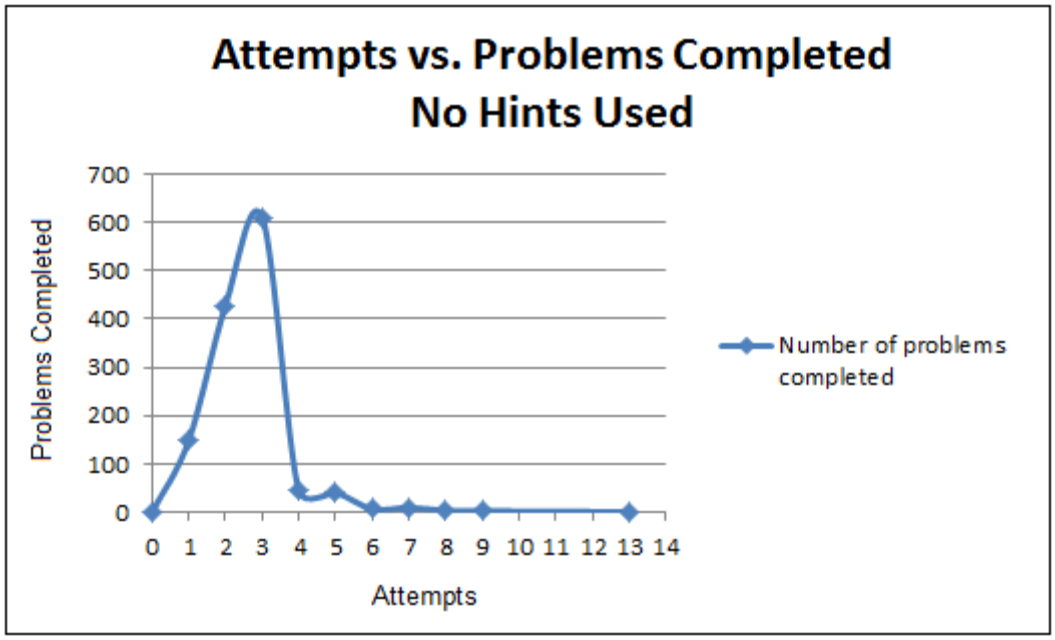


Figure 3.2: The number of attempts it takes to complete problems given that the problem was answered incorrectly and no hints were used.

Some points to notice are that only 6% of the time hints are used and the other 94% of the time students attempt the question without any hints. This number is lower than anticipated. One plausible explanation is that the use of hints is discouraged because the student gets the question wrong if they ask for a hint. What is worse is that nearly all the hint help (92 %) is just one hint containing the answer to the question, which is no different than looking in the back of the text book for the answers. This explains the graph on Attempts vs. Problems Completed. When hints are used students almost always finish the problem with one attempt because they were given the answer. When hints are not used students take within 1-3 attempts to solve the problem with the peak number at three attempts.

An important statistic is how well the students do on the next question. This is important because the graphs in Figures 3.1 and 3.2 can be deceptive. The graph in Figure 3.1 indicates that students who use hints almost always answer correctly in one attempt. The reason is because most problems only have a single hint which contains the answer. Therefore it may look like hints are helpful, but they may not be. A good question to ask is, "How well do those students do on the next problem after asking for a hint?". Data on students who received help via hints and data on students who did not receive any help was examined. A subset of the dataset was used where the previous question was answered incorrectly and the next question was also in the same assignment. The total number of data points in this dataset was 3,075. Table 3.2 summarizes this data.

	Used hints on previous problem	Did not use hints on previous problem
Problem count	1,821	1,255
Percent correct on next problem	32 %	84 %
Percent of hints used on next problem	64 %	5 %

Table 3.2: Next problem correctness and hint usage broken up by whether or not the student used a hint on the previous problem.

Students who did not use a hint and answered incorrectly got the next question correct 84% of the time compared to only 32% of the students who used hints. Also students who used hints on the previous question used hints 64% of the time on the next question where students who did not use hints on the previous question only used hints 5% of the time on the next question. From this data analysis it appears that not only are the hints not helping, but that the hints in ASSISTments are negatively affecting the students, although it is possible that the students asking for hints knew less than the students that did not ask for hints.

These statistics back up my own observations from my work with 7th grade middle school math classes. I typically notice that the students will try to get the questions correct. However as soon as they get the question wrong they will just use the hint to get the answer and move on. This makes some sense since they already got the problem wrong and their main goal is to complete the assignment as fast as possible. Learning is not the priority for most middle school students. I even had to hand-fight a student for mouse control to force him to redo a problem that he tried to skip via abusing the hint button. There are also several papers that address this issue (Beck et al., 2008; Baker et al., 2008).

To address several of the issues with hints, a different form of help needs to be revamped. In ASSISTments there is the form of help called a "buggy message". Buggy messages are when a message appears on the screen after a student enters a wrong answer. Messages will only appear if the answer given by the student matches an answer entered by the teacher for a predicted wrong answer. Buggy messages are supported but rarely used in ASSISTments because it takes too much time to predict and enter all the incorrect messages for all possible common wrong answers. The program and algorithms in this work solve this problem and can identify most wrong answers and exactly how a student derived them by taking advantage of the existing infrastructure and data in ASSISTments.

"Buggy" messages were first introduced by (Brown & Burton, 1978) and the early 1980's with their series of "BUGGY" programs. Work with these programs was done in (Brown & Burton, 1978; Brown & VanLehn, 1980; Burton, 1982). The main goal was to build a database of incorrect answers and the process of how those incorrect answers were generated. Unfortunately most of the work was done manually and for a very small domain of simple multi-digit addition and subtraction problems. The ASSERT c++ tutoring system developed in (Baffes, 1994) and (Baffes & Mooney, 1996) also walks students through their mistakes as they are made. They use

customized multiple choice questions in order to be able to identify what the student mistakes are. A machine learning algorithm developed in (Jarvis et al., 2004) also starts with a set of pre-made rules for their algorithm to work with. Their reason was that their program could not handle the search space without help and it also only worked with only correct instances of data.

The common theme throughout these papers is that all the intelligence to find a student's incorrect process is manually done in one way or another. In ACM (Langley et al., 1984) and in this work the program automatically machine learns the incorrect processes that lead to the incorrect answers. ACM uses a best-first search algorithm to generate a discrimination network and search through a user defined search space with a set of operators. When there are multiple paths to the student's answer the simplest path is chosen. This was acknowledged as an unsatisfactory method and is a much bigger problem than they claim. Since the numbers that student's work with are not very distinct, the problem of multiple solution paths occurs quite frequently and most of the time the simplest solution is not correct.

A good literature summary of many of the various algorithms to find procedural bugs can be found in (Sison & Masamichi, 1998). It briefly summarizes the history of the research in the area from the beginning up to the publication date. This chapter mentions some but not all of the research and machine-learning algorithms associated with buggy procedural rules. For a slightly dated, but more comprehensive description see (Sison & Masamichi. 1998).

In this chapter, the machine learning algorithm removes ambiguity by using the existing ASSISTment infrastructure of templates and instances, where one template generates several instance of the same problem with different numbers. This allows the algorithm to generalize derivations across multiple problems, to remove ambiguity from all the possible derivations to the incorrect answer. All of this is automated and the output is an expression of how the student solved the problem incorrectly. Buggy messages to target the specific error can then be manually generated based off of the output of the program.

Incorrect messages have several advantages over hints. Firstly, incorrect messages do not give the student the answer. This will prevent students from mindlessly typing in the answers given to them by the hints and force them to try to answer the question. Remember that the main goal of the student (from the student's perspective) is to finish the work, not to learn. Incorrect messages will force the student to learn how to do the problem correctly since that is the only

way for the student to finish the assignment. Secondly, incorrect messages provide more personalized tutoring. Hints are generic and may or may not apply to students depending on the reason why the student asked for the hint. Incorrect messages are specifically targeted to the exact incorrect answer the student typed in. This is one of the main differences from the work in (Razzaq & Heffernan, 2010) where the students were provided with help as soon as they answered a question incorrectly. The form of help they received was generic and was the same no matter what the student typed in, which may be the reason they found that students did better without automatically giving the students hints. In this chapter, the form of help (the messages) specifically target the incorrect answers entered by the student making them relevant to the student. Incorrect messages are also designed to improve student efficiency, providing immediate help as soon as the student answers incorrectly. Hints require that the students know when to use them and not abuse them. In (Aleven & Koedinger, 2000) it was shown that 72% of actions taken by the students were negative actions towards asking for or not asking for help. Lastly, the incorrect messages are very short and specific to the area the student needs help on. This was done intentionally because the attention span of a middle school student is very short. The design of the incorrect message is to provide the most possible help to the student within three seconds. Although the student may need more than three seconds of help, most of the time they are not willing to give more than three seconds of attention to text on a computer screen. (Brown & VanLehn, 1980) state that most mistakes are slight deviations from the correct procedure, where a few simple words of help are all that is required.

Incorrect messages can also be beneficial to teachers too. When a teacher is walking around the computer lab and sees a student get an answer wrong, the must first identify the error the student made in order to correct it. This process of identifying the error the student made can take less than five seconds but in several cases it can take several minutes. Long division in particular is a nasty problem for teachers to identify mistakes quickly. The incorrect message will solve this problem since it will point out the specific error immediately so the teacher does not have to waste valuable time in the error identification process.

Algorithms

The entire algorithm can be broken down into five main parts listed below.

1. Derive answer
2. Reconstruct solution path
3. Store expressions
4. Generalize expressions
5. Choose expressions

Each of these five parts will be described in detail in this chapter. Basically the algorithm starts by deriving all the incorrect answers for every problem for a given template. Deriving an incorrect answer is when a set of input symbols (numbers) and an incorrect answer are given and the program discovers a path to the incorrect answer. Each path to the answer is reconstructed and converted to a list of expressions. These expressions are stored and later cleaned of duplicate expressions. After removing duplicate expressions, all the expressions are combined into a single list which is then evaluated for all the incorrect answers. The expressions that did not generalize are removed and the most accurate expressions are chosen from the remaining list. The result is a list of expressions that generalize across multiple problems. This will show exactly how students are deriving their incorrect answers.

Derivation Algorithm ("The Deriver")

Derivation Algorithm Part 1

The algorithm used to derive an incorrect answer ("The Deriver") takes a set of symbols, a set of operations, and the incorrect answer as input. The output of the algorithm is a set of all unique paths that can be constructed from the input symbols and operators that evaluate to the given incorrect answer. Note that the search is cut off at a point and does not run forever, since there are theoretically an infinite number of ways to generate a single incorrect answer.

Through some trial and error the cutoff points were created to get the deepest search possible while not causing the program to crash due to insufficient memory. The first part of the algorithm stops when the number of unique input symbols generated is greater than 5,000 or when the number of iterations is greater than 15. The second part of the algorithm stops when the number of input symbols is greater than 10,000 or when the number of iterations is greater than 15. The input symbols come from the given problem and/or plausible numbers.

The input operators come from a list of plausible operators a student would use. Currently the operators supported are “add”, “subtract”, “multiply”, and “divide”. Users can select any subset of these operators as input for the search algorithm. It is recommended that all operators are used to machine-learn the incorrect paths, since limiting the operators and enforcing domain knowledge can cause the algorithm to be unable to find some unpredictable solutions that require the additional operators. A real example problem is shown below and an image of the derivation algorithm input setup is shown in Figure 3.3.

Example Problem

What is 2% of 60?

Input Symbols

2, 10, 60

Operators

Add, Subtract, Multiply, Divide

Incorrect Answer

12

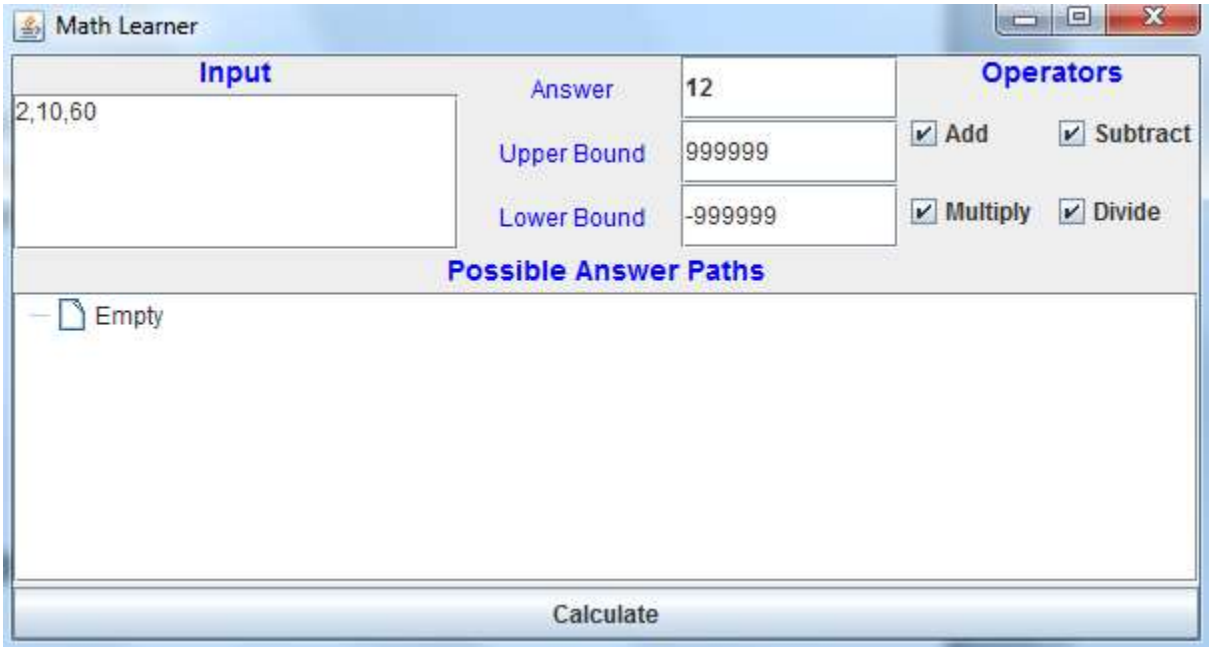


Figure 3.3: Example of input to “The Deriver”

In the example shown in Figure 3.3, the input symbols include the numbers given in the question as well as 10. The reason 10 was included is that a common error with percent problems is misplacing the decimal point. Therefore adding 10 to the input symbols will allow The Deriver to discover the incorrect answer with a more intuitive process. All the operators were included in this example. A good reason to include more operators is to be able to derive more possible incorrect answers. The problem with including more operators is that it makes the search space expand much faster and will end the search much quicker.

The algorithm starts by making a matrix of the input symbols and operators. Two dimensions of the matrix are the input symbols and the third dimension is operators. Each operation per input symbol combination is calculated for each cell in the matrix. Continuing from the previous example, Table 3.3 shows how the operations are done and the output of each operation for the first iteration.

Add

	2	10	60
2	2+2	2+10	2+60
10	10+2	10+10	10+60
60	60+2	60+10	60+60

Multiply

	2	10	60
2	2*2	2*10	2*60
10	10*2	10*10	10*60
60	60*2	60*10	60*60

	2	10	60
2	4	12	62
10	12	20	70
60	62	70	120

	2	10	60
2	4	20	120
10	20	100	600
60	120	600	3600

Subtract

	2	10	60
2	2-2	2-10	2-60
10	10-2	10-10	10-60
60	60-2	60-10	60-20

Divide

	2	10	60
2	2/2	2/10	2/60
10	10/2	10/10	10/60
60	60/2	60/10	60/60

	2	10	60
2	0	-8	-58
10	8	0	-50
60	58	50	0

	2	10	60
2	1	0.2	0.0333
10	5	1	0.1666
60	30	6	1

Table 3.3: Example three-dimensional matrix for iteration one of The Deriver

Once all the matrix values are calculated those values become new symbols which are added to the existing symbols and are used as the input symbols for the next iteration of the algorithm. For this example, there were 36 (3x3x4) symbols generated. These 36 symbols are then combined with the existing 3 symbols (2, 10, and 60) and stored into one array. Duplicate values are removed from the array to only include unique symbols in the next iteration of the algorithm. The 26 unique symbols after iteration 1 are shown below.

Unique Symbols after One Iteration

{2, 10, 60, 4, 12, 62, 20, 70, 120, 0, -8, -58, 8, -50, 58, 50, 100, 600, 3600, 1, 0.2, 0.0333, 5, 0.1666, 30, 6}

The unique symbols for every iteration of the algorithm are saved so when an incorrect answer is found, the path to it can be reconstructed. Each path to every answer cannot be stored because it requires too much memory, however once the algorithm ends there are usually less than 100 possible paths to the incorrect answer. That small subset of paths can be reconstructed and saved in memory. Once an incorrect answer is found, the iteration number, two symbols, and the operation is saved onto a stack. When the search ends, the reconstruction algorithm ("The Reconstructor") reconstructs the paths for all elements on the stack

This iterative process, of performing all operations on all input symbols, continues until The Deriver reaches one of the cutoff points. When this step is complete, the result will be a stack of "AnswerStorage" type elements. The "AnswerStorage" element contains a point type which consists of the two last input symbols that derived the correct answer. It also contains the operation used with those two points and the iteration number that those symbols and operations derived the incorrect answer.

Since the number of symbols will expand exponentially, the algorithm can only run for a few iterations. After only 3-4 iterations the search will take up both too much time and too much space. In order to compensate for this issue, another version of the algorithm is built on top of the current one. The first search will run normally until a certain number of input symbols are reached. By trial and error 5,000 symbols were found to be a good estimate of how many symbols can be used in the first algorithm before too much time and space are used. Once this number is reached the algorithm switches over to the other version of the search algorithm.

Derivation Algorithm Part 2

The other version of the algorithm is almost the same as the first with the exception that it will only use the symbols from the previous iteration on the next iteration. This is different from the original algorithm, which adds the new symbols to the list of unique symbols. All symbols generated from any iteration other than the previous one are not used and essentially forgotten. This allows the search to expand much slower and to last for more iterations at the cost of missing some of the less common possible incorrect answer derivations. Using the first algorithm is more thorough at the cost of the search space expanding too fast where this algorithm is less thorough but keeps the search space smaller. The combination of using the first search algorithm followed by the second utilizes the advantages of both algorithms.

An example of both algorithms working together is shown below. The same problem is used that was used in the first example, however to keep the matrix size displayable only the multiply operator is shown in Table 3.4.

	2	**10**	**60**
2	2*2	2*10	2*60
10	10*2	10*10	10*60
60	60*2	60*10	60*60

	2	**10**	**60**
2	4	20	120
10	20	100	600
60	120	600	3600

Table 3.4: First iteration calculations (multiply operator only)

Unique Values = {2, 4, 10, 20, 60, 120, 600, 3600}

For the sake of simplicity, in the example the next iteration of the algorithm will switch to the second version of the algorithm. In practice, the number of input symbols can reach 5,000 before this actually happens. The unique values listed above are the original input symbols plus the new symbols after the first iteration. The reason why 2, 10, and 60 are highlighted is because they will not be used in the next iteration of the algorithm. The second algorithm only uses the symbols generated from the previous iteration. The symbols 2, 10, and 60 were the original input to the algorithm and were not generated by previous iteration of The Deriver. Table 3.5 shows the second iteration matrix calculations for the Deriver.

	4	20	120	600	3600
4	4*4	4*20	4*120	4*600	4*3600
20	20*4	20*20	20*120	20*600	20*3600
120	120*4	120*20	120*120	120*600	120*3600
600	600*4	600*20	600*120	600*600	600*3600
3600	3600*4	3600*20	3600*120	3600*600	3600*3600

	4	20	120	600	3600
4	16	80	480	2400	14400
20	80	400	2400	12000	72000
120	480	2400	14400	72000	432000
600	2400	12000	72000	360000	2160000
3600	14400	72000	432000	2160000	12960000

Table 3.5: Second iteration calculations (multiply operator only)

Unique Values

{4, 16, 20, 80, 120, 400, 480, 600, 2400, 3600, 12000, 72000, 360000, 432000, 2160000, 12960000}

The non-highlighted unique values will be used in the third iteration of the algorithm and the highlighted values will not be used. Something to note is that in this particular example some of the symbols are very large numbers. After noticing this effect, both the first and second search algorithms accept a maximum and minimum value as parameters. If any of the symbols are out of the specified range then those symbols are discarded. Depending on the input symbols and operations, the incorrect answer values may be quite large or small. It is not uncommon to see answers that are several digits too big. A possible area of future work could be to determine a good range of values to minimize the search space. Currently -9,999,999 and 9,999,999 are used as default minimum and maximum values respectively, which are 1 digit less than the 8-digit school calculators.

Another observation is that using only the symbols from the previous iteration can cause the algorithm never to hit a maximum number of symbols for certain input combinations. This is because the number of unique symbols generated by each iteration of the algorithm never reaches the maximum number of symbols. In addition to a 10,000 symbol cap for the second algorithm found by trial and error, an iteration limit was also used and set at 15.

Strengths (combined algorithm)

The main strength of this algorithm is that given a set of correct input symbols and operations, it is guaranteed to find the incorrect answer. This comes with two assumptions. One assumption is that the student did not make a mistake performing the operations and merely chose an incorrect order of operations. For example if a student answered 12 for the question "What is 2% of 60?", the algorithm will be able to figure out the student performed the calculation (2/10) * 60 = 12, where the correct operations would be (2/100) * 60 = 1.2. However the algorithm may not be able to find the mistake if a student made a mistake multiplying and thought .02 * 60 = 7. The other assumption is that the algorithm has enough time and space to find the incorrect answer. Another strength of the algorithm is that is tends to find the given answer in a smaller number iterations. This is because each step of the problem can be done in parallel by pairs of input symbols. For example (7+8) + (3-2) = 16. Both 7+8 and 3-2 are done in one iteration of the algorithm. The symbols 15 and 1 will be generated and the answer 16 will be found on the second iteration.

Weaknesses (combined algorithm)

There are a few weaknesses of this algorithm. The first weakness is that it requires a lot of space to run since in the worst case the matrix is squared after each iteration of the first part of the algorithm. This means the algorithm can only run for a few iterations before both the time and space limits exceed what a computer can handle. Improving the algorithm by only remembering the input symbols of the previous iteration helped fix this weakness. Typically middle school math problems do not require many steps to complete, and therefore the incorrect answers do not consist of many steps either. Therefore the depth of the search is sufficient to find the derivation of the student's incorrect answer.

The second weakness is that mechanical errors are not considered, such as forgetting to carry a 1 when adding 3-digit numbers. The algorithm also does not catch errors such as adding a percent sign to the answer where one is not required or other input formatting issues. These errors are not math related and are not addressed by the program.

Lastly the algorithm currently only works for a small domain of problems. Some examples of supported problems are solving simple equations with one variable, order of operation, adding and subtracting integers, finding surface area and volume of shapes, 2-digit addition, and 2-dgit

subtraction problems. Problems related to fractions are not well supported unless they can be turned into a set of input symbols and operations. 3-digit addition and 3-digit subtraction cause search space problems and are not currently supported. It actually takes a lot of steps to solve a 3-digit subtraction problem when the input symbols are broken down into digits. Graph image problems as well as any imagery are also not supported. Basically the algorithm can support any problem that can be broken down into input symbols and operators where the correct and incorrect solutions can be solved in a small number of steps.

Path Reconstruction ("The Reconstructor")

The next step of the algorithm takes the stack of "Answer Storage" elements and reconstructs the path to each answer. For each element, a recursive function is called to reconstruct the path. For the final two symbols stored, the previous matrix (saved as a set of unique symbols) is used to find all symbols and operations which result in both of the two current symbols. This process will repeat recursively for all the new symbols. The recursion stops when the first iteration input symbols are reached. Using the same matrix as the previous examples, consider the answer 280, which we will assume is just found by 70 x 4. There are other ways to arrive at 280 but looking at just one is simpler. In the example only "The Deriver Part I" is used for simplicity.

	2	10	60	4	12	62	20	70	120	0
2										
10										
60										
4										
12										
62										
20										
70				280 = 70 x 4						
120										
0										

Table 3.6: Reconstruction Example: Second Iteration (multiplication operator only)

Since the answer 280 was found on the second iteration, this must mean that both 70 and 4 were derived on the first iteration matrix. Looking back to the first iteration matrix we find all places where 70 and 4 were found. The matrix in Table 3.7 shows all values of 70 and 4 in green. It shows all the input symbols that were used to get to 70 in red and all the input symbols that were used to get to 4 are in blue. Due to the reduced symbol size of the first iteration, all operators are can be shown in Table 3.7.

Add

	2	10	60
2	4	12	62
10	12	20	70
60	62	70	120

Multiply

	2	10	60
2	4	20	120
10	20	100	600
60	120	600	3600

Subtract

	2	10	60
2	0	-8	-58
10	8	0	-50
60	58	50	0

Divide

	2	10	60
2	1	.2	0.0333
10	5	1	0.1666
60	30	6	1

Table 3.7. Reconstruction Example: First iteration matrix. Reconstructing symbols 4 and 70

For iteration 1, the input symbols that were able to be combined with an operator to evaluate to 70 were 60 and 10. The input symbols that were able to evaluate to 4 was just 2. Since this is now the first iteration, the recursive algorithm ends since we have successfully gone from the resulting answer back to the original input symbols of the problem. For answers at a higher iteration, this process would repeat until the input symbols on iteration 1 are reached.

The results of the path reconstruction are stored in a tree structure. For the example "What is 2% of 60?", the output of the algorithm is shown in Figure 3.4. Each pair of symbols and operation is shown as well as the iteration number the answer was found. When a tab is expanded the sub-steps are shown to see how the input symbols for the current step were generated. Non-expandable tabs indicate that the symbol is an original input symbol. 0.2 * 60 was expanded to show a possible solution path. Starting from the inner most symbols and working to the outer most symbols is how someone can see how the student got the answer 12. First the student performed 2.0 / 10.0 to get 0.2. Next the student performed .2 * 60 to arrive at 12.

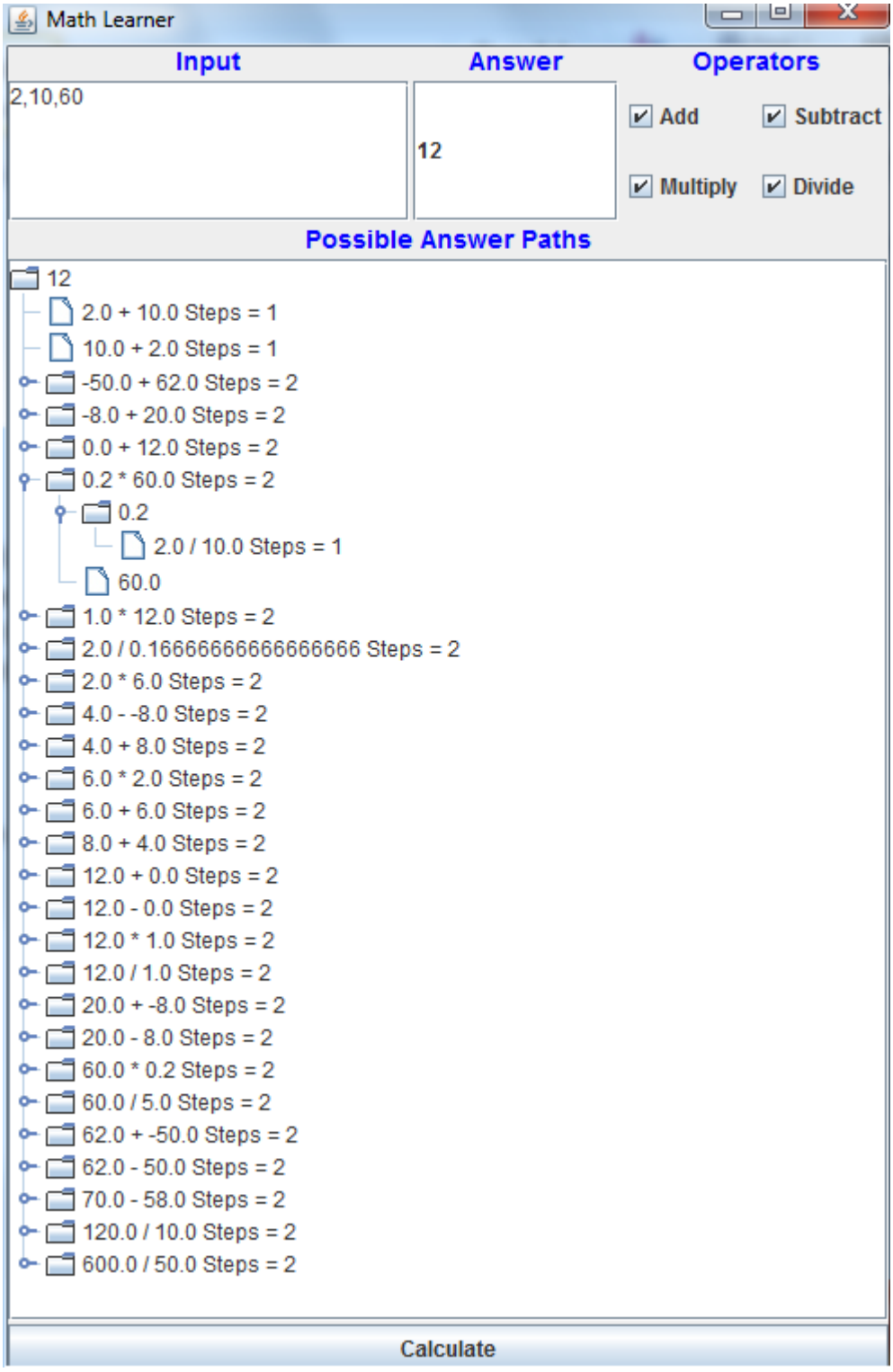


Figure 3.4: Results Derived and Reconstructed

It can clearly be seen that there are several ways to reach a single incorrect answer. However many of these possibilities are unrealistic and will not generalize to what the student was actually thinking. Therefore an algorithm to generalize solutions is needed. This generalization algorithm ("The Generalizer") takes several different problems generated from the same problem template and determines which rules apply to more than one instantiation of the template. The rules that apply to several templates are likely to be a correct interpretation of what the students did incorrectly. For example if a rule such as (Input 1 / Input 2) * Input 3 was able to correctly derive the incorrect answers for several problems then it is likely to be the correct buggy rule that those students applied. This rule would work for the following three problems generated from the same template. Think of the output shown in Figure 3.4 as a single tree of a forest. The derivation of a single incorrect answer is a tree, where the forest is made up of trees for all the incorrect answers for a given problem. A group of forests is all the derivations for all the problems within a single template.

Storing and Retrieving Output

Before the generalization algorithm ("The Generalizer") starts, the output of the derivation algorithm must be stored. This may seem simple since most people would assume the output of forests is just saved as a simple object in Java. This is somewhat true, since all the forests are saved as Java objects (an array of DefaultMutableTreeNodes type objects). However due to memory constraints only one forest can be held in memory at a time. Therefore all forests are saved to disk and only loaded into memory one at a time when needed.

Another issue is that the tree structure is a good form to hold the data output from the derivation algorithm. It is easy for users to visualize and for programmers to work with. The problem is that The Generalizer needs a list of expressions with only the input symbols and not the intermediate symbols. Figure 3.5 shows the output of The Deriver and Figure 3.6 shows what the required output form must be in to use as input to The Generalizer.

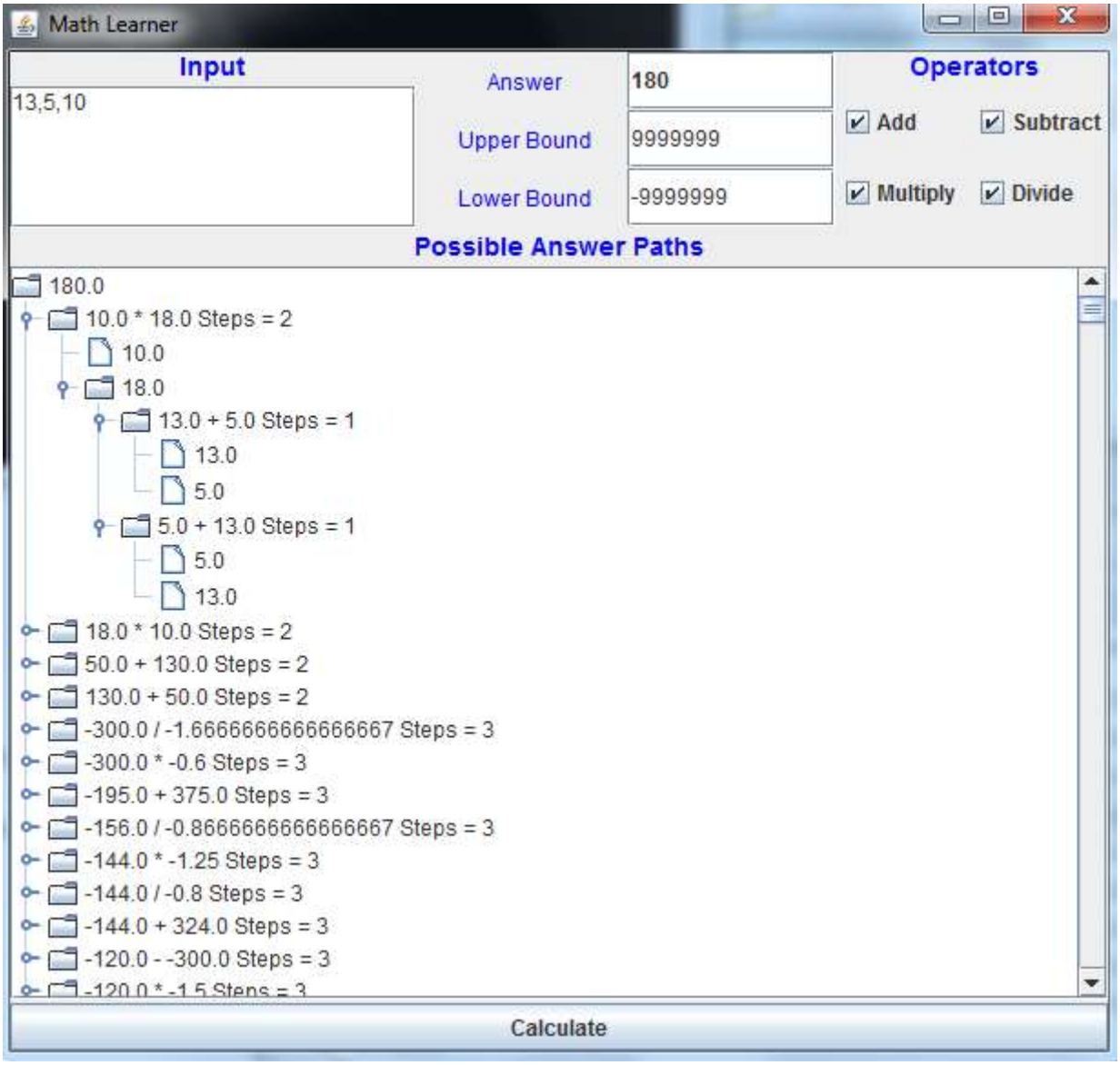


Figure 3.5: Example Output Tree Structure from the Derivation Algorithm**.** This image is from an example derivation from the problem 13 + 5 x 10 where the correct answer would be 63 and the incorrect answer entered was 180. The internal tree structure can easily be represented for the user to see; only the path for 10.0 * 18.0 was expanded.

1. 10 * (13 + 5)
2. 10 * (5 + 13)

Figure 3.6: Example Output Expression List**.** This is the lists of possible ways that 10.0 * 18.0 can be represented in terms of the original input symbols. In this simple example there are only two ways and they are actually duplicates.

One of the trickiest recursive algorithms in the program takes a branch of the tree and converts it to a list of all possible expressions only using the input symbols that could have generated the branch. Another expression from the example (50.0 + 130.0) is used to walk through the algorithm. The tree for 50.0 + 130.0 is shown in Figure 3.7.

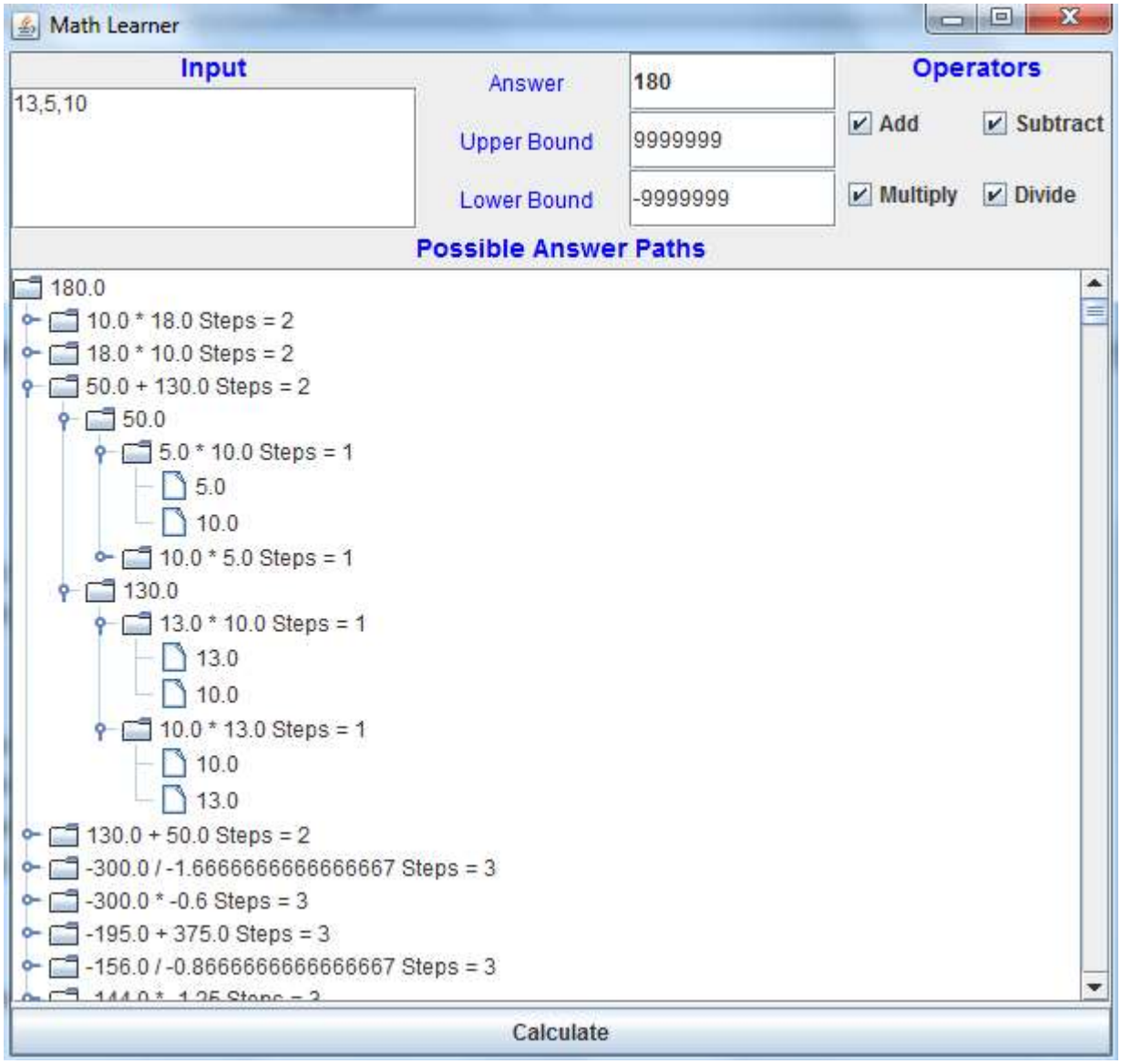


Figure 3.7: Example Output Tree Structure for 50.0 + 130.0

The recursive algorithm starts with the initial node passed to it, which is shown as 180.0 in Figure 3.7. In the tree, there are two types of nodes, single value nodes and single expression nodes. A single value node is just a single value such as 180.0 or 50.0. A single expression is 50.0 + 130.0 or 5.0 * 10.0. The algorithm always expects a node with a single value to be passed to it. The algorithm firsts checks to see if the node has any children. In this example the node 180.0 has several children, of which we will only look at the child 50.0 + 130.0. Before any further calculations are done, the initial node value of 180.0 is stored as a String. This can be considered the start of our expression list.

The next step of the algorithm looks at all children of the current node. In this case our current node is 180.0 and we will only look at the child 50.0 + 130.0. The "50.0 + 130.0" is extracted from the node as a string. An initially empty list of new expressions is created and a string substitution is made from parent value to the child value. This just means that the string "180.0" is replaced by "50.0 + 130.0" in our example. If we considered more children then "180.0" would also be replaced by those children and those new strings would also be added to the new expression list.

After this substitution, each child node of the node 50.0 + 130.0 is iterated through. In this case there are two child nodes containing the values 50.0 and 130.0. Each iteration recursively calls the expansion algorithm currently being described, which returns a list of expressions. That expression list is added to the current expression list and the values that were previously on the expression list are removed. After this is repeated for each child node, the final expression list is returned. A very simplified walkthrough is shown below. This example is not actually how the recursive calls work but is a way to describe the calls to readers. For example in the recursive depth of 1, where we recursively call the function for the value of 50.0, the result is not actually (5.0 * 10.0) + 130.0, and (10.0 * 5.0) + 130.0. It is really the result of the recursive call on those two values, which is the result of the recursive calls on those values, etc… Therefore the calls are shown in a somewhat backwards order to help the reader understand the process. For example in recursive depth 1, (5.0 * 10.0) is shown as something that needs to be broken down further, when in the actual code it already has been. The last step would be the base case were both 5.0 and 10.0 are single value leaf nodes that correspond to input symbols.

Example

Recursion Depth = 1

Current node value = 180.0
Child node value = 50.0 + 130.0

Expression list = 50.0 + 130.0

For each grandchild (50.0 and 130.0), Iterate and add expression to the list

Grandchild 50.0
(5.0 * 10.0) + 130.0 added to expression list
(10.0 * 5.0) + 130.0 added to expression list

(50.0 + 130.0) removed from expression list

Expression list =
(5.0 * 10.0) + 130.0
(10.0 * 5.0) + 130.0

Grandchild 130.0
(5.0 * 10.0) + (13.0 * 10.0) added to expression list
(5.0 * 10.0) + (10.0 * 13.0) added to expression list
(10.0 * 5.0) + (13.0 * 10.0) added to expression list
(10.0 * 5.0) + (10.0 * 13.0) added to expression list

(5.0 * 10.0) + 130.0 removed from expression list
(10.0 * 5.0) + 130.0 removed from expression list

Expression list =
(5.0 * 10.0) + (13.0 * 10.0)
(5.0 * 10.0) + (10.0 * 13.0)
(10.0 * 5.0) + (13.0 * 10.0)
(10.0 * 5.0) + (10.0 * 13.0)

Recursion Depth = 2

Current node value = 50.0
Child node value = 5.0 * 10.0

Expression list = 5.0 * 10.0

For each grandchild (5.0 and 10.0), Iterate and add expression to the list

Grandchild 5.0
    5.0 * 10.0 added to expression list

    5.0 * 10.0 removed from expression list

Expression list = 5.0 * 10.0

Grandchild 10.0
    5.0 * 10.0 added to expression list

5.0 * 10.0 removed from expression list

Expression list = 5.0 * 10.0

Recursion Depth 2 is similar for inputs 50.0 = 10.0 * 5.0, 130 = 10.0 * 13.0, and 130 = 13.0 * 10.0.

Recursion Depth = 5

Current node value = 5.0
Child node value = none

Return 5.0

Recursion depth 3 is similar for inputs 5, 10, and 13.

After running the tree structure through the recursive algorithm, the output is the complete list of all possible expressions that could generate the given incorrect answer. Before moving on it should be obvious that there are a large number of duplicate expressions in this list, where a duplicate expression is two or more expressions that will always evaluate to the same number for all possible inputs. A Simple example of duplicate expressions include (10.0 * 5.0) and (5.0 * 10.0), where the order of the numbers do not matter since (x * y) is the same as (y * x). In order to save space all duplicate expressions are deleted. The expression of the shortest length is chosen as the expression to keep, where all other longer duplicate expressions are removed. Counting sort (Cormen et al., 2001) can be used to sort the expressions by length because the expression length is typically not that long and is an integer. The value of 2000 is used as the maximum length of an expression although typically none of the expressions reach half that number. 2,000 is also just an arbitrary number greater than the maximum expected expression size and can be greatly increased if needed. Counting sort has a time complexity O (n + k) where n is the input size and k is the maximum value. The input size is the size of the list of expressions and the maximum value will be the expected maximum length defined as 2,000. The time complexity is basically O(n) since k is a constant. This is faster than the fastest possible comparison based sorting algorithms such as Mergesort or Heapsort, which have a time complexity of O(n log n). Since counting sort is not a comparison based sort and requires assumptions about the input, such as the maximum expression length being less than 2,000, it can achieve much faster performance.

In addition to using counting sort to sort the expressions by length, Mergesort (Cormen et al., 2001) (O (n log n)) is used to sort the expressions by value. Mergesort was chosen over other good sorts such as Quicksort and Heapsort because of the stability property. Order must be preserved among the expressions of the same value so they will be sorted by value then by length. Quicksort and Heapsort do not preserve this order, meaning that after running Quicksort on our list of expressions, the result will be a list sorted by value but not by length for each value. Mergesort will give the desired result of a list of expressions sorted by value then by length for each value.

A question to think about is why we are sorting by value when all the expressions are derived from the same value. The answer is that a heuristic to quickly detect duplicate expressions will substitute the original values in the expression with new values. If the expressions substituted with the new values evaluate to the same values then we assume they are duplicated expressions. This provides a quick way to detect duplicates without actually parsing each expression to compare to each other expression. In a single run of the program there could be over a billion expressions and the duplicate detections needs to run quickly. It is assumed if two expressions evaluate to the same values under two different sets of inputs then they are duplicate expressions. New input numbers are generated randomly for each old input symbol. The values can range from (-50.0, -2] and [2, 50). There is a 50% chance that the number generated will be negative. Numbers 1 and 0 are excluded since they do not work well at making distinctive expression evaluations. Zeroes' are especially problematic and are even caught and rejected by the derivation algorithm at the first step of the program. This means that the algorithm cannot machine-learn incorrect processes for an incorrect answer of zero.

Once the new values are generated, the existing expression strings are replaced with the new values. Wherever an old value is seen, the corresponding new value will replace it. This is done with basic string substitution, which is possible given the specific format of the expressions. Subtraction signs will never be confused with negative signs.

Given several strings of expressions, a basic recursive descent parser was created from a context free grammar to evaluate the expressions. The context free grammar is described below and is shown in Table 3.8. An example parse tree for an expression is shown in Figure 3.8. Since over one billion expressions may need to be parsed over the course of a run of the program some basic benchmark tests were done on the parser. For one of the tests the expression " (((-14.0 * 11.0) * 33.0) + ((11.0 / 33.0) + (-14.0 - 11.0))) " was run through the parser 10 million times. This took ~40 seconds to complete. Extrapolating this time to running the parser a billion times results in 4,000 seconds or roughly a little over an hour. This was considered an acceptable amount of time.

Symbol	Meaning	Example value 1	Example value 2
E	Expression	(19.0 - 17.0)	(((19.0 - 17.0) + (11.0 - 2.0))
d	Double value numbers	19.0	17.0
o	Operator	*	+

Table 3.8: Context free grammar for expressions

CFG

Set of rules (R) =

{

E → E o E,

E → d

}

Operators (o) = +, -, *, /

Terminal symbols (Σ) = operators (o), double value numbers (d)

Spaces are used to separate symbols (used by program only)

Parenthesis are used to separate expressions (used by program only)

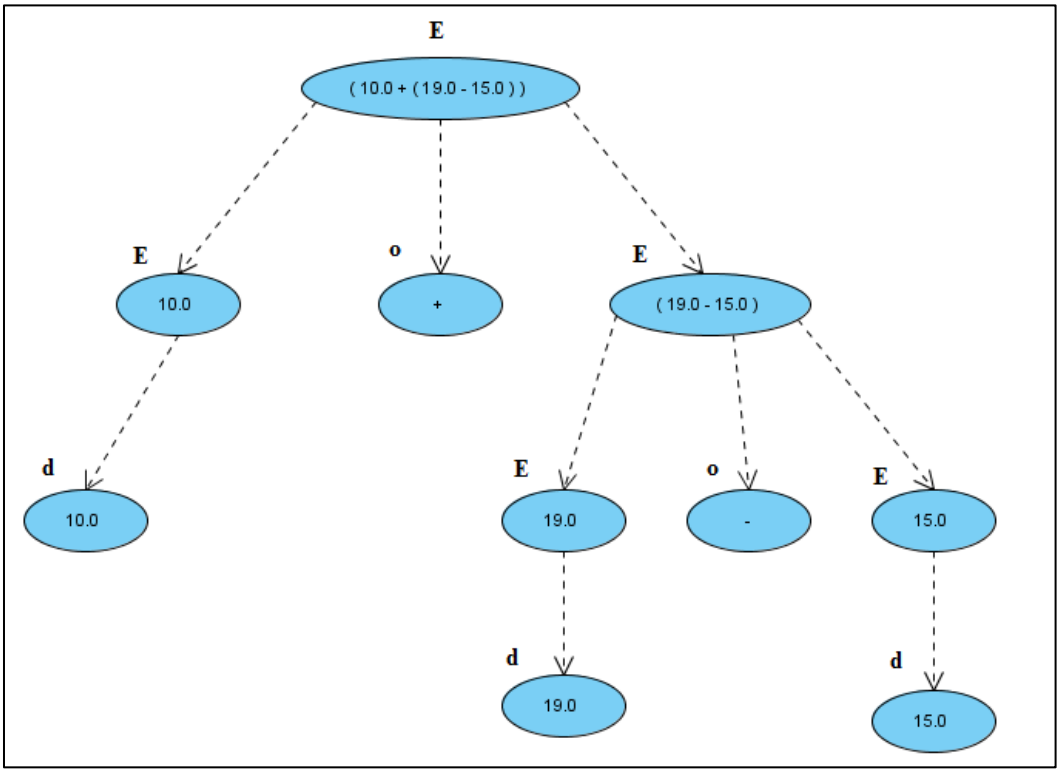


Figure 3.8: Parse Tree Example for expression " (10.0 + (19.0 - 15.0)) "

A little extra piece of information is that the sorting by value and detecting duplicates will not work correctly if the values are compared directly. The "==" operator in Java will do a direct comparison between two values, but using this caused hours of debugging pain. When expressions are evaluated, two or more of the same expression could in fact be the same but evaluate to slightly different values. The values are stores as double types and the order of operation matters especially when using division on repeating decimals. The last digit or last few digits in the decimal might differ although the expressions are really the same. To compensate for rounding errors a constant value of 0.0001 is used as acceptable error. If taking the absolute value of the difference of the two expressions results in a value < 0.0001 then the expressions are considered equal. Both the Mergesort operation and the deletion of duplicate values incorporate this logic into their respective algorithms. Values of NaN, which can be obtained by dividing by zero, are removed from the final results. A hypothetical example of the sorting is shown below in Tables 3.9 – 3.12, where the incorrect answer is 14 and the inputs are 2, 3, and 4.

Original Value	New Value
2	5
3	-2
4	7

Table 3.9: Key to map original values to new values

Original Expressions	Replaced Expression	Values
2 + (3 * 4)	5 + (-2 * 7)	-9
(2 + 2) + (2 + 2) + (2+ 2) + 2	(5 + 5) + (5 + 5) + (5+ 5) + 5	35
(2 * 2) + (3 * 3)	(5 * 5) + (-2 * -2)	100
(3 * 4) + (2 + 2) - 2	(-2 * 7) + (5 + 5) - 5	-9
(3 * 3) + (2 * 2)	(-2 * -2) + (5 * 5)	100
(3 + 3) + (3 + 3) + 2	(-2 + -2) + (-2 + -2) + 5	-3
(4 + 3) * 2	(7 + -2) * 5	25

Table 3.10: Expression list with replaced values

Expression	Value
5 + (-2 * 7)	-9
(-2 * 7) + (5 + 5) - 5	-9
(7 + -2) * 5	25
(5 * 5) + (-2 * -2)	100
(-2 * -2) + (5 * 5)	100
(-2 + -2) + (-2 + -2) + 5	-3
(5 + 5) + (5 + 5) + (5+ 5) + 5	35

Table 3.11: Expression List after Counting sort (Sorted by length)

Expression	Value
5 + (-2 * 7)	-9
(-2 * 7) + (5 + 5)- 5	-9
(-2 + -2) + (-2 + -2) + 5	-3
(7 + -2) * 5	25
(5 + 5) + (5 + 5) + (5+ 5) + 5	35
(5 * 5) + (-2 * -2)	100
(-2 * -2) + (5 * 5)	100

Table 3.12: Expression List after Mergesort (Sorted by value then by length)

After the list of expressions is sorted by value then by length, the program simply iterates through the list and deletes the duplicate expressions. The program makes the assumption that the expression of the shortest length is the best one to keep. For example in the table above there are two expressions that evaluate to -9. "5 + (-2 * 7)" and "(-2 * 7) + (5 + 5)- 5" both evaluate to -9. Therefore the assumption is that the extra "+ 5 – 5" in the longer expression is not needed or an accurate prediction of the thought process of the student and that expression is removed. In the case where the lengths of the expressions are the same the first expression is kept and all subsequent duplicate expressions are deleted. Table 3.13 shows the final output after removing the duplicate expressions.

Expression	Value
5 + (-2 * 7)	-9
(-2 + -2) + (-2 + -2) + 5	-3
(7 + -2) * 5	25
(5 + 5) + (5 + 5) + (5+ 5) + 5	35
(5 * 5) + (-2 * -2)	100

Table 3.13: The expression list after removing duplicate expressions. The expressions "(-2 * 7) + (5 + 5)- 5" and "(-2 * -2) + (5 * 5)" were removed from the original list of expressions.

Finally the results are in the correct format for The Generalizer to use. The final list is written to a file and removed from memory. This process is done for each problem until all the problems have been analyzed. Once all derivations are complete, all the possible expressions are stored in files. Each file stores a list of expressions for each incorrect answer for each problem. The next step of the machine-learning algorithm (The Generalizer) compares all the expressions in each of these files to generalize expressions across different problems.

Generalization Algorithm (The Generalizer)

How the generalization process works took careful thought to keep processing time to a minimum. What needs to be done is to find which expressions generalize across problems for every problem, for every incorrect answer, and for every derivation of those incorrect answers. There are too many possibilities to compare all the expressions to all the other expressions therefore all the expressions are evaluated instead.

Combine Expressions

First, all of the expressions are combined into a single list regardless of which problems they were generated from. Several factors make this possible in a quick amount of time. One factor is that the removal of duplicate expressions from the previous step of the algorithm greatly reduces the number of expressions. This allows all the remaining expressions to be stored in one list where the maximum size supported is 2^31 – 1 = 2,147,483,648 elements.

A second factor is the size of the number of problems, incorrect answers, and expressions are manageable. The largest number of problems for a given template is 200. In general there are no more than 200 distinct incorrect answers for a given problem. In cases where there are more than 200 distinct incorrect answers, only the first 200 most common are used. In this case typically only the first 50 answers have been entered more than once and the other answers have only been entered once. Those answers that have only been entered once have little value and throwing some of them away does not hurt the effectiveness of the program. A similar method is used for the expression list sizes. Most of the time there are no more than 200 unique expressions, however in rare cases there can be over 2000. In these cases, only the 200 shortest expressions are kept. The result of these size restrictions makes it possible to loop through each problem, each incorrect answer, and each expression at a maximum of 200 * 200 *200 = 8,000,000 operations to combine all the expressions into one list.

A third factor that makes it possible to combine the expressions and make the whole generalization process possible is the ability to convert all the expressions to a common input. As the expressions are being combined into a single list they are also being converted into generic input. An example is shown below on how the conversion is done.

In Assistments there is the ability to create templates for problems. This was developed in (Razzaq et al., 2009) to help teachers create problems for students. Templates can then be used to generate instances. Templates can contain a mix of HTML, text, and variable indicators. A variable is assigned a range of values that instances will use randomly to replace the variable. An example template is shown below. All template and instance text was cleaned of return characters and commas since those characters interfered with the formatting of the data. Return characters were replaces with [return] and commas were replaced with [comma]. This string substitution has no negative impact on the program in any way.

Template Example:

Template ID: 30833

Text:

“ Solve for %v{a} [return]   [return] %v{c1}%v{a} + %v{c2} = %v{c3} [return]   ”

Variables:

a = {a;b;c;x;y}

d = {1;-1}

b = rand(7)+2*d

c1 = rand(10)+2

c2 = rand(10)+2

c3 = c1*b*d+c2

final = (c3-c2)/c1

Answer:

%v{(c3-c2)/c1}

Instance Example:

Problem ID: 49609

Text:

“ Solve for y [return]   [return] 4y + 6 = 14 [return]   ”

Variable Replacement:

a = “x”

d = -1

b = 4

c1 = 4

c2 = 9

c3 = -7

final = -4

Answer:

-4

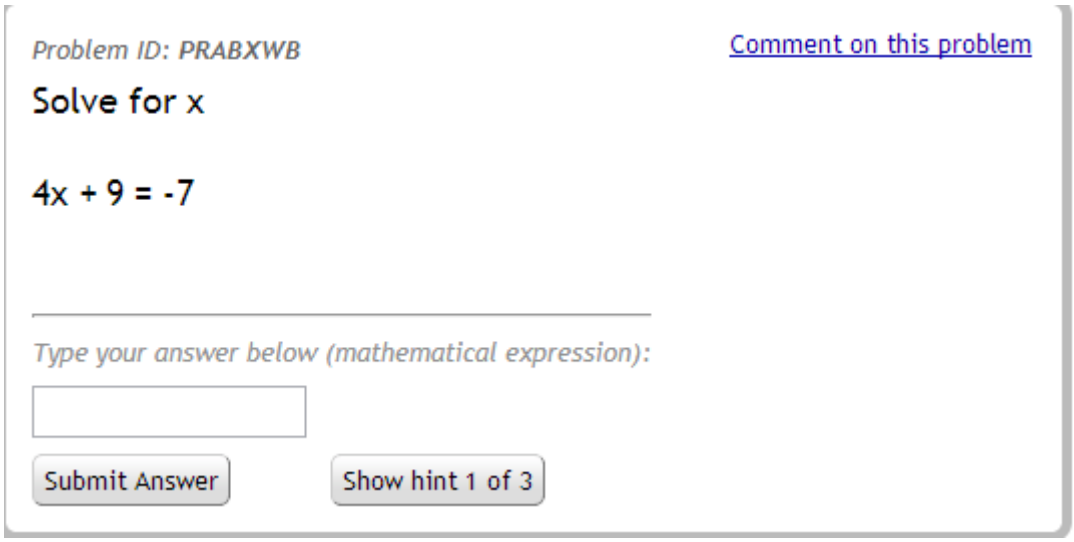


Figure 3.9. Example image of the problem instance for the template example

Since all the problems were generated by the same template, they can be converted back to a generic form. In the problem text for the instance shown in Figure 3.9, there are the input symbols 4, 9, and -7. These numbers can be mapped back to variables. The program names the generic variables as "input" followed by the input number starting at zero. Therefore the first input symbol gets mapped to "input0". In this example 4 will be mapped to "input0", 9 will be mapped to "input1", and -7 will be mapped to "input2". Table 3.14 shows the mapping from the numbers in this instance to the generic numbers.

Instance Numbers	**Generic Input**
4	input0
9	input1
-7	input2

Table 3.14: Mapping from instance values to generic variable names

As the program combines all the expressions into a single list it converts the numbers to generic input symbols as described above. The result is a single list of generic expressions. An example taken from a different problem is shown below.

Example Generic Expression

(input1 * (input0 + input2))

The next step is to delete any duplicate expressions again. This needs to be done again because now our expression list contains all expressions from all the problems. Previously deleting duplicates was done for each problem but did not compare expressions across problems. Deleting duplicates needs to be done at two different times instead of all at once for efficiency purposes. Deleting duplicates for the generic expressions works the same way as it did for the specific instances. Basically the generic inputs are substituted with random values and then the expressions are evaluated. The values are then sorted and iterated over to remove the duplicate expressions.

For each problem, every expression has its generic input substituted in for the input of the given problem. The expression is evaluated and the result is compared to the incorrect answer value. If the result of the expression and the answer value are within 0.0001 of each other than that expression generalizes to that incorrect answer for that problem.

Generalization Criteria

Once all the expressions have been evaluated for each problem, some generalization criteria need to be decided on. The question of when should we say the expression has generalized is asked. Two factors considered for this are the number of answers covered by the expression per problem and the number of problems covered.

The higher the number of problems covered means the more likely the expression truly generalizes across problems. For example, if an expression evaluated to an incorrect answer across all problems, then it is likely that expression is the correct expression that generated the given incorrect answer as opposed to an expression that only generalizes for a few problems. It can also be said that those incorrect answers are all generated from that expression given their respective input numbers. To make this method a little more powerful the number of answers covered per problem is used as well. An expression could generalize across all problems but may only cover a single answer per problem. This happens as a result of a large number of rare incorrect answers, which provide little value. An expression that generalizes across 75% of the problems and covers over 5 answers (5 different students have entered the same answer) is more

likely to be the true incorrect process performed than an expression that generalizes across 90% of the problems but only covers a single answer per problem. Generalizing across a large number of problems, but only a small number of instances for each problem is usually an indicator of noise that is picked up by the small numbers students usually work with.

The question of how many answers per problems and how many problems an expression should cover to give accurate generalization has not been explored extensively. Currently the criteria are that an expression must generalize to at least five problems. In order for an expression to generalize to a problem it must cover at least 5 answers for that problem. Once an expression does generalize it is also used to cover the incorrect answers with a count less than 5, which it covered originally. This creates an unforeseen problem where multiple expressions generalize to the same incorrect answer shown in Figure 3.10.

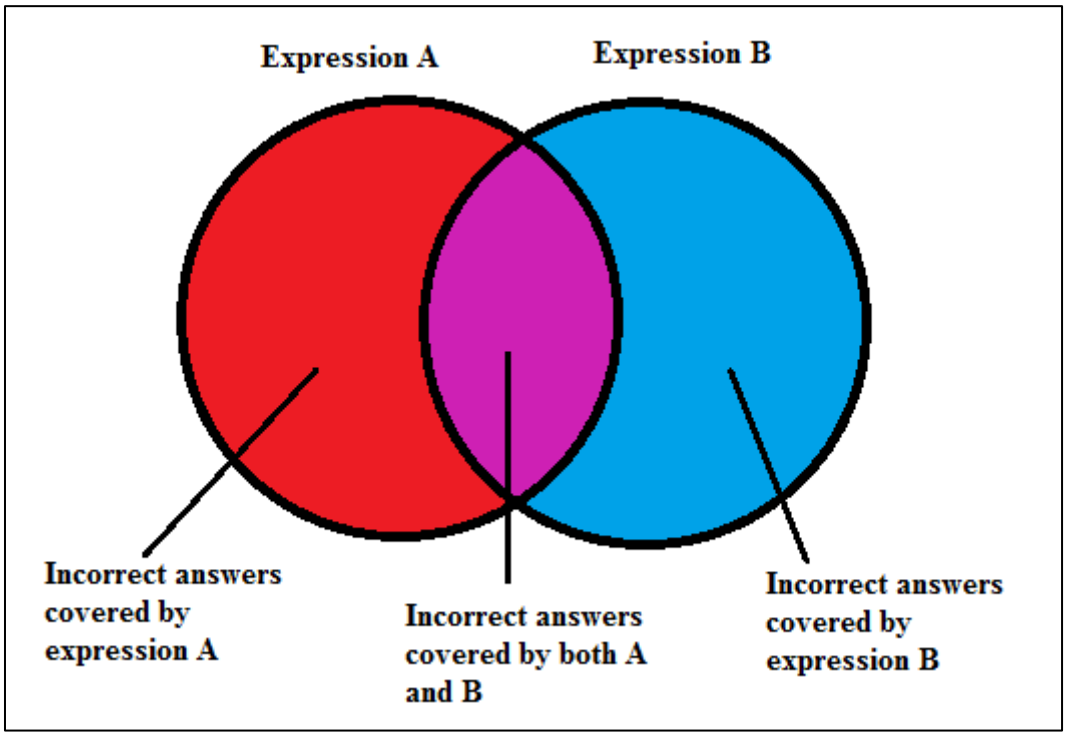


Figure 3.10: Expressions Generalizing to Incorrect Answers

The problem where multiple expressions can generalize to the same incorrect answers can happen for two reasons. One reason is that one expression that is accurate for one incorrect answer may, by chance, also work for another incorrect answer of another problem. Several of the problems in a single template use relatively small numbers where the distinction is not that great among input numbers. Another reason is that both expressions may be valid ways to derive

an incorrect answer. To solve the problem where multiple expressions can generalize to the same incorrect answers a simple an additional step was added to the algorithm. After generalizing expressions the step of choosing the expressions is introduced.

Expression Choosing ("The Chooser")

An iterative algorithm was made to choose which expressions cover which incorrect answers for cases where an incorrect answer is potentially covered by more than one generalized expression. As a result of the generalization step, it is known which incorrect answers of all problems are covered by an expression. This information is stored with the actual expression itself. The iterative algorithm starts by sorting all the expressions by the number of problems they cover. It assumes an expression only covers a single incorrect answer per problem since an expression can only evaluate to one value for a single problem.

Two lists are managed in this algorithm, a current list of expressions and a final list of expressions. The current list of expressions is the list of expressions that the algorithm is still working with and the final list of expressions is the list of expressions that the algorithm will return as the chosen list of expressions.

After sorting the expressions, the first expression that covers the most problems is removed from the current list and added to the final list. At the same time all the incorrect answers covered by that expression are removed from all the other expressions. This is basically saying that this expression has covered those incorrect answers and no other expression can cover them. The logic behind this method is that an incorrect answer is more likely to result from an expression that covers more problems than an expression that covers fewer problems.

Any expressions in the current list that do not cover any problems are removed from the current list. The whole process is repeated until there are no more expressions left on the current list and the final list is returned. The basic steps of the algorithm are listed below.

Step 1. Sort the current list of expressions by the length of expression then by number of problems covered

(Results in a list of expressions sorted by the number of problems they cover then by the length)

Step 2. Add the first expression on the current sorted list to the final list of expressions and remove it from the current list of expressions

Step 3. Remove all incorrect answers and problems from the other expressions that were covered by the expression added to the final list in step 2.

Step 4. Remove all expressions from the current list that do not cover any problems

Step 5. Repeat steps 1-4 until there are no expressions left on the current list

Step 6. Return the final list of expressions

An example of how expressions are chosen is shown in Figures 3.11 – 3.14. Each circle is marked by a letter from A-E to represent expressions. The numbers shown inside the circles represent problem numbers covered by one or more expression. The areas of the circles covering the problem numbers show how an expression covers a problem. The algorithm starts at the initial state in Figure 3.11 and continues to Figure 3.14.

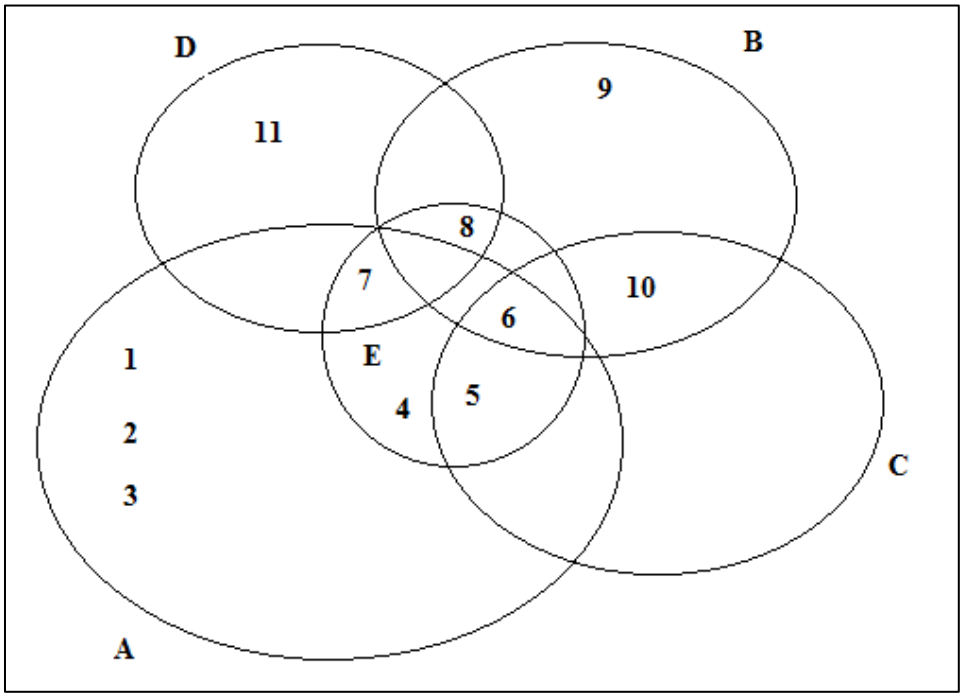


Figure 3.11: Initial State. Each letter corresponds to a circle and represents an expression. Each number represents a problem. Problems covered by expressions are shown as numbers inside the circles.

Current sorted expression list with problems covered

A(7). 1, 2, 3, 4, 5, 6, 7

E(5). 4, 5, 6, 7, 8

B(4). 6, 8, 9, 10

C(3). 5, 6, 10

D(3). 7, 8, 11

Final expression list with problems covered

[empty]

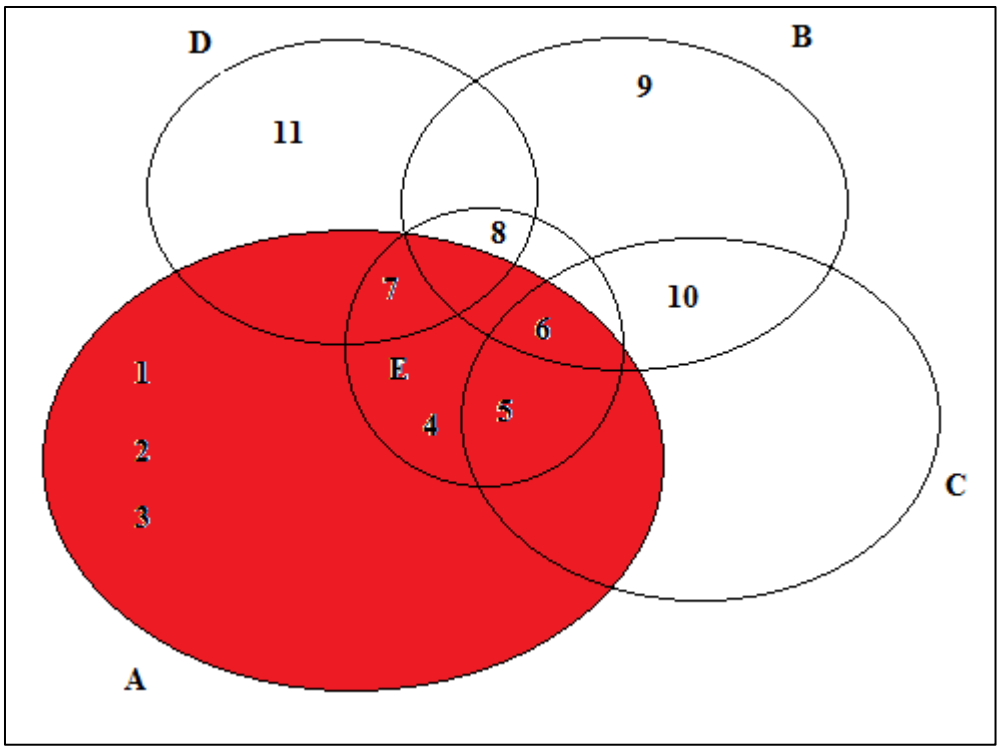


Figure 3.12: After choosing expression A. Expression A is chosen because is covers the most problems. Every problem covered by expression A is now covered only by expression A and not any other expressions.

Current sorted expression list with problems covered after choosing expression A

B(3). 8, 9, 10

D(2). 8, 11

C(1). 10

E(1). 8

Final expression list with problems covered after choosing expression A

A(7). 1, 2, 3, 4, 5, 6, 7

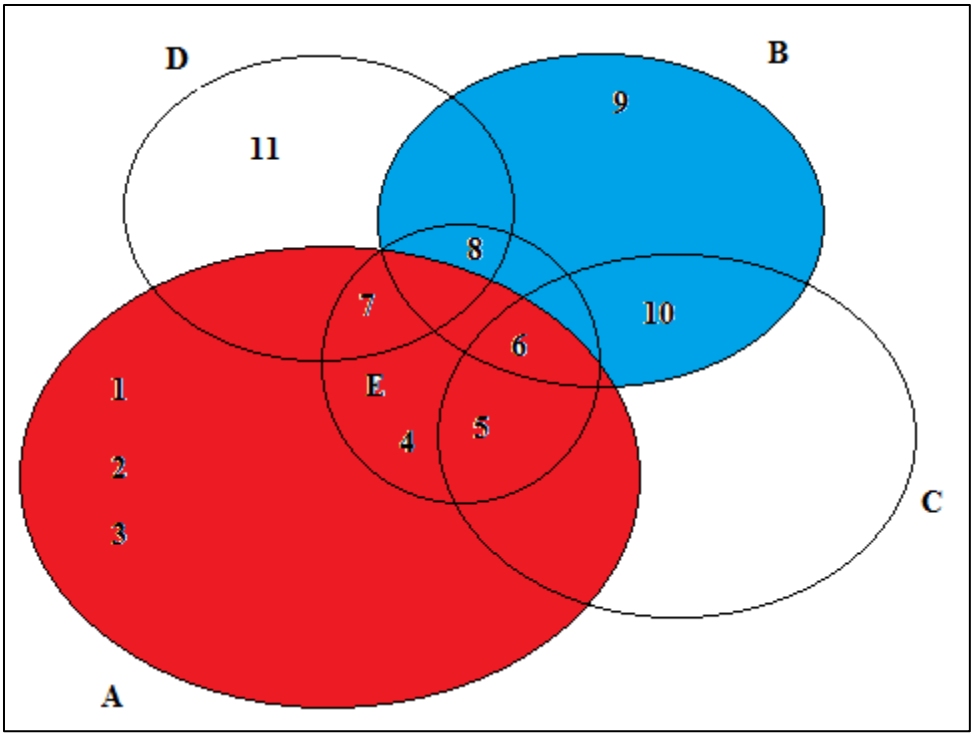


Figure 3.13: After choosing expression B. Expression B is chosen next because it covers the most remaining problems. This leaves expressions E and C with no problems covered; thus removing them from the current list.

Current sorted expression list with problems covered after choosing expression B

D(1). 11

Final expression list with problems covered after choosing expression B

A(7). 1, 2, 3, 4, 5, 6, 7

B(3). 8, 9, 10

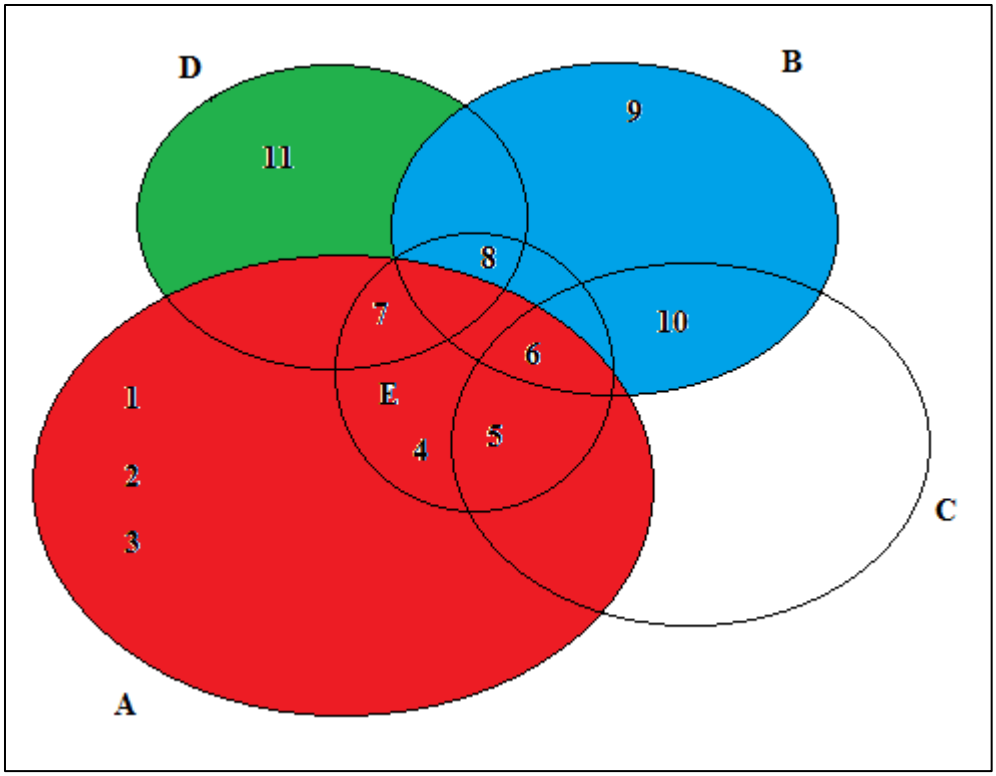


Figure 3.14: After choosing expression D. The final Venn-Diagram shows all problems covered by expression A, B, or D. This diagram is the result of the algorithm running on the initial input in Figure 3.11.

Current sorted expression list with problems covered after choosing expression D

[empty]

Final expression list with problems covered after choosing expression B

A(7). 1, 2, 3, 4, 5, 6, 7

B(3). 8, 9, 10

D(1). 11

Program Evaluation

To examine the effectiveness of the program, two problem sets commonly used in the ASSISTments system were chosen for analysis. Each of these two problem sets consisted of problems generted from two templates each for a total of 4 templates analyzed among the two problem sets. There were around 50 problems generated per template for a total of 200 problems, 100 for each problem set. There were between 4,000 and 10,000 incorrect answers

(not all distinct) for all of the problems generated from a single template. A diagram of the described structure is shown in Figure 3.15.

Two main factors were considered when analyzing the generalized expressions; the percent of problems a given expression covered and the percentage of answers an expression covered. The percent of problems an expression covered is how many unique problems had incorrect answers that could be derived by the expression. This is basically a factor of how well an expression generalized. The more problems that were covered, the more likely it is that the expression generalized across the problems. It is an indicator of common mistakes across the domain of all the instances instead of mistakes made to a specific problem. The second factor is the percent of answers covered, which is the total number of incorrect answers an expression covers across all the instances of the template divided by the total number of incorrect answers across all the instances in a template. This is basically how often are mistakes covered by the given expression.

Problems that could not be derived are included in the problem total and incorrect answer totals. Therefore the analysis represents a worst case scenario or lower bounds on the how many answers the expressions covered. The Deriver rejects all inputs with zeros or inputs with duplicate numbers; however the expressions do still cover incorrect answers for those problems. This was done because the algorithm itself was being evaluated and if the algorithm cannot handle such inputs then those incorrect answers should not be counted as covered. Also only the expressions that covered over 1% of the answers and over 40% of the problems were shown. Expressions that covered less tended to cover the incorrect answers by chance with so many problems having small numbers. Another situation was when some expressions were specific to one or two problems and did not really generalize across all the problems.

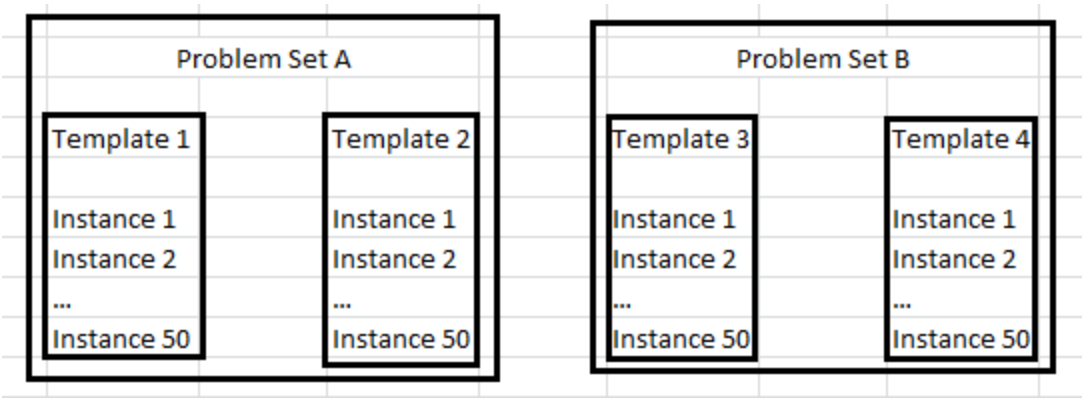


Figure 3.15. Problem Set Structure

Problem Set A: Template 1

Total number of problems = 60

Total number of incorrect answers = 5,542

Total number of expressions generalized = 144

Total number of expressions generalized (> 1% problems and > 40% answers) = 4

Simplified template text: input2 + input0 x input1

Example instance: 5 + 4 * 7

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 43.7 %

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	(input1 * (input0 + input2))	(7 * (5 + 4))	wrong order of operation	71.7 %	34.7 %
2	(input0 * input1)	(7 * 4)	forgot to add as the last step	65 %	4.3 %
3	(input0 * (input1 * input2))	(7 * (5 * 4))	accidentally multiplied all the numbers	43.3 %	3.1 %
4	((input2 - (input1 / input1)) + (input1 * input0))	((5 - (7 / 7)) + (7 * 4))	accidentally added one less	41.7 %	1.6 %

Table 3.15: Expression Table for Problem Set A: Template 1

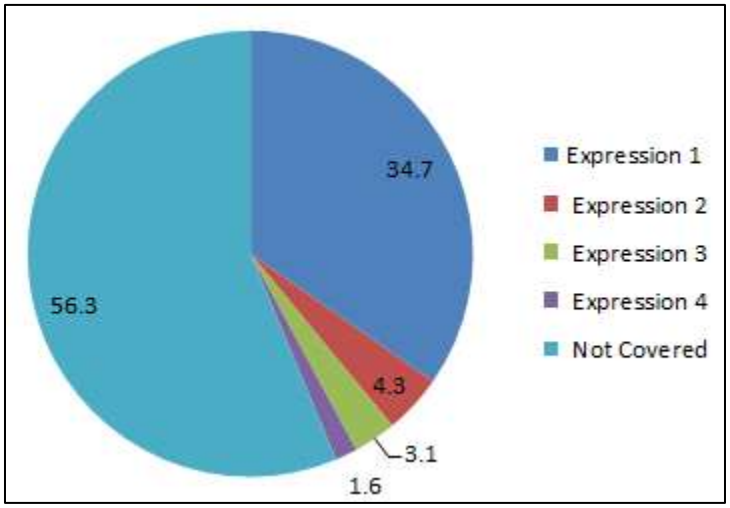


Figure 3.16: Problem Set A: Template 1. Percentage of answers covered by expressions.

Problem Set A: Template 2

Total number of problems = 51

Total number of incorrect answers = 4,034

Total number of expressions generalized = 4

Total number of expressions generalized (> 1% problems and > 40% answers) = 4

Simplified template text: input1 - input0 x input2

Example instance: 49 - 54 * 7

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 58.7

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	((input0 * input2) - input1)	((54 * 7) – 49)	wrong sign for answer	74.5 %	45.5 %
2	((input1 - input0) * input2)	((49 - 54) * 7)	wrong order of operation	56.8 %	5.6 %
3	(input0 * input2)	(54 * 7)	forgot to subtract this number from 49	54.9 %	5.6 %
4	((input0 - input1) * input2)	((54 - 49) * 7)	wrong order of operation and wrong sign	49 %	2 %

Table 3.16: Expression Table for Problem Set A: Template 2

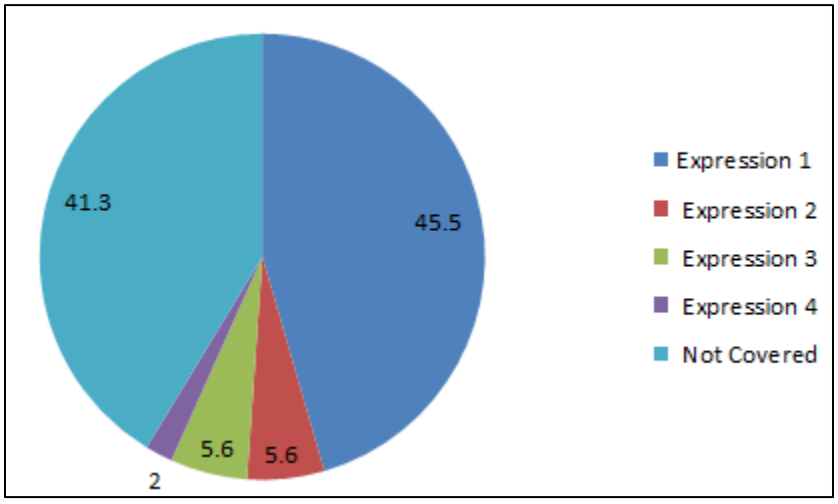


Figure 3.17: Problem Set A: Template 2. Percentage of answers covered by expressions.

From looking at the results of the two templates for problem set A, the algorithm did not do as well for template 1, only covering 43.7% of the incorrect answers. It also found 144 total expressions although only 4 of those met the generalization criteria of covering over 40% of the problems and over 1% of the total incorrect answers. This can be compared to template 2, where the algorithm covered 58.7% of the answers. Most importantly the algorithm only found four expressions and all of them met the generalization criteria. The reason for the improvement is because the instances generated by template 2 uses a wider range of numbers. Using more distinct numbers makes it easier for the program to remove ambiguity and attribute an incorrect answer to a specific expression.

For problem set B similar analysis was done. One extra step taken was for template 3, where the results were harder to interpret than expected. Since the algorithm allows for numbers not given in the input to be injected into the input list, that was done for template 3. This feature was designed for problems that do not explicitly give the input symbols needed. In this case errors were believed to be operational such as being off by 1 or forgetting a negative sign. Other cases, such as problems with percentages, are typically done with 10.0 injected into the input list because it is a common mistake to be a multiple of 10 off (the placement of the decimal point). Template 3 was analyzed both with the regular inputs and with -1.0 injected into the input list.

Problem Set B: Template 3

Total number of problems = 52

Total number of incorrect answers = 6,100

Total number of expressions generalized = 149

Total number of expressions generalized (> 1% problems and > 40% answers) = 6

Simplified template text: input0 x + input1 = input2

Example instance: 11x + 7 = 51

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 36.5

Expression Number#	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	(input2 - input1)	(51 - 7)	forgot to divide for the last step	80.8 %	8.2 %
2	(((input0 / input0) / (input0 * input2)) * ((input2 + input2) * (input2 + input2)))	(((11 / 51) / (11 * 51)) * ((51 + 51) * (51 + 51)))	unknown	53.8 %	8.1 %
3	((input1 + input1) / input1)	((7 + 7) / 7)	unknown	61.5 %	6.3 %
4	(((input0 – input1) + input2) / input0)	(((11 - 7) + 51) / 11)	unknown	71.1 %	5.8 %
5	((input2 – input1) – input0)	((51 - 7) - 11)	subtracted instead of divided	69.2 %	5.2 %
6	(input0 / input0)	(11 / 11)	unknown	53.8 %	2.9 %

Table 3.17: Expression Table for Problem Set B: Template 3

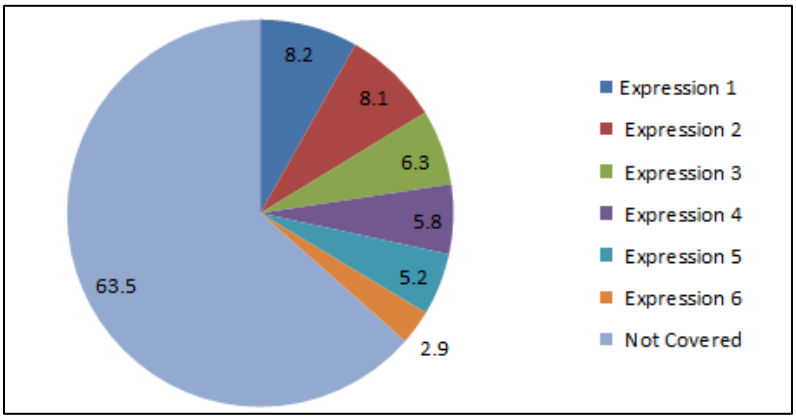


Figure 3.18: Problem Set B: Template 3. Percentage of answers covered by expressions.

Problem Set B: Template 3, -1.0 Injected

Total number of problems = 52

Total number of incorrect answers = 6,100

Total number of expressions generalized = 149

Total number of expressions generalized (> 1% problems and > 40% answers) = 7

Simplified template text: input0 x + input1 = input2

Example instance: 11x + 7 = 51

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 38.2

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	((input1 – input2) / input0)	((7 –51) / 11)	forgot the negative sign or made a mistake with the rule	80.8 %	17.7 %
2	(input2 – input1)	(51 – 7)	forgot to divide for the last step	80.8 %	8.2 %
3	((-1.0 + -1.0) / -1.0)	((-1.0 + -1.0) / -1.0)	unknown	59.6 %	3.7 %
4	((input2 – input1) * input0)	((51 – 7) * 11)	multiplied instead of dividing at last step	69.2 %	3 %
5	(((input2 – input1) / input0) - ((-1.0 + -1.0) * (-1.0 * -1.0)))	(((51 – 7) / 11) - ((-1.0 + -1.0) * (-1.0 * -1.0)))	unknown	44.2 %	2.1 %
6	(input0 + -1.0)	(11 + -1.0)	unknown	51.9 %	1.8 %
7	(input1 + -1.0)	(7 + -1.0)	unknown	51.9 %	1.7 %

Table 3.18: Expression Table for Problem Set B: Template 3, -1 injected

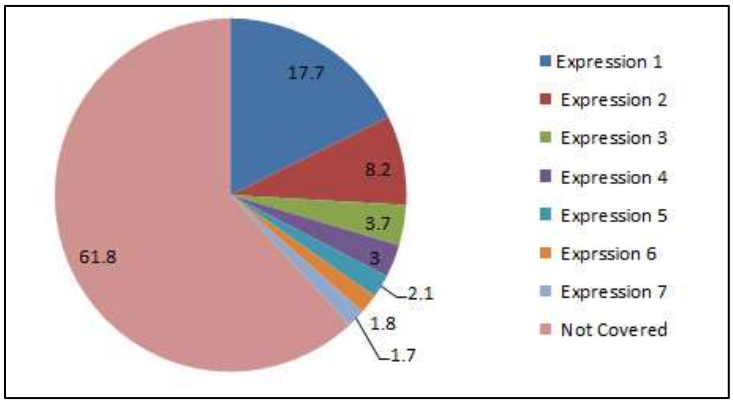


Figure 3.19: Problem Set B: Template 3, -1 injected. Percentage of answers covered by expressions.

Problem set 3 shows two of the main issues with the algorithm. The first issue is how operational errors are handled. This is shown where using -1.0 as an additional input symbol recovers the most common incorrect answer derivation of ((input1 – input2) / input0). 17.8% of these answers were not recovered when just using the original input list shown in Figure 3.18. The main issue here is that the algorithm currently does not explicitly handle operational errors, which makes the results harder to interpret. Since this is the biggest limitation of the algorithm it is immediate future work to create algorithms that can find operational errors and build them into the existing program.

The second issue is interpretability of the results. For template 1, all the reasons of how a student arrived at an incorrect answer were shown in the Table 3.15. For template 2, a reason for what the student was thinking when he or she made the mistake could not be formed for several of the expressions. This is a typical problem with machine learning algorithms in general, where the results are not always easily interpretable. One cause of poor interpretability was that operational errors were not explicitly handled, however this is not the only cause. The interpretability of the results will have to be handled in some way, especially for more complicated problems. Template 2 was a representation of a worst case scenario for the algorithm, as it attacked the weaknesses of the algorithm. Overall the worst case performance was not horrible, although it was not that good either.

Problem Set B: Template 4

Total number of problems = 50

Total number of incorrect answers = 9,792

Total number of expressions generalized = 300

Total number of expressions generalized (> 1% problems and > 40% answers) = 11

Simplified template text: x / input1 + input2 = input0

Example instance: x/4 + 5 = 3

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 50.1

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	((input2 – input0) * input1)	((5 - 3) * 4)	wrong sign	80 %	16.2 %
2	(input0 – input2)	(3 - 5)	forgot to multiply for the last step	72 %	9.7 %
3	((input0 * input1) – input2)	((3 * 4) - 5)	multiplied before subtracting	72 %	6.8 %
4	(input * (input0 + input2))	(4 * (3 + 5))	added instead of subtracted or made a subtraction error	78 %	3.5 %
5	(input0 * input1)	(3 * 4)	forgot to subtract first	74 %	2.6 %
6	(input0 / input0)	(3 / 3)	unknown	66 %	2.5 %
7	(input1 * input2)	(4 * 5)	unknown	78 %	2.5 %

8	(input2 - input0)	(5 - 3)	forgot to multiply and wrong sign	60 %	1.9 %
9	(((input0 – input2) * input1) + input1)	(((3 - 5) * 4) + 4)	unknown	62 %	1.6 %
10	(((input0 – input2) * input1) – input1)	(((3 - 5) * 4) - 4)	unknown	68 %	1.5 %
11	((input0 / input0) + input2)	((3 / 3) + 5)	unknown	52 %	1.3 %

Table 3.19: Expression Table for Problem Set B: Template 4

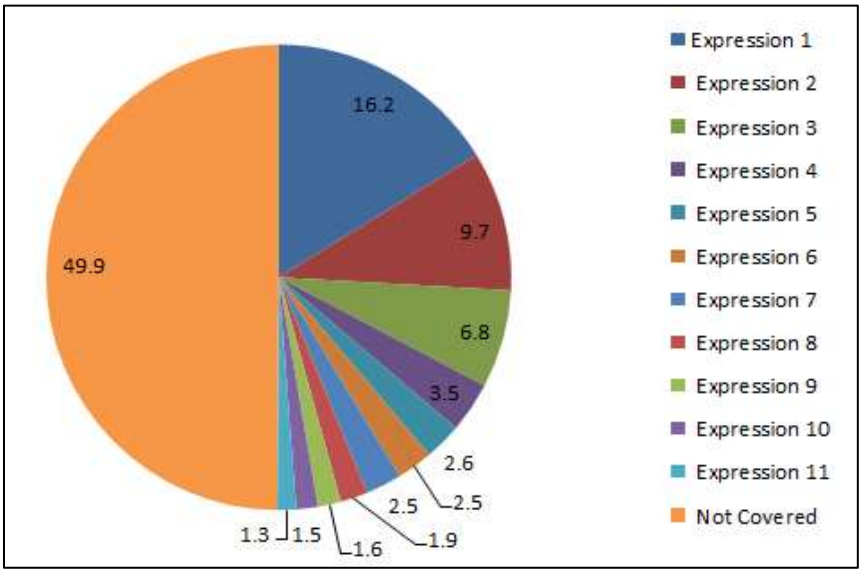


Figure 3.20: Problem Set B: Template 4. Percentage of answers covered by expressions.

After analyzing all four templates in both problem sets, the percentage of answers covered were 43.7, 58.7, 38.2 (-1.0 injected), and 50.1 for the four templates respectively. This initial analysis was a pessimistic view of the algorithms performance because all problems with duplicate inputs were not even used for answers that could be covered. Although the derivation algorithm does not handle duplicate input symbols yet, the expressions that are generated can still be applied to those symbols. All the problems were re-analyzed allowing the expressions to cover these problems to see how many more incorrect answers could be covered. This analysis is a more realistic estimate of the algorithms coverage on incorrect answers.

Problem Set A: Template 1 (duplicate inputs included)

Total number of problems = 60

Total number of incorrect answers = 5,542

Total number of expressions generalized = 262

Total number of expressions generalized (> 1% problems and > 40% answers) = 5

Simplified template text: input2 + input0 x input1

Example instance: 5 + 4 * 7

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 69.9

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	(input1 * (input0 + input2))	(7 * (5 + 4))	wrong order of operation	95 %	54.9 %
2	(input0 * input1)	(7 * 4)	forgot to add as the last step	86.6 %	5.3 %
3	(input0 * (input1 * input2))	(7 * (5 * 4))	accidentally multiplied all the numbers	65 %	4.5 %
4	(input1 + (input0 + input2))	(7 + (4 + 5))	accidentally added all the numbers	75 %	3.5 %
5	(input1 + (input0 * input1))	(7 + (4 * 7))	unknown	50 %	1.7 %

Table 3.20: Expression Table for Problem Set A: Template 1 (duplicate inputs included)

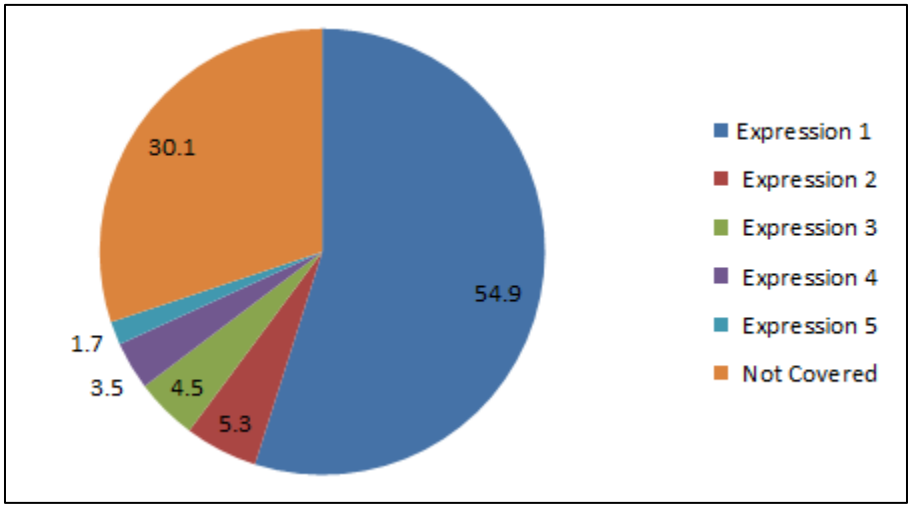


Figure 3.21: Problem Set A: Template 1 (duplicate inputs included). Percentage of answers covered by expressions.

Problem Set A: Template 2 (duplicate inputs included)

Total number of problems = 51

Total number of incorrect answers = 4,034

Total number of expressions generalized =4

Total number of expressions generalized (> 1% problems and > 40% answers) = 4

Simplified template text: input1 - input0 x input2

Example instance: 49 - 54 * 7

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 62.8

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	((input0 * input2) - input1)	((54 * 7) – 49)	wrong sign for answer	80.4 %	48.8 %
2	((input1 - input0) * input2)	((49 - 54) * 7)	wrong order of operation	60.8 %	5.9 %
3	(input0 * input2)	(54 * 7)	forgot to subtract this number from 49	58.8 %	6 %
4	((input0 - input1) * input2)	((54 - 49) * 7)	wrong order of operation and wrong sign	52.9 %	2.1 %

Table 3.21: Expression Table for Problem Set A: Template 2 (duplicate inputs included)

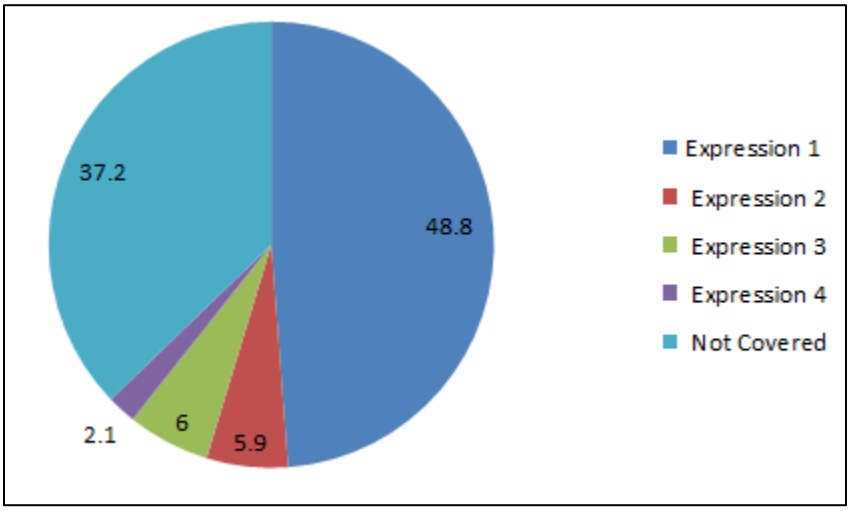


Figure 3.22: Problem Set A: Template 2 (duplicate inputs included). Percentage of answers covered by expressions.

Problem Set B: Template 3 (duplicate inputs included)

Total number of problems = 52

Total number of incorrect answers = 6,100

Total number of expressions generalized = 322

Total number of expressions generalized (> 1% problems and > 40% answers) = 10

Simplified template text: input0 x + input1 = input2

Example instance: 11x + 7 = 51

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 63.1

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	((input1 - input2) / input0)	((7 - 51) / 11)	forgot the negative sign	88.5 %	18 %
2	(input2 - input1)	(51 – 7)	forgot to divide for the last step	88.5 %	9 %
3	((input2 - input1) - input0)	((51 - 7) - 11)	subtracted instead of divided	90.4 %	9 %
4	(((input0 - input1) + input2) / input0)	(((11 - 7) + 51) / 11)	unknown	92.3 %	8.5 %
5	((input0 - input1) + input2)	((11 - 7) + 51)	unknown	55.8 %	4.6 %
6	(((input2 - input0) - input1) / input0)	(((51 - 11) - 7) / 11)	subtracted all the numbers from 51 then divided	76.9 %	4.3 %
7	((input2 - input1) * input0)	((51 - 7) * 11)	multiplied instead of dividing for last step	76.9 %	3.4 %
8	(((input0 - input1) + (input0 + input2)) / input0)	(((11 - 7) + (11 + 51)) / 11)	unknown	69.2 %	3.1 %
9	(((input2 - input1) - (input0 + input0)) / input0)	(((51 - 7) - (11 + 11)) / 11)	unknown	67.3 %	1.7 %
10	(input1 + input2)	(7 + 51)	added 7 instead of subtracting then forgot to divide by 11	50 %	1.5 %

Table 3.22: Expression Table for Problem Set B: Template 3 (duplicate inputs included)

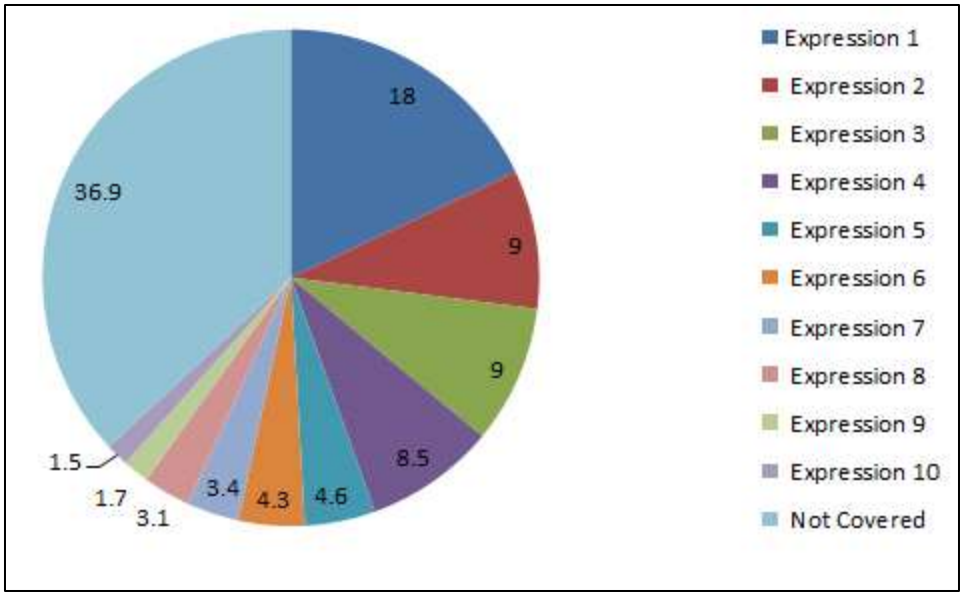


Figure 3.23: Problem Set B: Template 3 (duplicate inputs included). Percentage of answers covered by expressions.

Problem Set B: Template 4 (duplicate inputs included)

Total number of problems = 50

Total number of incorrect answers = 9,792

Total number of expressions generalized = 302

Total number of expressions generalized (> 1% problems and > 40% answers) = 12

Simplified template text: x / input1 + input2 = input0

Example instance: x/4 + 5 = 3

Percent of answers covered by all expressions (> 1% problems and > 40% answers) = 60.6

Expression Number	Expression	Example Instance	Reason	Problem Percent	Answer Percent
1	((input2 - input0) * input1)	((5 - 3) * 4)	wrong sign	78 %	15.3 %
2	(input0 - input2)	(3 - 5)	forgot to multiply for the last step	70 %	9.4 %
3	((input0 * input1) - input2)	((3 * 4) - 5)	multiplied before subtracting	90 %	8.9 %
4	((input1 - input2) + input0)	((4 - 5) + 3)	added instead of multiplying at last step	90 %	6.1 %

5	(input0 * input1)	(3 * 4)	forgot to subtract first	90 %	5.3 %
6	(input1 * (input0 + input2))	(4 * (3 + 5))	added instead of subtracted or made a subtraction error	88 %	4.6 %
7	(input0 - input1)	(3 - 4)	unknown	78 %	2.8 %
8	(input1 * input2)	(4 * 5)	unknown	80 %	2.4 %
9	(input2 - input0)	(5 - 3)	forgot to multiply and wrong sign	70 %	1.9 %
10	(((input0 - input2) * input1) + input1)	(((3 - 5) * 4) + 4)	unknown	66 %	1.5 %
11	(input1 + input2)	(4 + 5)	unknown	76 %	1.3 %
12	(input1 - input0)	(4 - 3)	unknown	54 %	1.1 %

Table 3.23: Expression Table for Problem Set B: Template 4 (duplicate inputs included)

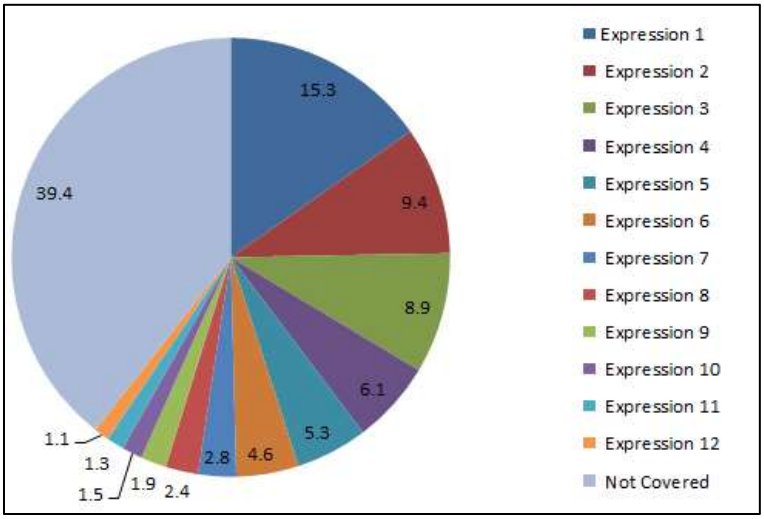


Figure 3.24: Problem Set B: Template 4 (duplicate inputs included). Percentage of answers covered by expressions.

Significant improvement was made when allowing the expressions generalized to cover incorrect answers for problems with duplicate input symbols. The graph in Figure3.25 shows the improvement made for each of the four templates. In all cases the algorithm did better, with some improvements greater than 20%. Template 2 showed almost no improvement, which is a result of template 2 having more distinct numbers and almost no problems with duplicate input symbols. This is also why template 2 generalized the expressions so well in the analysis earlier.

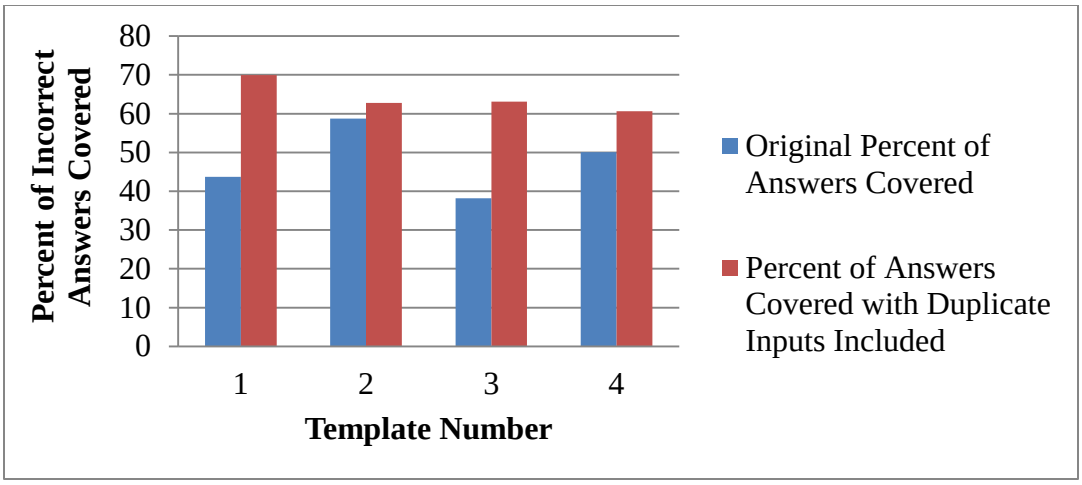


Figure 3.25: Incorrect answer coverage, original problems vs problems with duplicate input

Overall Strengths

The main strength of this algorithm is that no solutions are hard-coded. In past research done on how students arrived at incorrect answers, the algorithms were not as strong. Data was manually analyzed by the researcher and the patterns were manually extracted as what (Brown & Burton, 1978) did. Their program basically just searched for solutions out of a pre-made set of solutions over a small domain of problems (subtraction problems). In their defense, at the time (1980's) this was all that was possible due to the lack of computing power. The program described in this work makes no assumptions about the input or the process. It will find whatever patterns exist given the set of input and operations. This is similar to FOIL (Quinlan, 1990; Quinlan & Cameron-Jones, 1993), except it is much more powerful. FOIL requires a set of data instances to all be valid instances of a function and the data must be in the correct format of a set of attributes and a class attribute. This level of hand-holding falls apart with real data where not just one function needs to be derived at a time. The program described in this chapter can take several expressions, derive and generalize several functions at the same time as well as map all of them to the appropriate incorrect answer. In addition the inputs do not even have to be correct. For interpretation, it would be to the user's advantage to use correct input numbers, but the program was created and designed to persevere no matter what is thrown at it. It will find correct answers whether you want it to or not! Even when given incorrect input, it is still possible to find a correct derivation. For example if the number 2 is required in the expression the program may

output something like ((input0 + input0) / input0), which always evaluates to 2 for any value of input 0.

Another strength of the algorithm is that it runs fairly quickly on a good home computer. To complete the analysis of a template the program will take somewhere between 0.5 - 3 hours to run depending on the template. This is pretty good considering all the work it is doing. The program will not be capable of running in real time, but it does not have to. It does require at most 10GB of RAM, which is on the high end of a home computer. It typically needs 6GB of RAM and uses an extra 2GB in special cases. To be safe it is run with 10GB of RAM. It also requires only 10GB of disk space, which is easily available on any regular computer.

Overall Weaknesses

The algorithms main weakness is that it still requires users to interpret the output. Each expression will need to be interpreted as a conceptual or careless mistake of the student. The appropriate incorrect message will also need to be generated. This process is challenging to do manually since from the data it can only be seen how the student made the error but not why. For example if a student thinks -5 – 3 = 2, the program can easily find how the student arrived at 2 as an incorrect answer. It is hard to tell why the student solved the problem this way. Did the student move the wrong way on the number line by accident or confuse rules about a negative minus a positive? For incorrect message generation this is important to know but it may be impossible to know what the student was really thinking. Some incorrect answers do tend to be the result of certain flawed methodology, which could be a possible area to explore.

A second weakness is that the program does not explicitly look for operational errors such as multiplying incorrectly. A common mistake is that students are one-off with either addition or subtraction and one-off (by a multiple) when multiplying. Despite not being created to explicitly find these type of errors the program still finds them by using (input0 / input0) to get the ‘1’ symbol. It then uses this sub-expression when needed. This causes the results to be less interpretable at the cost of being correct. It is future work to implement algorithms to detect operational errors. Other limitation are that the program currently does not support rounding operations or fractions, which limits the domain of the problems that it can be used for. It also rejects the input symbol of ‘0’ because the search cannot handle zeroes. Obviously it cannot support problems with image as inputs such as problems related to analyzing graphs.

Chapter 4: Reducing Student Hint Use by Creating Buggy Messages from Machine Learned Incorrect Processes

Abstract

The goal of this work is to improve upon existing help in tutoring systems. More specifically, "buggy" messages are improved and compared against existing hints. The machine learning algorithm described in detail in the previous chapter is applied to ASSISTment data and buggy messages are then manually created by the incorrect processes learned from this algorithm. These messages are then run in a randomized control trial in the ASSISTments system to see if they increase student performance compared to just hints.

Introduction

To address several issues with hints mentioned in chapter three, different Buggy Messages are used in this work to provide assistance to students who give a predicted incorrect answer. "Buggy" messages are when a message appears on the screen after a student enters a predicted wrong answer. Messages will only appear if the answer given by the student matches a predicted wrong answer. An example of a problem that has a hint in ASSISTments is shown in Figure 4.1. An example of the same problem that has a buggy message is shown in Figure 4.2. Buggy messages are supported but rarely used in ASSISTments because it takes too much time to predict and enter all the incorrect messages for all possible common wrong answers. The machine learning algorithm in this work can identify most wrong answers and exactly how a student derived them by taking advantage of the existing infrastructure and data in ASSISTments.

Figure 4.1: Buggy message example

Figure 4.2: Hint example

Since an introduction was already provided in chapter three, only a brief version is mentioned in the following paragraphs. Buggy messages were first introduced by Brown and Burton (Brown & Burton., 1978) in 1978 and the early 1980's with their series of "BUGGY" programs. Work with these programs was done in (Brown & Burton. 1978; Brown & VanLehn. 1980; Burton. 1982). Their main goal was to build a database of incorrect answers and the process of how those incorrect answers were generated. VanLehn's early work on bugs has been very influential in the intelligent tutoring field. Others in the AIED community (Jarvis et al., 2004; Matsuda et al., 2011) have looked at machine learning from examples the bug message. Classical Model tracing requires authors to write rules.

Jarvis, Jones & Heffernan report on a technique to try to infer these bug rules from small numbers of instance. Heffernan eventually gave up on that line of research, finding the task of

inferring bug rules to be too difficult, but Masuda et al (Matsuda et al., 2007; Matsuda et al., 2011) have push this field further, and reports more success on this challenging task of inferring the cognitive models rules that explain errors..

Systems like Webwork or StudyIsland appear to generate problems from some sort of algorithmic process. Therefore, our contribution could help any system that has a template like way of generating answers. We image that this algorithm's results can then be presented to teachers who can then write bug message for that function.

Randomized Control Trial Design

The purpose of this study is to see if specifically targeting a student's incorrect answer and providing them with immediate specialized help in the form of buggy messages will result in an increased learning rate. To evaluate the effectiveness of the buggy messages a randomized control trial was run on the ASSISTment's intelligent tutoring system. In ASSISTments two problem sets were chosen to run the randomized control trial on, "Order of Operation" and "Solving Equations". Both problem sets are typically taught in 7th grade (student ages 11-12). An example problem for "Order of Operations" with hints is shown in Figure 4.3 and with buggy messages in Figure 4.4. An example problem for "Solving Equations" is shown with hints in Figure 4.5 and with buggy messages in Figure 4.6. A complete list of all the problems in both problem sets can be found at [1] for "Order of Operation", and [2] for "Solving Equations". Each of these problem sets were generated by two templates each. 25 instances of each template were generated for a total of 50 different problems that each condition group could potentially get, where instances are generated simply by plugging in different numbers for the variables in the template. Instances that contained one or two of the same number (Ex: "7 + 7 * 7") were replaced with different instances and duplicate instances were also replaced with different instances.

For each of these problem sets, the students must get three questions right in a row to complete the problem set. Students were randomly assigned to one of the two condition groups. The control group received the existing content in ASSISTments. Typically the standard form of help in ASSISTments is just a single hint consisting of the answer to the problem. However in

[1] https://sites.google.com/site/selentits2014/home/order-of-operation
[2] https://sites.google.com/site/selentits2014/home/solving-equations

this experiment those problem sets were not chosen. The chosen problem sets contained customized sets of hints to provide a stronger control group. This method may not be the ideal way to obtain good results, but it was done to compare the effectiveness of buggy messages to the strongest possible existing tutoring in ASISTments. The experiment would not have much external validity if the control group was weak or sub-optimal. The experiment group used the same content as the control group with buggy messages added to the problems.

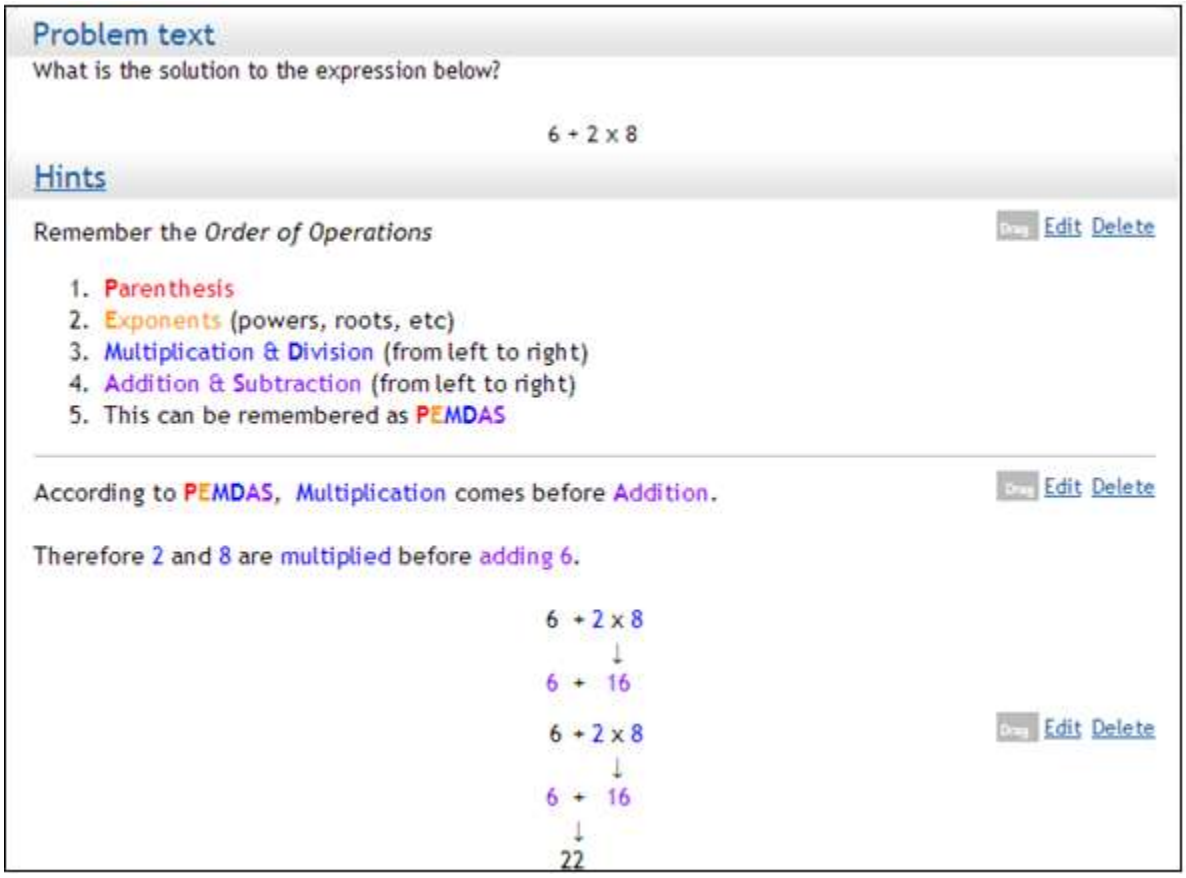


Figure 4.3: Example of a problem with hints (control group) on "Order of Operations" shown inside the ASSISTment builder.

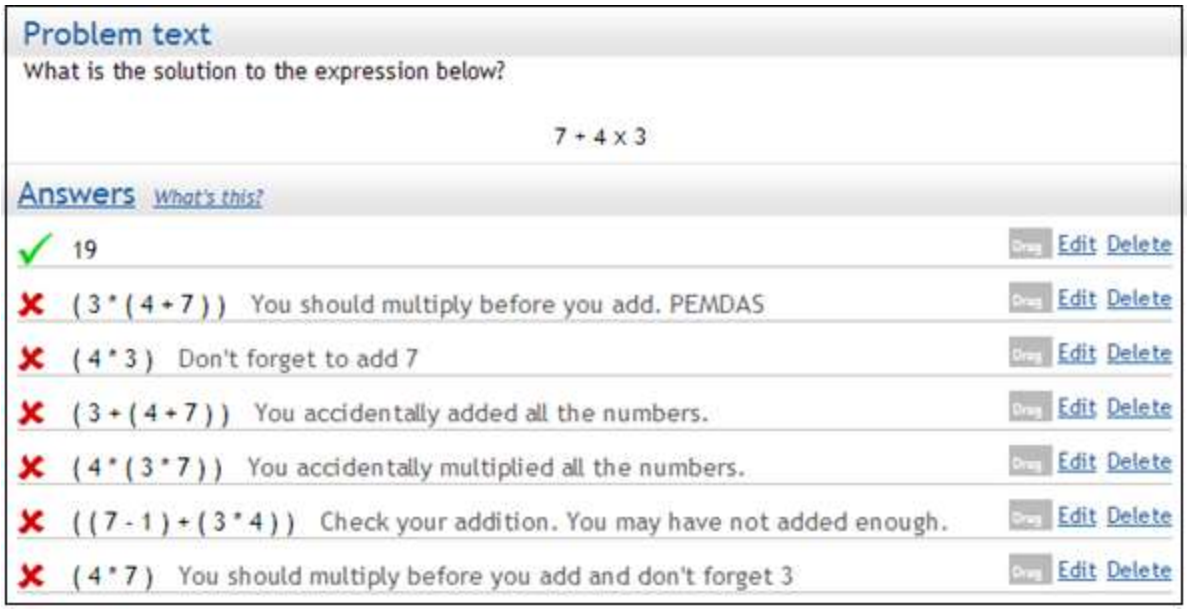


Figure 4.4: Example of a problem with buggy messages (experimental group) on "Order of Operations" shown inside the ASSISTment builder.

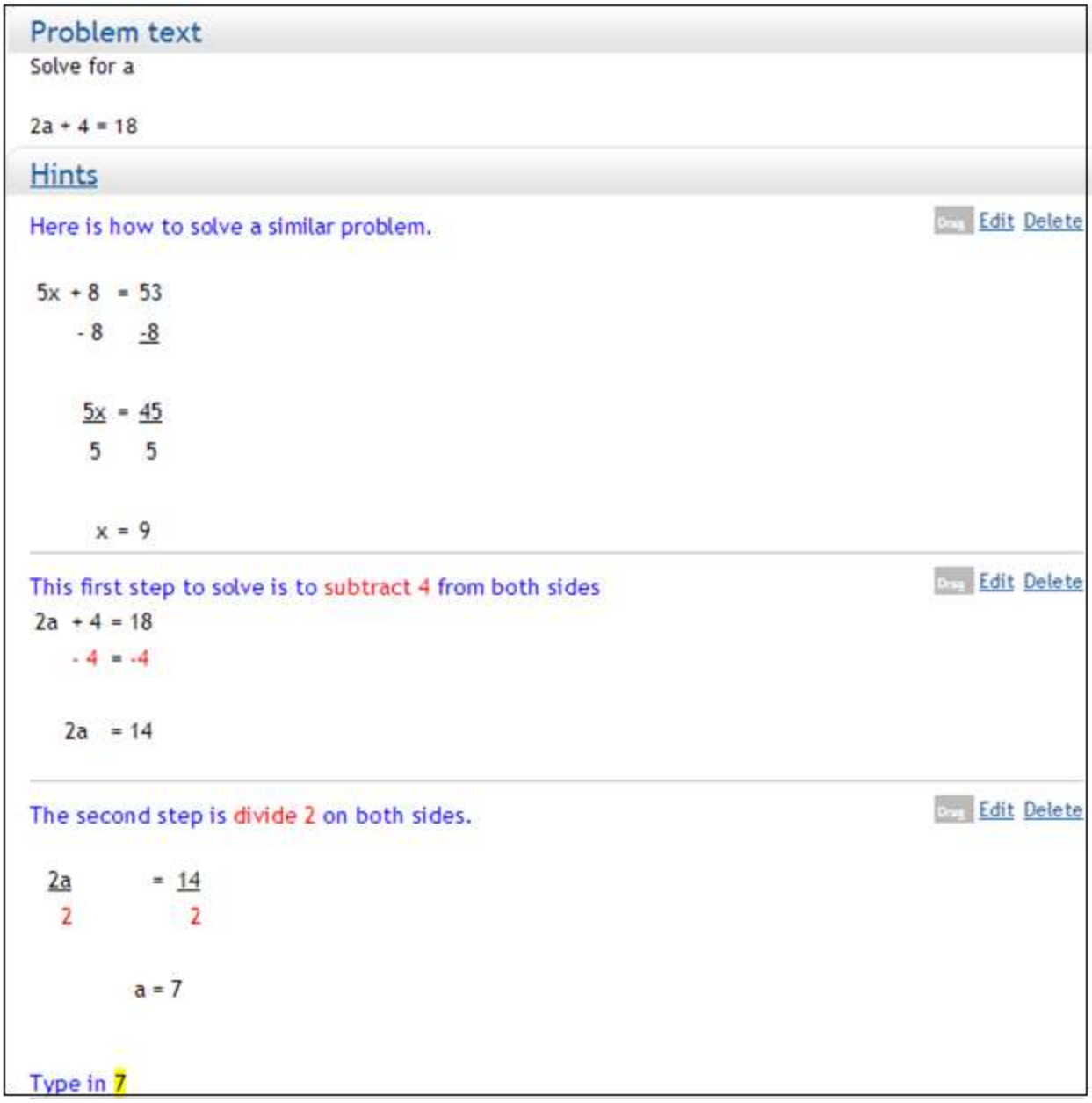


Figure 4.5: Example of a problem with hints (control group) on "Solving Equations" shown inside the ASSISTment builder.

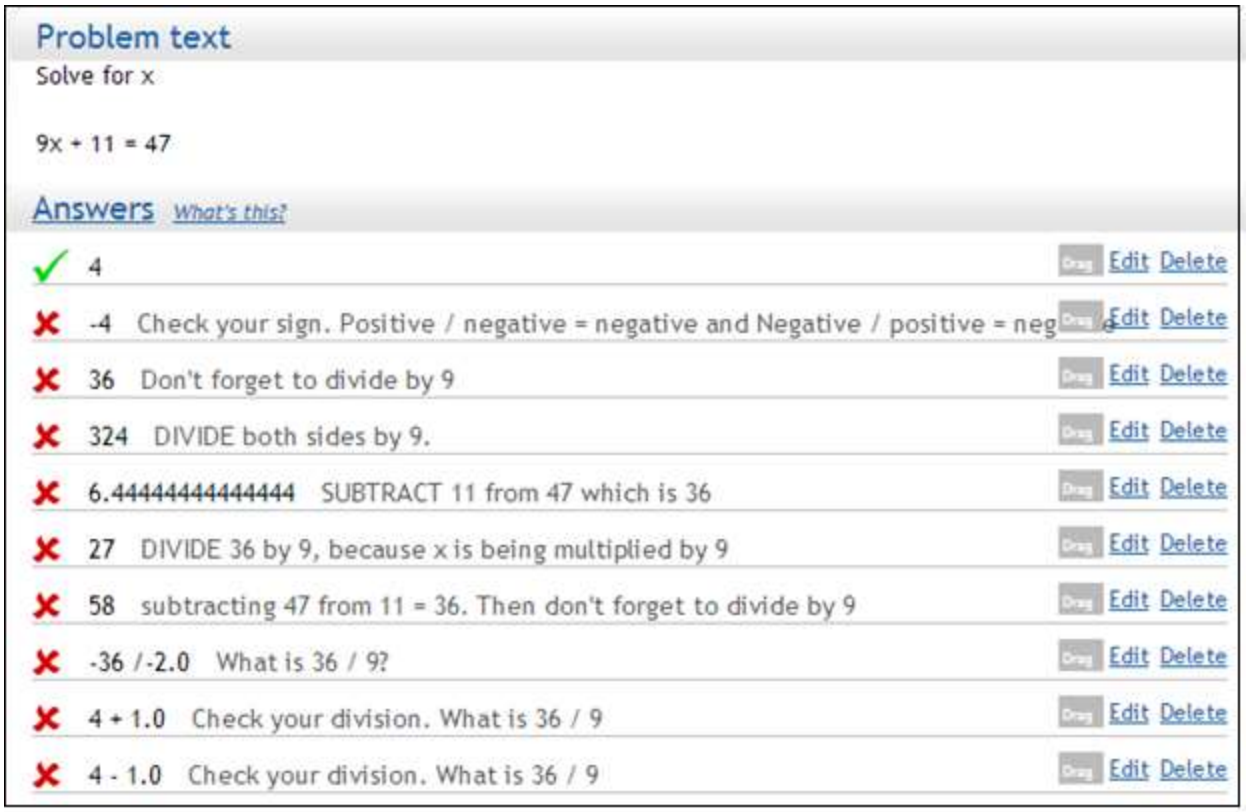


Figure 4.6: Example of a problem with buggy messages (experimental group) on "Solving Equations" shown inside the ASSISTment builder

Buggy messages were used with the existing hints rather than by themselves due to the results of a previously run pilot experiment. Taking hints away from students is risky because it prevents them from continuing on to the next problem until they get the current problem correct. The last hint provides the students with the answer to the problem allowing them to continue on to the next problem. If only buggy messages were used, the students would have to get the correct answer on their own in order to continue on to the next problem. Results from the pilot study on a homework assignment showed that sometimes when the student's answer is not caught by the buggy message they will get completely stuck and be unable to continue with the assignment. This is the reason why buggy messages were added to the existing content rather than being by themselves. The students need a way to continue in the situations where the buggy messages do not help. This design also makes the condition groups invisible to the users since their problems look exactly the same. It is only different when an incorrect answer is caught by the buggy messages, which is much more subtle than the disappearance of the hint button.

The buggy messages used in the problem sets were generated by the machine learning algorithm in this work. All ASSISTment data (roughly 5 years of data) was supplied to the machine learning algorithm. This consisted of all the incorrect answers for every problem generated by the given template. The algorithm was applied to each of the four templates to generate the generalized incorrect solution paths for the problems. From all these incorrect solution paths, the paths that covered over 40% of the problems and over 1% of the incorrect answers were used to supply incorrect messages for. Each incorrect message was created by me.

Analysis

To analyze the randomized control trial three types of information were examined. Several basic statistics we analyzed to see if the buggy messages made a difference in student learning. Secondly, a performance curve was looked at to see if there were any increases in learning for a certain number of problems. Lastly, some error analysis was done to get a deeper understanding of what mistakes were being made. All students that did not receive any form of help were removed from the results since they did not use any form of tutoring. For Order of Operation, 71/263 unique students completed the problem set without making any mistakes and were removed. For Solving Equations, 213/378 unique students completed the problem set without making any mistakes and were removed. This left 192 unique students who received some form of help for the Order of Operations problem set and 165 unique students for Solving Equations.

Basic Statistics

A brief description of the statistics analyzed is listed below. The dataset and complete analysis tables can be found at[3]. The important points of the analysis are summarized in the paper and most of the unimportant results are not shown or mentioned. These statistics are shown Figure 4.7 comparing the control group, which only received hints, to the experimental group, which received buggy messages in addition to the existing hints. How well students performed on the next question after their first mistake was looked at but not included because the trends were similar.

[3] https://sites.google.com/site/selentits2014/home/datasets

1. **Correctness** – The average correctness per student.
2. **Hints** – The average number of hints used per student.
3. **Hints per Problem** – The average number of hints used per problem per student.
4. **Last hint per problem** – The average number of last hints used per problem per student. The last hint gives the student the answer to the problem.
5. **Attempts per Problem** – The average number of attempts per problem per student. An attempt is when a student enters an answer (correct or incorrect).
6. **Time per Problem** – The average time per problem per student rounded to the nearest second where the median time was taken for each student. Times greater than 20 minutes and less than 0 were removed.
7. **Problems per Student** – The average number of problems it took a student to complete the problem set per student. This statistic can only be calculated at the problem set level.

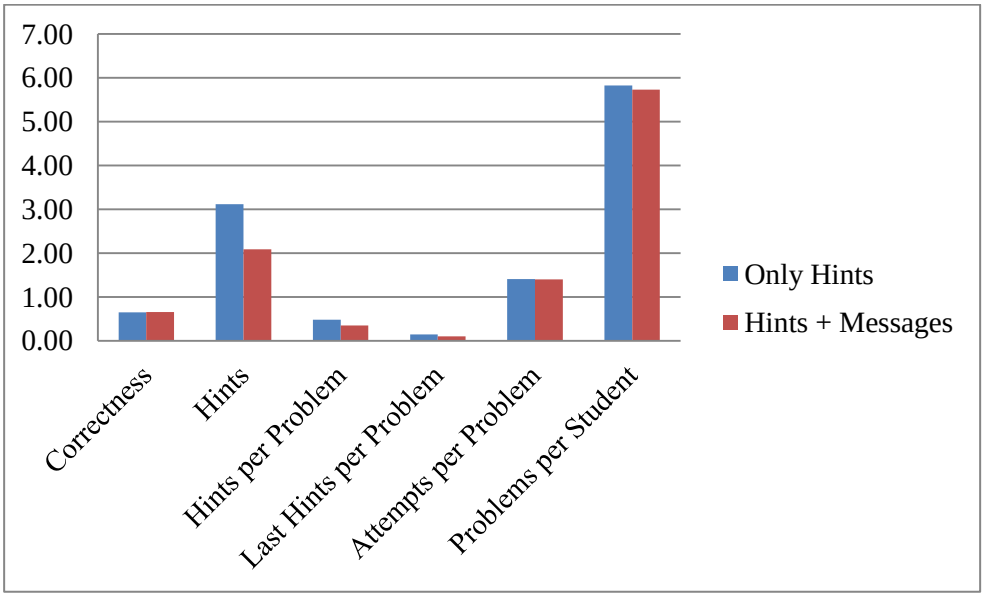


Figure 4.7: Hints vs. Hints + Buggy Messages

Similar trends existed for both the Order of Operation and Solving Equation problem sets, therefore both were combined in Figure 4.7. Out of all the statistics examined, the only significance difference was the amount of hints used, with the experimental group using fewer hints. The control group that only had hints used an average of 3.12 (n=172) total hints, where the experimental group that had buggy messages used an average of 2.09 (n=184) total hints. This is a significant difference ($p<.03$) with an effect size of 0.24. Hints per problem also had a

significant reduction (p=0.03), with an effect size of 0.24, where the group with hints (n=172) used 0.48 hints per problem and the group with buggy messages (n=184) used 0.35 hints per problem. There were no significant differences in correctness, time spent, or number of problems. This means that the buggy messages were able to reduce the amount of hints used without decreasing learning.

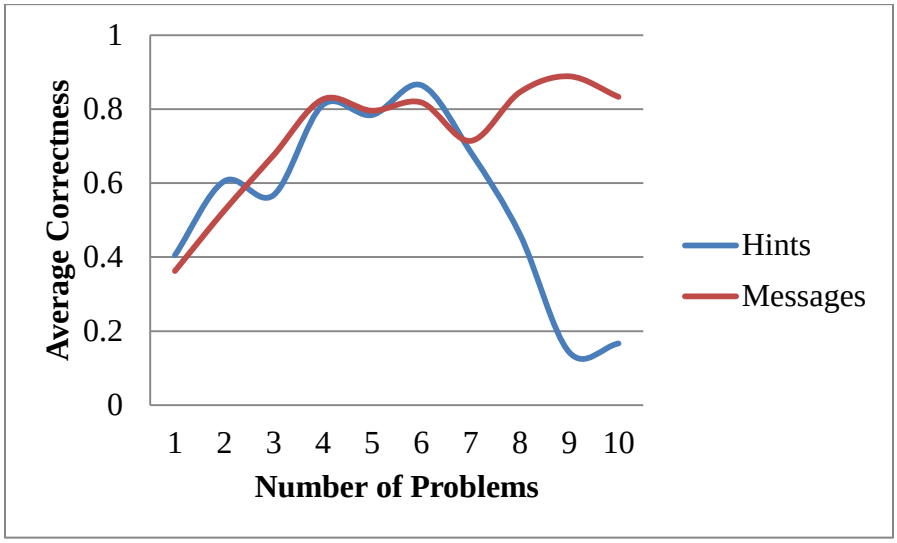


Figure 4.8: Average Correctness per Problem

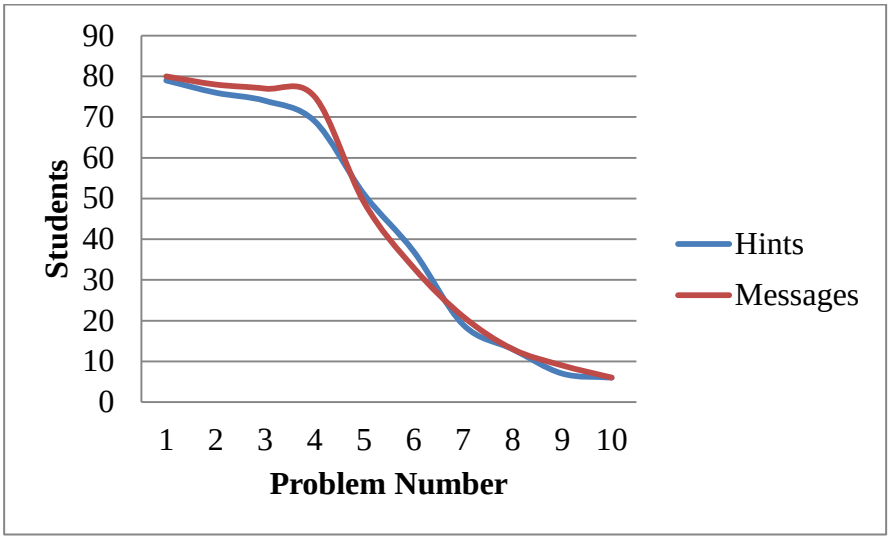


Figure 4.9: Number of student vs. Number of problems done

Learning Curve

The student learning curve is shown in Figure 4.8 for Solving Equations. As the number of problems increases, the number of students decreases, because students either finished the problem set or gave up. This can be seen in Figure 4.9. From these graphs the point to see is what happens from problems 8-10. At problem 8 there are 13 students in both the control group and the experimental group. There is a noticeable difference in the graphs for correctness on these problems. Those students, who received buggy messages and reached 8 problems, finished the problem set within 10 problems 93% of the time. Those students, who did not receive bug messages and reached 8 problems, finished the problem set within 10 problems 54% (7/13 students) of the time. Therefore buggy messages were able to help more than hints for the small subset of <4% of the students.

Error Analysis

From the previous analyses, buggy messages reduced the number of hints used but do not appear to provide any additional help to most of the students. Other than the fact that the existing hints are good, the question of why are the buggy messages not providing additional help arises. The types of errors students made were analyzed. Figure 4.10 shows that students typically do not make the same mistake twice regardless of whether or not they see a buggy message. For all error processes, the control group made the same mistake twice 12% of time and the experiment group made the same mistake twice 15% of the time. This is similar to findings in (VanLehn, 1982) where few students made systematic errors. More data on this analysis can be found at[4]. If students are not making the same mistakes twice then what mistakes are they making? For each erroneous process a percent of the students making another known process was calculated. Figure 4.10 shows all the known incorrect processes students made for the first problem set. For both the control and experimental group there were eight processes total. The control group errors are represented as the odd process numbers and the experimental group errors are represented as the even process numbers. This means that processes 1 and 2 are the same where process 1 is in the control group and process 2 is in the experimental group. This is true for the other processes. Combining all the results for both problem sets shows that 63% of the time students make another known process error.

[4] https://sites.google.com/site/selentits2014/home/datasets

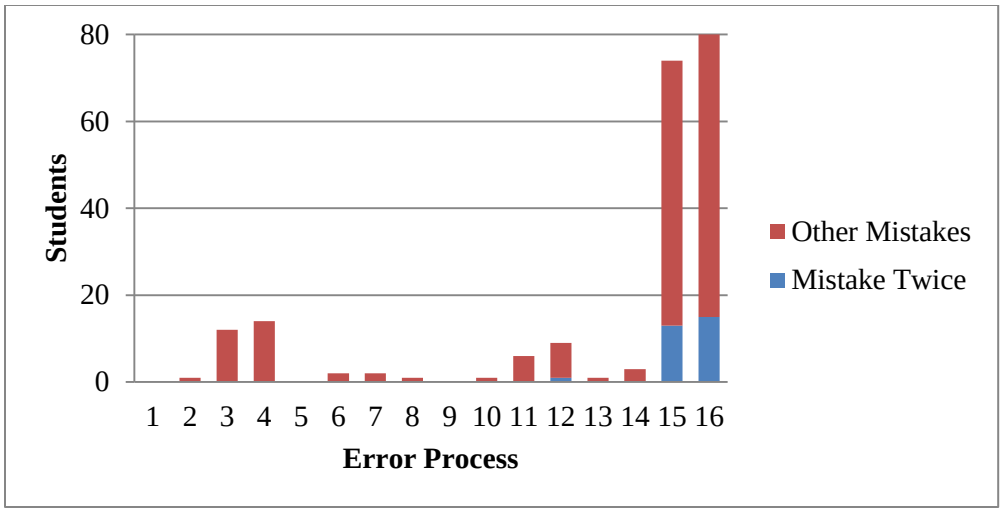


Figure 4.10: This figure shows how many students made the same mistake twice for known incorrect processes in the first problem set. The control group has odd error process numbers and the experimental group has even error process numbers. Process 1 and 2 are the same but for different groups, as are 3 and 4.

Contributions, Conclusions and Future Work

The contribution that this work makes is a study using buggy messages where we showed that although we did not get an increase in student learning, we did show a positive effect in decreasing the number of hints that students use. This suggests that some of the bug messages were helpful, and thus providing some level of evidence to suggest this algorithm could be used in practice to find common answer patterns.

We also replicated Van Lehn's work, reporting that we had a large number of common wrong answers that were not systematic, where few students make the same mistake twice. It showed that students rarely make the same mistake twice regardless of whether or not they received specialized feedback on their mistakes. Merely being told they were wrong was enough for students not to make the same mistake twice. It also shows that receiving the specialized feedback reduced the amount of students that asked for help in the form of hints without inhibiting learning.

The process of learning exactly what the student's erroneous process was and being able to provide specialized help are two separate processes. The machine learning process solves the problem of identifying the erroneous process but does tell the author how to make a good buggy message that will be helpful to students. As future work there needs to be a way to create buggy messages that are known to be good buggy messages, since the ones used in the randomized control trial may, or may not, have been good messages.

Chapter 5: When More Intelligent Tutoring in the Form of Buggy Messages Do Not Help

Abstract

This chapter reports on the null results from two large scale randomized controlled trials that were run in the ASSISTments online tutoring system. Both studies attempted to use reactive buggy messages to help students learn; one in the form of short 20-30 second videos and another in the form of large color-coded text. Buggy messages were supplied for common wrong answers for one-step equation problems in both studies. Despite the large amount of prior research done on error analysis, both interventions using the predicted common wrong answers were unsuccessful at helping students.

Introduction

A "buggy message" is where each hint is unique and specific to the incorrect answer entered by the student. The notion of "bugs" was first introduced by Brown and Burton in 1978 and the early 1980's with their series of "BUGGY" programs (Brown & Burton, 1978). Their first program called BUGGY was designed as a game where teachers try to diagnose the bugs in the student's incorrect process. A manually generated procedural network (tree structure of skills) for multi-digit arithmetic was used by the program to generate the incorrect answer. The main goal of BUGGY was to show teachers that incorrect answers supplied by the student are not random and are a result of systematic errors and misconceptions. An extension of the BUGGY program called DEBUGGY and IDEBUGGY (an online interactive version of DEBUGGY) attempted to match the incorrect answer to a set of previously defined bug rules (Burton, 1982B). Brown and VanLehn continued this work using repair theory to explain why students make some errors but not others and the cause of the errors (Brown & VanLehn, 1980). The idea of repair theory is that a student will attempt to solve a problem with incomplete knowledge. At some point during the problem solving problem process the student will recognize that they have made a mistake, noted as an impasse. Once the student has hit an impasse, the student will attempt to repair the procedure to proceed past the impasse. After running some empirical studies VanLehn found that errors were not stable and students did not consistently make the same mistake twice (VanLehn, 1982).

Sleeman also created a program to discover bugs. Sleeman ran experiment on a group of 24 14-year-old students on algebra problem using the Leeds Modeling System (LMS) bug database to establish classes for the different types of errors (Sleeman, 1985). Further analysis was done on the types of errors student's made in attempt to explain bug-migration (inconsistent student errors). Sleeman proposed the mechanism of Mis-generalization (incorrectly inferring rules) to explain bug migration and why LMS originally missed several bugs (Sleeman, 1984).

Throughout the 80's and 90's there was a large amount of work focused on student modeling. This work includes student modeling based on the incorrect answers and automatically discovering the incorrect processes that generated the incorrect answers. Sison summarizes several machine learning programs that attempt automatically construct student models (Sison & Masamichi, 1998).

More recent work has been done with misconceptions in the DIAGNOSER project. DIAGNOSER is an online tutoring system providing both assessment and tutoring for high school physics students. It is able to identify student errors and misconceptions, which are called facets and provide that information to the teachers, who can then provide help specific to the misconception (Thissen-Roe, Hunt, & Minstrell, 2004).

The two studies in this chapter follow up on the randomized controlled trial run by Selent and Heffernan, who tried to help students with buggy messages (Selent & Heffernan, 2014). Since it was believed that their initial experiment failed due to poor tutoring content, two additional randomized controlled trials were run with focus on improving the tutoring content. One experiment used buggy messages in the form of short 20-30 second videos. Combining both buggy messages with short videos has not been done before and is a novel form of tutoring. Another experiment provides buggy messages with large color-coded text providing both context to the problem and the reason why the answer is incorrect. The hypothesis is that the answer-specific help would help students more than on-demand help.

This chapter reports on the null results of two randomized controlled trials run in the ASSISTments online tutoring system, which attempted to help students by using various forms of buggy messages. In the first study, a total of 490 students participated, where the experimental group received buggy messages specific to the given answer in the form of color-coded text. In the second study a total of 1,379 students participated, where the experiment

group received buggy messages in the form of 20-30 second videos. In both studies, the buggy message intervention had no significant effects of student performance.

Study 1 (Text Buggy Messages)

A skill builder problem set was created for problems on solving one-step equations with integers, which is typically taught in seventh grade for students of ages 12-13. The control group received problems where students had the option to click on a hint button. A student could click on the hint button up to three times. The first hint contained a similar example problem, showing all the steps to reach the solution. The second and third hint contained steps to solve the current problem each time the hint button was clicked, until the last hint revealed the answer to the problem. This is currently the best form of help for this problem set in the ASSISTments system, therefore was used as the problem set for the control group. The experiment group received the same problems as the control group with the addition of buggy text messages that appeared as soon as a student submitted an incorrect answer. These buggy messages showed the steps of the problem the student did correctly and the step that the student performed incorrectly as well as providing the correct rules the student needed to fix their mistake. If a student entered an incorrect answer that was not predicted, a generic message was shown to the student to notify the student that he/she had answered the problem incorrectly. An example problem for the control group with all the hints requested is shown in Figure 5.1[5]. An example problem for the experiment group with a buggy message is shown in Figure 5.2. This experiment design compares a strong control group, where students have a set of hints that walk through an example problem, to an experiment group that adds buggy messages to the current problems.

[5] All figures showing example problems are modified screenshots of what is presented to students in the ASSISTments system. They were modified to fit into the spatial constraints of the paper. Only the location and spacing of the content has changed and not the content itself.

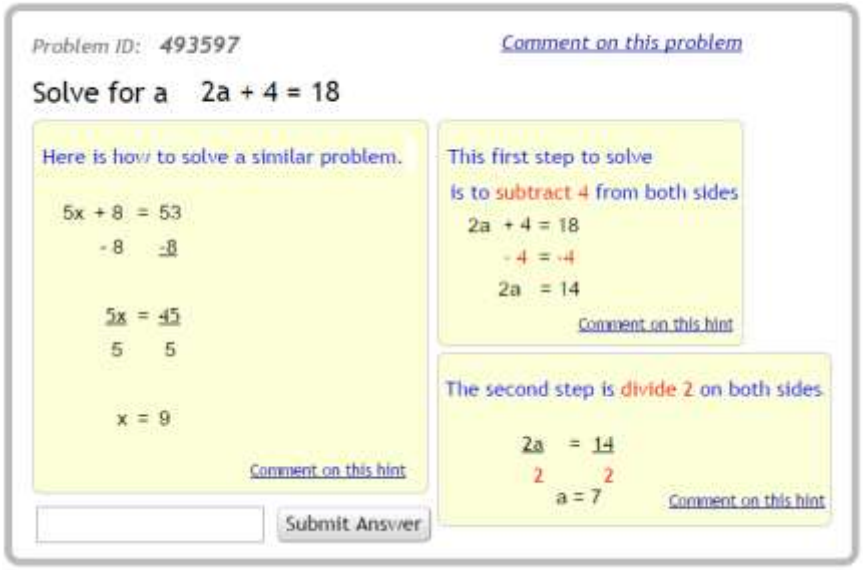


Figure. 5.1: Example Problem - Control Group Showing All Hints

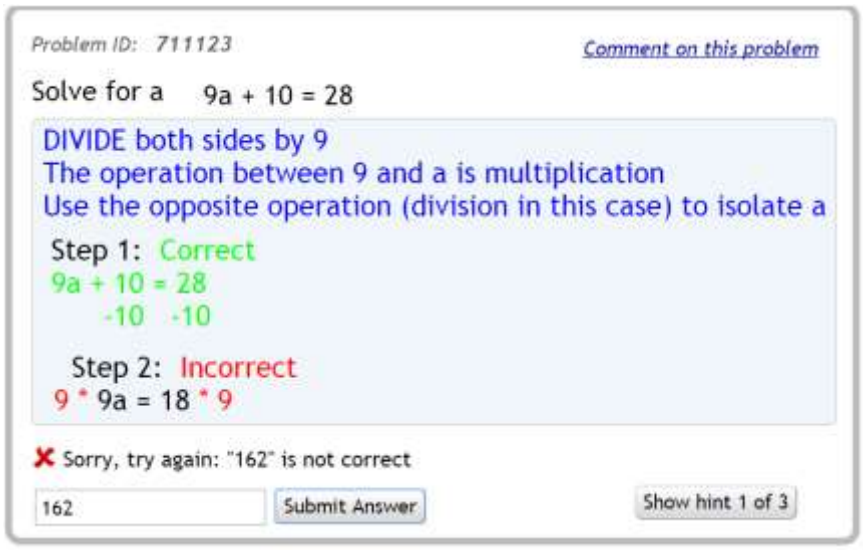


Figure 5.2: Example Problem - Experiment Group Showing a Buggy Message

The skill builder problem set for each condition consisted of a problem bank of fifty problems for each condition. Two templates were used to generate twenty-five instances of each type of problem for each condition. Table 5.1 shows the two different templates used to generate the instance problems. Both the variabilized template form and a concrete example of each template are shown for better understandability. Students in each condition were randomly assigned one of the fifty instances with their respective forms of tutoring until they completed the problem set.

	Variabilized Template	Example Instance
Template 1	%v{c1}%v{a} + %v{c2} = %v{c3}	2a + 4 = 18
Template 2	%v{a} / %v{b} + %v{c} = %v{d}	b / 6 + 11 = 5

Table 5.1: Example Templates

Study 2 (Video Buggy Messages)

Similar to the first study, a skill builder problem set was created for problems on solving one step equations with integers. The control group received problems where students had the option to click on a hint button which gave the answer. This is the standard form of help in ASSISTments, therefore was used as the control. The experiment group received problems that did not have the option to click on a hint button, but received help in the form of video buggy messages. Students received a short 20-30 second video when they entered a predicted incorrect answer. This video explained what process the student used to arrive at their incorrect answer and how to start on the correct solution path. The videos were created by Andrew Burnett, a former middle school teacher, where Andrew explained the problem using a white board. The videos did not give the solution to the student. If a student entered an incorrect answer that was not predicted, a generic message stating that the student's answer was incorrect was shown. An example problem for the control group is shown in Figure 5.3 and the same example problem is shown for the experiment group in Figure 5.4. This experiment design compares a weak control group, where students only had the option to see the answer, to an experiment group that had proactive help in the form of video buggy messages hints but did not have on-demand hints.

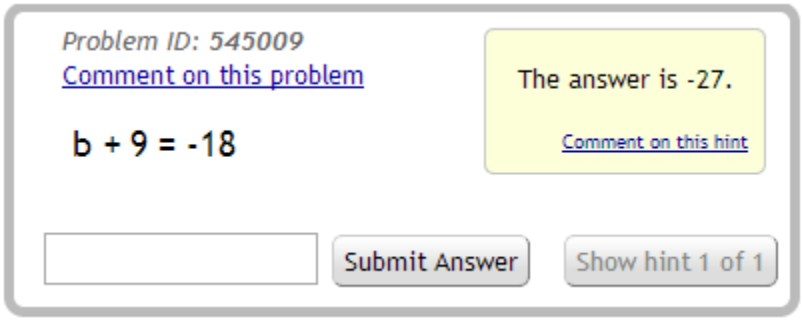


Figure 5.3: Example Problem - Control Group Showing the Answer

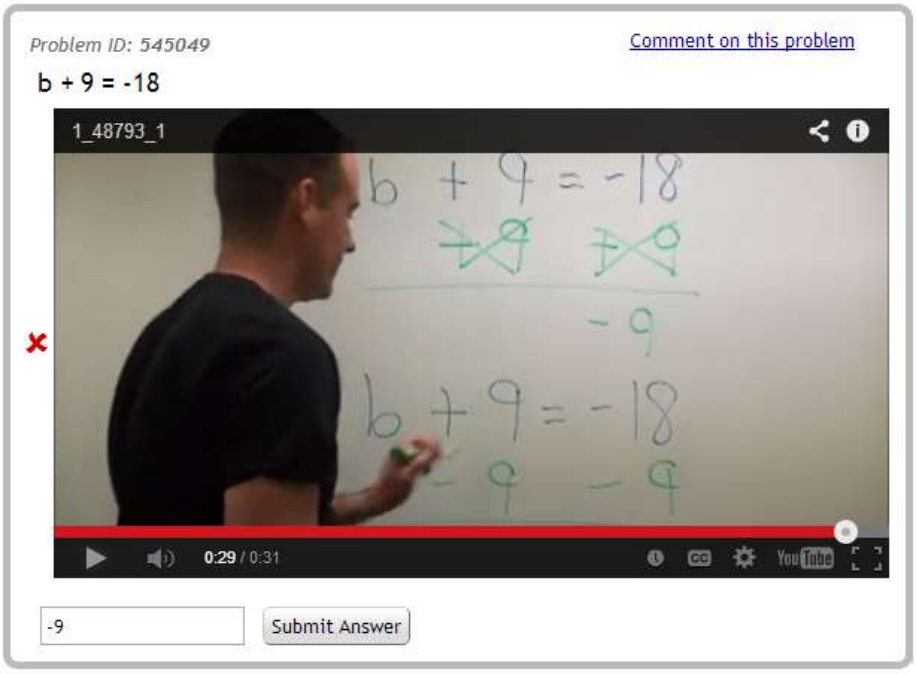


Figure 5.4: Example Problem - Experiment Group Showing a Video Buggy Message

In this study, four similar types of problems related to solving one step equations were used. Both the control and experiment conditions consisted of three different sequences of problems where a student was randomly assigned one of the six sequences of problems. This was done so that students would unlikely be on the same problem at the same time to prevent possible cheating. Each sequence of problems only has the first twelve problems related to the study. After the first twelve problems, all problem sequences, regardless of condition, receive the same type of problems as the control condition. One reason why the problem set was created this way is because of the time it takes to create the videos. Each problem consists of 2-3 common incorrect answers, resulting in a total of twenty-three videos for the twelve problems. Another reason only twelve problems were used is because prior skill builder data shows that it is rare that students ever exceed this number of problems. Table 5.2 shows each of the first twelve problems that have videos. Table 5.3 shows the structure of the problem set, where the order of the problems is shown for each possible sequence as well as the type of the problem designated by the letter after the problem number. Students were randomly assigned to one of the six problem sequences.

Problem Number	Problem Type	Problem	Correct Answer
1	Addition	b + 9 = -18	-27
2	Subtraction	c – 2 = -18	-16
3	Multiplication	-6x = 54	-9
4	Division	y / 8 = -9	-72
5	Addition	b + 11 = -12	-23
6	Subtraction	c – 4 = -20	-16
7	Multiplication	-10a = 50	-5
8	Division	x / -7 = -1	7
9	Addition	x + 3 = -13	-16
10	Subtraction	y – 9 = -14	-5
11	Multiplication	-5b = 40	-8
12	Division	a / 6 = -8	-48

Table 5.2: List of problems

Problem Set						
Control				Experiment		
Sequence 1	Sequence 2	Sequence 3		Sequence 1	Sequence 2	Sequence 3
1	9	5		1	9	5
2	10	6		2	10	6
3	11	7		3	11	7
4	12	8		4	12	8
5	1	9		5	1	9
6	2	10		6	2	10
7	3	11		7	3	11
8	4	12		8	4	12
9	5	1		9	5	1
10	6	2		10	6	2
11	7	3		11	7	3
12	8	4		12	8	4

Table 5.3: Problem Set Structure. This table shows the structure of the problem set. A student is randomly put into one of the six possible sequences of problems. The numbers in the cells correspond to the problem numbers students will receive I the order shown in the table.

Table 5.4 shows the participation of unique students for each of the studies. It is broken into three sections for the initial number of students, students removed from the analysis, and the students remaining for the analysis. Students who answers three problems correctly in a row and completed the problem set without any tutoring were removed. Since these students did not receive any tutoring, they did not experience the conditions of the experiment. A χ^2 of goodness fit test was used to ensure the number of students was not significantly different between the two groups for the initial condition assignment and that the number of students removed in each group was not significantly different. Significance values of 0.12 and 0.80 for studies 1 and 2 respectively indicated that the initial random assignment was not significantly different ($p > 0.05$) between the two groups in each study. Significance values of 0.92 and 0.93 for studies 1 and 2 respectively indicated that the percentage of students removed was not significantly different ($p > 0.05$) between the two groups in each study.

		Study 1 Text Buggy Messages	**Study 2 Video Buggy Messages**
Initial	**Control**	228	694
	Experiment	262	685
	Total	490	1379
Removed	**Control**	124	366
	Experiment	144	364
	Total	268	730
Remaining	**Control**	104	328
	Experiment	118	321
	Total	222	649

Table 5.4. Number of Students in Studies

The following statistics were used to measure the helpfulness of the tutoring.

1. The number of students that finish the problem set
2. Number of problems it takes a student to complete the problem set
3. Correctness on the next problem following a student's first incorrect response
4. Attempts on the next problem following a student's first incorrect response
5. Hints on the next problem following a student's first incorrect response

The first measure is the number of students that were able to answer three problems correctly in a row and finish the problem set. All other measures only include students who have finished the problem set. The second measure is the number of problems it took a student to complete the problem set. The last three statistic show how well students do on the next problem after their first incorrect response. After the student's first incorrect answer, the correctness on the next problem, attempts on the next problem, and number of hints used on the next problem are calculated for all the students. Hints were only used for study 1, because the experiment group in study 2 did not have the option to click on the hint button. Only the problem after the student's first incorrect answer is used because if problems after other incorrect answers were used, credit could not be attributed to a hint or buggy message since the student could have seen both prior to their second incorrect response. Overall correctness is not used as a measure, since the problem set ends when a student gets three correct in a row and not after a static number of problems. In addition to the previous statistics, the overall "hit rate" for the buggy messages was 58% and 77% for study 1 and 2 respectively. This includes the control group which did not see a buggy message, but would have seen one if it was the experiment group. All the data associated with the analysis can be found at[6].

Table 4.5 shows the completion rate for each condition in both studies. A χ^2 test of independence was used to determine if there were any significant differences in completion rates between the control group and the experiment group in both studies. Study 1 had a completion percentage significance value of 0.91 and study 2 had significance value of 0.45 indicating that there were no significant differences in completion rate between conditions for both studies.

		Students Completed	**Total Students**	**Completion Percentage**
Study 1 Text Buggy Messages	**Control**	67	104	64%
	Experiment	80	118	68%
Study 2 Video Buggy Messages	**Control**	289	328	88%
	Experiment	275	321	86%

Table 5.5: Completion Percentage

[6] https://sites.google.com/site/assistmentsdata/projects/selent2015

A two-tailed t-test was used to determine if there were any significant differences for the number of problems it took students to complete the problem set. In study 1, students in the control group took an average of 6.0 problems to complete the problem set and students in the experiment group took an average of 5.8 problems to complete the problem set. These averages were not significantly different ($p = 0.69$, $n_1 = 67$, $n_2 = 80$). Study 2 also showed a similar trend where students took an average of 7.3 and 6.9 problems to complete the problem set for the control and experiment group respectively. This was also not significantly different ($p = 0.30$, $n_1 = 289$, $n_2 = 275$).

Table 5.6 shows the results for next question correctness for both studies. A MANOVA was done using SPSS to see if there was a significant different between the control and experiment conditions on the next question after a student's first incorrect response. For the first study a MANOVA was run with condition as the only factor and correctness, attempts, and hints as dependent measures. The results show no main effect with Wilks' $\lambda = 0.98$, $F(3, 143) = 0.989$, and $p = 0.4$. For the second study the same analysis was done, except hint use was excluded as a measure since the experiment group did not have the option to ask for hints. The results show no main effect with Wilks' $\lambda = 0.991$, $F(2, 561) = 2.674$, and $p = 0.07$.

		Correctness	**Attempts**	**Hints**
Study 1 Text Buggy Messages	**Control**	79%	1.42	0.16
	Experiment	74%	1.51	0.09
Study 2 Video Buggy Messages	**Control**	71%	1.43	0.15
	Experiment	71%	1.65	0

Table 5.6: Next question summary statistics after the first incorrect response

Conclusions and Future Work

In this chapter two studies were run in the ASSISTments tutoring system using tutoring with different types of buggy messages. Although both studies resulted in a null result, we choose to report on them for the following reasons. Despite the large amount of research done on error analysis, there were few interventions done using buggy messages and none done at the scale in this chapter. We show that neither color-coded text messages nor short videos were effective. It

is future work to focus less on how or why a student arrived at a specific wrong answer and focus more other methods of tutoring to communicate better to the students.

Since all three prior randomized controlled trials failed to increase student learning it was acknowledged that other people are needed in order to create effective tutoring. Currently a system called PeerASSIST is being built in the ASSISTments online tutoring system to crowdsource various forms of tutoring from teachers and students. The goal is to collect a large amount of diverse forms of tutoring to be distributed to students semi-real-time via a multi-armed bandit algorithm. These forms of tutoring can include on-demand hints and buggy message in the form of text or video. An evaluation system using a student rating system as well as statistical data analysis will be used to determine if/which tutoring is effective. Once certain tutoring is determined to be effective, the properties that make it effective can be extracted and used in the creation of future tutoring.

Chapter 6: Exploring the Predictive Power of Common Misconceptions with Tabulation Methods

Abstract

In this chapter, multiple tabling models to predict student correctness on the next problem are created from features focusing on the commonality of student misconceptions. These models were tested on two large datasets (open-response questions and multiple-choice questions) consisting of several hundred thousand rows of data on mathematical problems from the ASSISTments tutoring system. These models were compared to Knowledge Tracing and Performance Factor Analysis, which are existing standards for predicting student performance. We showed that our simplest tabling model predicted as well as the standard models for open-response questions indicating that the commonality of incorrect answers can potentially be a useful predictive factor. We also showed that there was no difference between the complexity of the tabling models with either open-response or multiple-choice questions. Although our models performed as well as KT and PFA, all models did poorly when compared to a simple baseline. Further research was done to determine the root problem of why our models did not predict well with a large dataset.

Introduction

Student incorrect responses were first studied in 1978 by John Seely Brown and Richard R. Burton (Brown & Burton, 1978). They manually analyzed student incorrect responses for multi-digit subtraction problems. These incorrect responses were then attributed to an incorrect process or "bug" performed by the student. From this analysis they created a procedural network, where a skill is broken down into sub-skills. Their main goal was to construct a database of all the possible buggy procedures with their series of "BUGGY" programs that helped to identify bugs (Burton, 1982). This work was extended by Brown and VanLehn with their introduction of "Repair Theory" (Brown & VanLehn, 1980). The idea of Repair Theory is that when a student realizes that he/she has performed a buggy operation he/she will attempt to repair the bug. This work looked at how bugs are caused and what bugs can be predicted. Several researchers explored the area of finding common incorrect processes 1990's summarized in (Sison & Masamichi, 1998). Various machine learning algorithms in different systems

attempted to predict student misconceptions and react appropriately. More recent algorithms have been made to automatically discover these buggy rules. One of the most recent examples is a machine learning algorithm developed in (Selent & Heffernan, 2014). In their work, a machine-learning algorithm was developed to automatically discover all buggy rules for a given set of problems generated by the same template. This algorithm works by first taking the input symbols of the problem (the numbers in the problem), a set of basic operations (addition, subtraction, multiplication, and division), and all the student incorrect responses for all problems generated by a single template. Next, the algorithm derives all possible incorrect processes for all the incorrect responses and generalizes the incorrect response across all problems generated by a given template. Finally the incorrect processes are assigned to the most likely generalized incorrect processes. The output of the algorithm is the machine learned process for the incorrect response (how the student arrived at their incorrect answer), as well as a percentage breakdown for all the incorrect processes. Although the method has a few weaknesses in terms of computation time and computer memory, it is a sufficient solution to the problem of finding bugs.

Despite such a large amount of past work done on predicting student incorrect responses, there is little work done to use these incorrect response to predict future performance or correctness on the next question for open-response questions. The National Council on Measurement in Educational Measurement (NCME) community has come up with methods that weight incorrect responses differently. Work done by DeMars shows that polytomous models (weighting the incorrect responses differently) for multiple-choice questions predict better than dichotomous models (weighting all incorrect responses the same) (DeMars, 2008). Our contribution in this chapter is to use the idea of weighting incorrect responses differently and apply it to open-response questions. DeMars notes that, "The polytomous models, which weighted the distractors differentially, yielded small increases in reliability compared to their dichotomous counterparts.". We believe there is value to this and hope to find larger improvements with open-response questions as opposed to the multiple-choice questions DeMars used.

It is expected that knowing the student's common wrong answers will provide more predictive power, especially for open-response questions. These expectations are based on some relevant prior work done by Yigal Attali (Attali, 2014). In his work a randomized controlled trial was run to see what types of practice and feedback on the practice GRE test leads to better performance

on the real GRE test. It was shown that practice tests with open-response questions and immediate correctness lead to better GRE performance than practice with multiple-choice questions. This suggests that student attempts help them learn more and that multiple-choice causes some students to guess more and use less effort.

The goal of this chapter is to see how well we can predict student performance on the next question by knowing the commonality of their previous incorrect response. In this chapter we look at three questions. How well can we predict student performance on the next question given that (1) we know what the most common wrong answer is for open-response questions, (2) we know the top few common wrong answers for open-response questions, and (3) we know all the possible incorrect answers for multiple-choice questions. We want to answer the question; do common wrong answers on open-response questions provide more information than common wrong answers on multiple choice questions? If this is true, then we also want to answer the question, Can we better predict student performance on the next question given that we know what common wrong answers on previous question?

To answer these questions we use two datasets, described in the data section, and build a tabling model for each dataset that considers the commonality of the incorrect answers, described in the model section. We compare the predictive accuracy of our tabling models to the Knowledge Tracing (KT) model (Corbett & Anderson, 1994) and the Performance Factor Analysis (PFA) model (Pavlik, Cen, & Koedinger, 2009). Both KT and PFA are current standards in the field for predicting student performance. Knowledge tracing works by creating a Bayesian network with a latent student knowledge node (whether the student knows the concept, K) and an observable student performance node (whether the student answered the problem correctly, Q) for each time slice. The previous knowledge node transitions to the next knowledge node, creating the Bayesian network shown in Figure 6.1.

Performance Factor Analysis works by creating a regression model, shown in Equation 6.1, that uses prior successes and failures for each student/skill pair and learns a skill difficulty parameter (β) as well as parameters (γ) and ρ for the benefit of prior successes and failures. s and f respectively represent the number of successes and failures the student has had prior to seeing the current problem. m is the logit value for the accumulated learning for a given student at the time the student encounters the current knowledge. Equation 6.2 is the logistic function that returns the student's observed prediction.

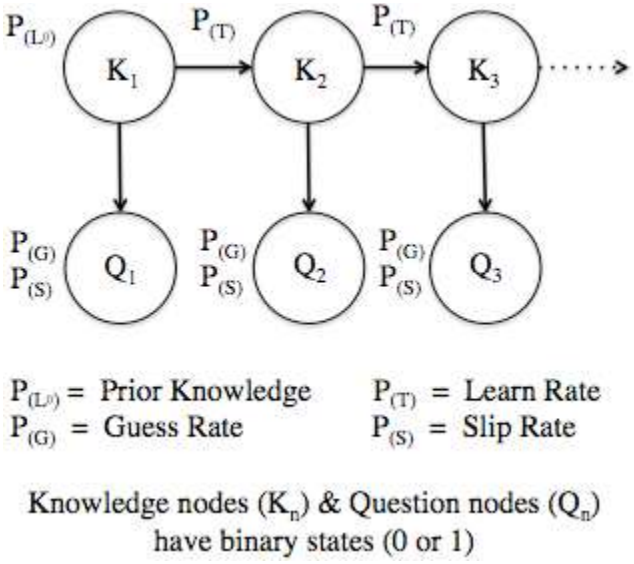


Figure 6.1: Standard Bayesian Knowledge Tracing model (Corbett & Anderson, 1994)

Equation 6.1. $m(i,j \in KCs, s, f) = \sum_{j \in KCs}(\beta_j + \gamma_j S_{i,j} + \rho_j f_{i,j})$

Equation 6.2. $p(m) = 1/(1 + e^{-m})$

Tabling Models

A tabling model provides a mapping from data to predictions based on the attributes of the data. This generates a probability table to use for predicting. Tabling has been applied in past research in (Wang et al, 2011). Their tabling method provided a complement to the Knowledge Tracing model by using past response sequences to predict future responses. Our tabling method expands on that idea by looking at what types of incorrect responses were generated in addition to basic correctness. There are three tabling method tested in this chapter. Each method incrementally increases in complexity compared to the previous model.

The first tabling method is built using four values calculated from the training set: 1) the average next problem correctness probability for all correct answers; 2) the average next problem correctness probability of all cases where the students asked for hints; 3) the average next problem correctness probability of all common wrong answers; 4) the average next problem correctness probability of all uncommon wrong answers. When selecting common wrong answers from a given problem, we pick the answers that cover more than 4 data entries and more than 10 percent of all wrong answer entries under the given problem. All other wrong answers

fall into the uncommon wrong answer category. When a student asks for a hint immediately, it is recorded as a blank response because their first response was to ask for a hint without typing any input in. We are focusing on incorrect answers and a blank answer (asking for a hint) is a special case that we consider.

By binning the student responses into four categories; correct entry, hint entry, common wrong answer entry, and uncommon wrong answer entry, we may use these four values to predict the students' next problem correctness. In doing so we hope to create a rudimentary tabling model that considers the commonality of student incorrect responses to compare to the other models. We also use this simple tabling model to compare to our other tabling models to determine which added granularities contribute to better predictions.

The second tabling method adds more complexity to the prediction table where each common wrong answer and each problem will have an individual probability. This is different from our first model, which groups all the common incorrect responses together for a single probability value. Correct responses and hint use are modeled in the same way as the first tabling method where the probabilities are calculated using the entire training set. In making our predictions we prefer using only the good common wrong answers - common wrong answers that are significantly more predictive than uncommon wrong answers. In selecting the good common wrong answers, we picked the common wrong answers whose prediction values differ from the average next problem correctness probability of all uncommon wrong answers by at least 0.3. We use this table method to compare to our first tabling method to see if the specific common incorrect responses give more predictive power than treating all common incorrect responses the same.

The third tabling method expands on the second tabling method. A probability value is calculated for correct responses and hint use for each problem in order to predict correctness on the next question. This is different from models 1 and 2, which use global probabilities for correct responses and hint use. By implementing this table we hope to see if the same approach used in the second method works with correct answers and situations where students asked for hints.

An example of each tabling method is shown in Tables 6.1 - 6.3. Only a table for a single problem is shown where the entire model learns the probability tables for each problem by calculated all answers of the given type in the entire training set.

Answer Type	Prediction
Correct	0.76775
Hint	0.43836
Common wrong answers	0.53248
Uncommon wrong answers	0.52669

Table 6.1: An example of what tabling model 1 looks like.

Problem ID	Answer	Prediction
71367	Correct	0.76775
71367	Hint	0.43836
71367	21/4 = 5/c	1
71367	Uncommon	0.707317
71360	Correct	0.76775

Table 6.2: An example of what tabling method 2 looks like. Common/uncommon wrong answers are used, where we consider the common wrong answer text as shown as "21/4 = 5/c" in this example. A probability of 1 is used for students who make this common wrong answer, meaning that every student in the training set got the next question correct after making this mistake. Note that we are still using the same values from tabling model 1 in predicting Correct and Hint entries, as presented in the last row the Correct prediction values are the same

Problem ID	Answer	Prediction
71367	Correct	0.931034
71367	Hint	0.321429
71367	21/4 = 5/c	1
71367	Uncommon	0.707317

Table 6.3: Table 6.3 takes one step further from Table 6.2, giving each problem its individual Correct and Hint predictions.

Data

For our experiments we use two skill builder datasets of responses on mathematical questions, one for open-response questions and one for multiple-choice questions. The data ranges from years 2008-2014 and grades 4-12 from the ASSISTments online tutoring system. A skill builder is a set of problems where a student must get a certain number of problems correct in a row (usually three) in order to complete. A skill builder typically consists of a bank of 50-100 possible questions that are randomly drawn from and given to the student. These questions are generated by one or more templates. For example, a template problem for the skill "Order of Operation", looks like "a + b * c". This template generates the problem instances "1 + 2 * 3", and "5 + 4 * 9" as well as several other similar problems by substituting different numbers in for the variables 'a', 'b', and 'c' (Razzaq et al., 2009). The reason why we chose to only focus on data from skill builders is because this data is more realistic and less subject to noise, which is a common problem in online tutoring systems. Skill builders are used by several teachers and are not specific to a certain demographic. Due to the randomized questions, skill builders are also less subject to cheating by students. Students are given correctness feedback, on whether they got the problem right or wrong, after submitting an answer for a problem in.

We continued to refine our datasets based on a few requirements of the models we train and test on. To give all models a fair chance we needed a single data set where all the models can be trained equally on. Our tabling method focuses on the commonality of incorrect answers and does not require any data on knowledge components, which is analogous to what are called "skills" in the ASSISTments system. This is an advantage of our model, since PFA requires each question to have a knowledge component (skill) associated with it. Therefore we used datasets that PFA can be trained correctly on. We chose to remove all problems that were associated with more than a single skill for simplicity. For the problems that did not have an associated skill, we used the template id the problem was generated by to group those problems into an unnamed "skill".

In order for our datasets to be modeled by KT, we had to limit the number of problems a student attempted for a given skill. If we did not limit the number of problems for a student/skill pair, the Bayesian network for KT would need a latent knowledge node and an observable node for all of the student responses for a student/skill pair. Due to outliers in our original datasets, this results in a Bayesian network of 229 knowledge/observable node pairs, which takes too long to

train on. For this reason, we removed all data where a student had completed more than ten problems for a given skill. To be clear, the first ten problems for a given student/skill were not removed. Only the data after ten problems for a given student/skill was removed. To get more accurate predictions we also removed data for skills we felt did not have sufficient data to train on. Any skill that had less than 10,000 data rows was removed from the dataset.

Our final dataset for open ended questions consisted of 343,024 rows of data with 6,796 unique problems, 32,068 students and 14 skills. Our final dataset for multiple-choice questions consisted of 129,728 rows of data with 1,410 unique problems, 20,870 students and 7 skills. Both datasets are available at the following URL address[7]. The datasets are split up into pieces due to size restrictions.

We note that to have a single dataset to train and test all models equally on, we had to modify the original dataset to compensate for the weaknesses of the other standard models. We consider these weaknesses unquantifiable, but worth considering when analyzing the results.

Analysis

For both datasets we randomly split our data into a training set and test set based on the student ID. All of the student response data is associated with a unique ID for each student. If the student ID is divisible by five, all data of that student are in the test set and the remaining data are part of the training set. Our questions are about predicting student performance on the next question. Therefore all data where there is no next-question, such as the last question for a given student/skill pair, was not predicted on for the test set. This resulted in a training set of consisting of 275,183 (80 %) rows of data and a test set of 67,842 (20 %) rows of data for open-response questions and a training set of 129,728 (84 %) rows of data and a test set of 25,261 (16 %) rows of data for multiple-choice questions.

To analyze our results we use four different metrics to measure the prediction accuracy of the models. We use Accuracy, Root Mean Squared Error (RMSE), Area Under the ROC Curve (AUC), and R^2 to measure the accuracy of the predictions from all the models. We also included the number of parameters each model needed to learn. Our predictions are a range of probabilities from [0, 1], representing the prediction probability that the student will get the next question correct.

[7] https://sites.google.com/site/zlas2015/data

Accuracy is calculated by the number of correct predictions divided by the total number of predictions. To calculate basic accuracy, we use 0.5 as a cutoff point to round for our accuracy metric. If our prediction value is < 0.5 we round to 0 and if our prediction value is > 0.5 we round to 1. Once our prediction is converted to a binary value (0 or 1), we then calculate the absolute difference from the actual value and use that as our accuracy measure. A higher accuracy indicates a better prediction.

The RMSE metric can range anywhere from [0, 1] where a '0' represents a perfect model and a '1' represents the worst possible model. A lower RMSE indicates a better prediction. Since RMSE is not normalized, high values of RMSE may not necessarily be poor predictions.

AUC can range anywhere from [0, 1], where a higher value indicates a better prediction and a value of '1' represents a perfect model. A model should never have a value below 0.5, since that would mean the model is predicting worse than chance.

R^2 measures how much variance is explained by the model or how similar two models are. R^2 is normalized on a [0, 1] scale where an R^2 value of '1' means that the two models are exactly the same. A higher R^2 indicates better predictive accuracy when comparing predictions to the actual values.

ZeroR predictions (predicting the majority class) are provided as a baseline model to compare all the other model metrics to because "good" and "bad" metric values are dependent on the specific datasets

Since a single student attempts multiple problems, the data samples are not independent of each other. For this reason we report both overall metrics as well as metrics calculated per student. For per student metrics, we calculate the metric for each student and then calculate an average of those averages. Since many students attempted a small number of problems (<3), we are unable to calculate AUC and R^2 for each individual student. We used a two-tailed paired t-test on the metrics calculated for each student to determine the significance of the model differences. This cannot be done for overall metrics because of the independence violation.

Results on open-response dataset

	Accuracy	RMSE	AUC	R^2	# of Parameters
ZeroR	66 %	0.5868	0.5	N/A	1
KT	58 %	0.5050	0.5057	<0.001	203,526
PFA	69 %	0.4570	0.6596	0.079	25,066
Tabling 1	66 %	0.4564	0.6482	0.0778	4
Tabling 2	67 %	0.4559	0.6514	0.0799	5024
Tabling 3	67 %	0.4584	0.6740	0.0796	13630

Table 6.4: Overall Prediction Accuracy

	Accuracy	RMSE
ZeroR	70 %	0.4442
KT	59 %	0.479
PFA	72 %	0.4335
Tabling 1	67 %	0.4676
Tabling 2	67 %	0.4684
Tabling 3	67 %	0.4788

Table 6.5: Student Level Prediction Accuracy

Tables 6.4 and 6.5 show the results comparing the three tabling methods to ZeroR, KT, and PFA. Some things to note are that ZeroR predicted all correct responses (‘1’ values) for the majority class and for that reason the R^2 metric could not be calculated for ZeroR. All models in the per-student table were significantly different from each other in terms of RMSE. Significance values were all less than 0.001 likely due to the large size of the dataset. However for accuracy, our tabling models are significantly different than all other models with $p<0.001$ but not significantly different from each other with $p>0.05$ for all tabling models. This suggests that there is little difference between our three tabling methods. We also looked at the number of parameters that were calculated for each model. ZeroR only learns a single parameter (the majority class), where

the number of parameters in the other models are affected by the input size. KT learns parameters for the guess value, slip value, and knowledge value. Each response requires each of those values. Since there are a total of 67,842 responses, there KT model requires 3 x 67,842 = 203,526 parameters on this open-response question dataset. PFA learns a difficulty parameter for each skill, and success and failure rate parameters for each student/skill pair. There are 14 skills and 12,526 student/skill pairs, which results in 14 + 12,526*2 = 25,066 parameters.

The main takeaway from Tables 6.4 and 6.5 is that all three tabling methods performed equally compared to each other, and compared to KT and PFA. This would suggest that merely considering the common wrong answers is enough to get an adequate model. Further complexity did not provide any more predictive power. This would suggest that binning all entries into four categories, correct, hint, common wrong answers, uncommon wrong answers, is enough to get a good model. It also shows that learning a smaller number of parameters is advantageous for both model training time and model accuracy. ZeroR even performed better than KT and almost as well as PFA, which means that all the models are not performing very well.

We investigated why the three tabling models producing similar results is that the common wrong answers cover only as little as 4% of all data entries. For the open-response data set, the averaged common wrong answer next problem correctness is 0.53248, while the averaged uncommon wrong answer next problem correctness is 0.52669. The two values do not differ much from each other. Only the good common wrong answers, common wrong answers with next problem correctness significantly higher or lower than the averaged uncommon wrong answer next problem correctness, may improve our prediction accuracy.

Results multiple-choice dataset

Tables 6.6 and 6.7 show the results on the multiple choice dataset. Overall all the models gave better predictions for the multiple-choice dataset than the open-response dataset. However compared to a baseline of ZeroR, our tabling method did significantly worse for student level accuracy and RMSE ($p<0.001$). For these reasons we are unable to show that open-response questions provide more predictive power than multiple-choice questions and were unable to replicate results of DeMars.

	Accuracy	RMSE	AUC	R^2	# of Parameters
ZeroR	75 %	0.5018	0.5	N/A	1
KT	77 %	0.3973	0.7584	0.1628	75,783
PFA	73 %	0.4376	0.5752	0.0068	12,767
Tabling 1	75 %	0.4249	0.621	0.043	4
Tabling 2	75 %	0.4266	0.6164	0.0378	1364
Tabling 3	75 %	0.4194	0.6863	0.0797	3834

Table 6.6: Overall Prediction Accuracy

	Accuracy	RMSE
ZeroR	82 %	0.2662
KT	83 %	0.29
PFA	82 %	0.35
Tabling 1	76 %	0.3695
Tabling 2	75 %	0.3693
Tabling 3	76 %	0.3756

Table 6.7: Student Level Prediction Accuracy

Future Work

In our experiments we used the common incorrect responses on individual problems to predict next question correctness. We think there might be more to gain by generalizing our method to predicting generic incorrect processes that can be applied to multiple problems at the template level and not just the problem level. One reason for this is because our model requires several common wrong answers for each problem to benefit the most. There are thousands of problems, but typically a smaller amount of data for each problem. It is future work to see if we can apply our algorithm to more general levels, such as using common wrong processes for a template that can apply to 50-100 individual problems. This will allow our algorithm to be applicable in a more general context, rather than being limited to large datasets not always available.

Conclusion

We set out to investigate the possibility of improving our ability to predict student performance depending on whether they provided a common wrong answer to a previously related question. This was inspired by work done by DeMars, who showed that polytomous models predict performance better than dichotomous models on multiple-choice questions, and Attali, who showed in a randomized controlled trial that open-response questions lead to greater learning than multiple-choice questions.

We created three different tabling models where one model expanded upon the previous model and increased in complexity. This allowed us to test different granularity levels for commonality of incorrect responses. In addition we compared our model to current standard models in the field of predicting student performance for two different datasets. By choosing a dataset consisting of only open-response questions and a dataset consisting of only multiple choice questions, we could try to reproduce the results of DeMars and Attali.

Our results showed that the complexity of the tabling did not have any significant effect, where each model predicted roughly the same and our models did no better than a baseline model. Unfortunately we were unable to answer our initial questions of whether or not knowing the commonality of incorrect responses can lead to better predicting models, and whether or not common wrong answers on open-response questions provide more information than common wrong answers on multiple-choice. Upon deeper investigation on why our models performed poorly, we believe it is because there were too few responses for a given problem. Due to the large number of problems and the low number of incorrect responses, there were not many data points to train from for a given incorrect response on a problem.

Chapter 7: Applying and Exploring Bayesian Hypothesis Testing for Large Scale Experimentation in Online Tutoring Systems

Abstract

This chapter demonstrates the viability of using Bayesian hypothesis testing for statistical analysis of experiments run in online learning systems. An empirical Bayesian method for learning a genuine prior from past historical experiment data is applied to a dataset consisting of twenty-two randomized controlled A/B experiments collected from the ASSISTments online learning platform. Using this data a prior probability is learned for an experiment having differences between the conditions of the experiment. The hyperparameters of this process are explored to determine both their stability and reliability. Confidence interval estimates are supplied to determine the accuracy of this learned genuine prior on the actual dataset and in hypothetical scenarios under the same distribution. We show that using only twenty-two experiments results in a learned genuine prior with poor confidence interval estimates, and that roughly 200 experiments are required for a reasonable estimate of the true probability of an experiment having differences between experiment groups. A leave-out-experiment-out cross-validation experiment is conducted, where a genuine prior is learned from twenty-one of the randomized controlled experiments provided in the dataset and then used to evaluate the remaining experiment. From this experiment we show that Bayesian hypothesis testing performs similar to Frequentist hypothesis testing and both methods were in agreement.

Introduction

Within the last decade, online experimentation is becoming more popular in the area of learning sciences and educational research. Several intelligent tutoring systems and online learning platforms are now running randomized controlled experiments on a regular basis. A few of these systems include ASSISTments, the Cognitive Tutor, and the Andes physics tutor. ASSISTments is an online learning platform developed at Worcester Polytechnic Institute, which is largely focused on K-12 mathematics (Heffernan & Heffernan, 2014; Razzaq et al., 2005; Roschelle et al., 2016). The Cognitive Tutor is an intelligent tutoring system developed at Carnegie Mellon,

which is also focused on mathematics for grades 6-12 (Anderson et al., 2005; Koedinger & Corbett, 2006). The Andes Physics Tutor is an intelligent tutoring system, developed at Arizona State University and the University of Pittsburgh, which is used for introductory college physics classes (Gertner & VanLehn, 2000). All these systems have shown a large amount of success for improving student learning.

A common reason for the growing popularity of these systems is grounded in research done by Bloom (Bloom, 1984). Bloom showed that a small student-teacher ratio has a large impact on the effect size for improving student learning (two standard deviations). This is now commonly known as "Bloom's 2 Sigma Problem". Many online tutoring systems attempt to achieve the same result as Bloom with a reduced cost, by having the computer tutor mimic the functionality of a one-to-one tutoring scenario with a teacher/tutor.

As a result of the increased popularity of online tutoring systems, there are a growing number of randomized controlled experiments being run in these systems. Online experimentation at large scale is an area initially studied by various web companies such as Google, Microsoft, Facebook and others (Kohavi et al., 2013). Many of these companies are moving toward using Bayesian hypothesis testing (as opposed to the traditional Frequentist null hypothesis statistical testing) to analyze the results of their experiments. For more information on the differences between Bayesian hypothesis testing and Frequentist null hypothesis testing see (Efron, 2013).

There are several advantages to using Bayesian hypothesis testing over Frequentist hypothesis testing. Kruschke even argues that Bayesian methods are superior to Frequentist methods (Kruschke, 2013). One advantage of using Bayesian hypothesis testing is that the null hypothesis can be accepted given enough data. This is contrary to Frequentist statistics where we can only fail to reject the null hypothesis. As a result, a large number of scientific experiments are not reported or not accepted because of the lack of ground-breaking findings. Typically most experiments do not result in a conclusive (positive/negative) result. Kohavi discusses in some cases up to 90% of experiments do not result in any changes in companies such as Google and Netflix (Kohavi et al., 2013). This contributes to the "File Drawer" problem, where a large number of null results fail to be published due to publication bias, which favors positive results (Rosenthal, 1979). Consequently, other researchers may run the same experiments in vain without knowing about the previous null result.

Using Bayesian statistics will help alleviate the reproducibility problem, where the reported results of many research findings cannot be reproduced. Ioannidis has demonstrated this in the medical field by analyzing 49 clinical research studies reported in the top three medical journals. He has shown that out of the 49 studies, 45 reported that the intervention was effective. Out of these 45 interventions, 44% were replicated and 32% were either contradicted or had found effects larger than the effects found in the replication experiments (Ioannidis, 2005A). There was no attempt to replicate the remaining 24%.

Ioannidis has stated that one of several reasons for the lack of reproducibility is because of a low prior probability of the research findings being true (Ioannidis, 2005B). Standard statistical methods do not consider this piece of information, which results in a large number of false positive (Type-I error) results. This concept was nicely and humorously illustrated in an online comic (Munroe, N.D.). The comic describes a machine to detect if the sun has gone nova. The machine works by rolling two dice and returns a positive result if the values of both dice are equal to six. If the machine reports a positive result, the researcher can conclude with $p < 0.05$ that the sun has gone nova because the chance of a positive result by chance is $1/36 = 0.027$. This comic illustrates the need to consider the prior probability of a positive result in order to prevent reporting false positive results.

Another advantage of Bayesian hypothesis testing is the ability to run experiments with sequential experiment design that supports multiple hypothesis testing without inflating Type-I error and also supports optional stopping (the ability to stop an experiment at any time based on the current information at that time). The landmark paper on Sequential design was first published by Herbert Robbins in 1952 (Robbins, 1952). Sequential design is when subjects are assigned into conditions based on data from prior subject-condition assignments. This is different from the standard A/B experiment where all subjects are assigned into conditions before the experiment is run. One advantage of sequential design is to limit the number of subjects assigned into an inferior and potentially harmful experimental condition (Wegscheider, 1998). Although this is more common in the medical field with potentially dangerous medical treatments, we believe that exposing students to inferior or even harmful treatments is a major issue that has been relatively undiscussed in the context of educational research. It is also possible to expose an unnecessary number of students to the control group of an experiment,

when the treatment is found to be effective. It is presumed that may researchers may not attempt to publish null and especially negative results because of the potential consequences.

In order to gain these various Bayesian advantages, a genuine prior must be known or learned from data. This genuine prior represents the prior probability of an event being true and is required to perform Bayesian statistical analysis. For example, a coin flip has a prior probability of 0.5 for the coin to land on heads. In this example the prior is known, however for online controlled experiments the prior is not known. Those who are using Bayesian hypothesis testing for online controlled experiments are taking advantage of the fact that there have been thousands of previously run experiments to accurately learn a prior from those previously run experiments.

In the context of this work, the genuine prior will represent the prior probability of an A/B experiment having a difference between the experiment group and the control group of an experiment. With the increasing amount of randomized controlled experiments run in online tutoring systems, we believe that we can start to apply techniques initially used by Google, Microsoft, and other large web to learn a genuine prior for randomized controlled A/B tests in the online learning context. With this genuine prior that is learned from prior experiments, we can apply Bayesian hypothesis testing to various online tutoring systems and benefit from the advantages.

This work demonstrates the viability and advantages of applying Bayesian hypothesis testing to analyze randomized controlled experiments being run in online tutoring systems. First we first apply the methodology described by Alex Deng to learn a Bayesian genuine prior from an unbiased collection of experiment data on twenty-two randomized controlled A/B tests obtained from the ASSISTments online learning platform (Deng, 2015; Selent & Patikorn, 2016). We provide an example of how this can be done and report on the learned genuine prior. In addition to learning the prior, we provide confidence intervals for the learned prior and show how many experiments would be necessary, under the same distribution, to learn a prior with reasonable confidence intervals.

We then run a separate experiment, where we learn the genuine prior in a leave-one-experiment-out cross-validation format. This means that a prior is learned from twenty-one experiments and used to perform Bayesian hypothesis testing on the twenty-second experiment. This is done for

each of the twenty-two experiments, where one experiment is evaluated using the genuine prior learned from the other twenty-one experiments. We compare the results from the Bayesian hypothesis testing methodology to the results of Frequentist hypothesis testing methodology.

Data Description

The Dataset we use in our experiments comes from 22 randomized controlled A/B tests run inside the ASSISTments online learning platform (Patikorn et al., 2017; Patikorn et al., 2016; Selent & Patikorn, 2016). All of the experiments were created by either internal or external researchers working with ASSISTments. Several of these experiments have been previously published with topics ranging from student confidence, student choice, and buggy messages (Lang et al., 2015; Ostrow, 2015; Selent & Heffernan, 2015).

There are two major characteristics of this dataset that make it ideal for conducting our experiments. Firstly all of the 22 experiments are in a canonical format with the same dependent measure. All experiments were math assignments with a control group and an experiment group. Students were randomly assigned into one of the two groups. Students continued to receive problems inside of the ASSISTments tutoring system until they had reached mastery of the content. Mastery occurs after a student has answered *n* problems correctly in a row, where *n* is typically set to three. Master speed is the total number of problems a student attempted before reaching mastery and completing the assignment (Xiong et al., 2013). The logarithm base ten of Mastery Speed is used as the dependent measure in this dataset to reduce the effect of outliers on the mean.

Secondly these experiments represent an unbiased collection of experiments with positive, negative, and null results. Due to publication bias, it is hard to obtain such a dataset as positive results are reported more often than negative or null results. These two characteristics are important in order to learn a genuine prior, which requires an unbiased sample estimate of the population of experiment outcomes.

Learning a Genuine Prior

We implemented the method described by Alex Deng to learn a genuine prior from the dataset (Deng, 2015). The method is summarized below. First we define the following equations. The effective sample size for a given experiment is defined in Equation 7.1, where N_T represents the number of students in assigned into the treatment group and N_C represents the number of students assigned into the control group.

Equation 7.1. $$N_e = \frac{1}{(\frac{1}{N_T} + \frac{1}{N_C})}$$

Let δ be the effect size for an experiment measured with Cohen's D. Equation 7.2 defines how the effect size is calculated. In Equation 7.2, μ_T is the mean of the treatment group, μ_C is the mean of the control group, and *s* is the pooled standard deviation.

Equation 7.2. $$\delta = \frac{\mu_T - \mu_C}{s}$$

The pooled standard deviation is defined in Equation 7.3, where σ_T and σ_C are the standard deviation of treatment and control groups respectively.

Equation 7.3. $$s = \sqrt{\frac{(N_T - 1)\sigma_T^2 + (N_C - 1)\sigma_C^2}{(N_T + N_C)}}$$

In the two group model, let H_0 be the null hypothesis and H_1 be the alternate hypothesis. In Bayesian statistics the posterior probability is the probability of an event occurring given evidence. In our context the posterior probability is the probability of an experiment having differences in conditions given a set of prior experiments and their effect sizes. Typically this is defined as posterior odds instead of posterior probability; however it is simple to convert from one to another. The posterior odds of δ_i belonging to H_1 against H_0 for i=1,…,N is shown in Equation 7.4. The posterior odds are also represented as the Bayes factor multiplied by the prior odds, which can easily be seen in the equation. Expectation-Maximization (EM) is used to learn a probability *p* and a variance *V* without knowing whether δ_i belongs to H_1 or H_0, by applying Equations 7.4 and 7.5 (Dempster et al., 1977).

Equation 7.4. $$P_i = \frac{\emptyset(\delta_i; 0, \frac{1}{N_{Ei}} + V^2)}{\emptyset(\delta_i; 0, \frac{1}{N_{Ei}})} \times \frac{p}{1-p}$$

In Equation 7.4, $\emptyset(x; \mu, \sigma^2)$ is the normal density with mean μ and variance σ^2. V^2 is the variance of the effect sizes, and p is the probability of an experiment having differences between conditions. Equation 7.5 and 7.6 are used in EM to update the variance. The new variance is calculated from the difference in weighted average of the effect sizes, shown in Equation 7.6. It is bounded away from zero to prevent EM from producing incorrect values as a result of no variance.

Equation 7.5. $$V^2 = WAvg(\delta_i^2; P_i) - WAvg\left(\frac{1}{N_{Ei}}; P_i\right)$$

Equation 7.6. $$V^2 = MAX(V^2, (4 * Avg(\frac{1}{N_e})))$$

Expectation Maximization Algorithm

1. Initialize *p* and V to reasonable values
2. Initialize *p'* to MAX
3. Set *diff* = |*p'* - *p*|
4. Loop until (diff < ε)
    a. Update P_i using Equation 4
    b. Set *p* to AVG(P_i)
    c. Update V using Equation 5
    d. Set *diff* = |*p'* - *p*|
    e. Set *p*' = *p*

The Expectation-Maximization works as follows. Fist an initial prior *p* and variance *V* are chosen. *p*' is initialized to a maximum sentinel value. It represents the probability after the EM iteration, which initially does not have a valid value. This variable is used in combination with diff, to calculate the amount that the probability has changed from before and after the iteration. The absolute value of this difference is compared to ε, to determine if EM has converged. In our experiments, we set ε = 0.0001.

In the loop, the posterior probabilities for each experiment are updating using Equation 4. The prior is set to be the average of the posterior probabilities and the variance is also updated with Equation 5. When the algorithm converges, the posterior probability learned will represent the learned prior probability.

Using the algorithm from Deng summarized above, we ran it on the set of twenty-two randomized controlled experiments. After running EM, the posterior probability learned back was 0.39. This means that there is a 39 % chance of an experiment having differences between conditions. This prior can be used in Bayesian hypothesis testing on future experiments conducted in similar formats.

Since the method to learn prior values for p and V uses a form of EM, one question to ask is whether or not the initial choice of hyperparameters are influential on the final learned values. It is known that the EM algorithm can converge to a local minimum and learn back suboptimal values (Wu, 1983). It has been reported by Baker that the EM algorithm can learn back degenerate parameters for the Knowledge Tracing model when poor initial values are chosen (Baker et al., 2008). Pardos also reports that when poor initial hyperparameters are chosen, EM can learn back a probability of (1-*p*) where *p* is the true probability for guess and slip values in the Knowledge Tracing model (Pardos & Heffernan, 2008). For these reasons, we run EM with different initial probabilities to determine if the same values are learned back for different initial parameter choices. Figure 7.1 shows the results of several executions of EM with different starting prior probabilities.

Figure 7.1 shows that a prior probability of ~0.4 is learned by EM for all initial probability choices. Luckily the results are stable for the distribution of data used in this experiment. Regardless of the initial prior probability chosen, all posterior probabilities learned back are stable at roughly 0.4. The only influence the choice of an initial prior probability has is on the number of iterations it takes EM to converge. Note that although EM gives stable results in the context of our data, it may not always give stable results. It is recommended to test the stability of the learned parameters when using EM.

Knowing that our initial choice of prior probability does not affect the learned value means we do not have to be as concerned about our initial choices. For all subsequent experiments, we fix

our initial hyperprior to 0.2 as a somewhat arbitrary choice for what we believe is a reasonable value. We do not choose 0.4 because we do not want to "cheat" by using the value learned by EM as input to EM in other experiments (although it will not make a difference).

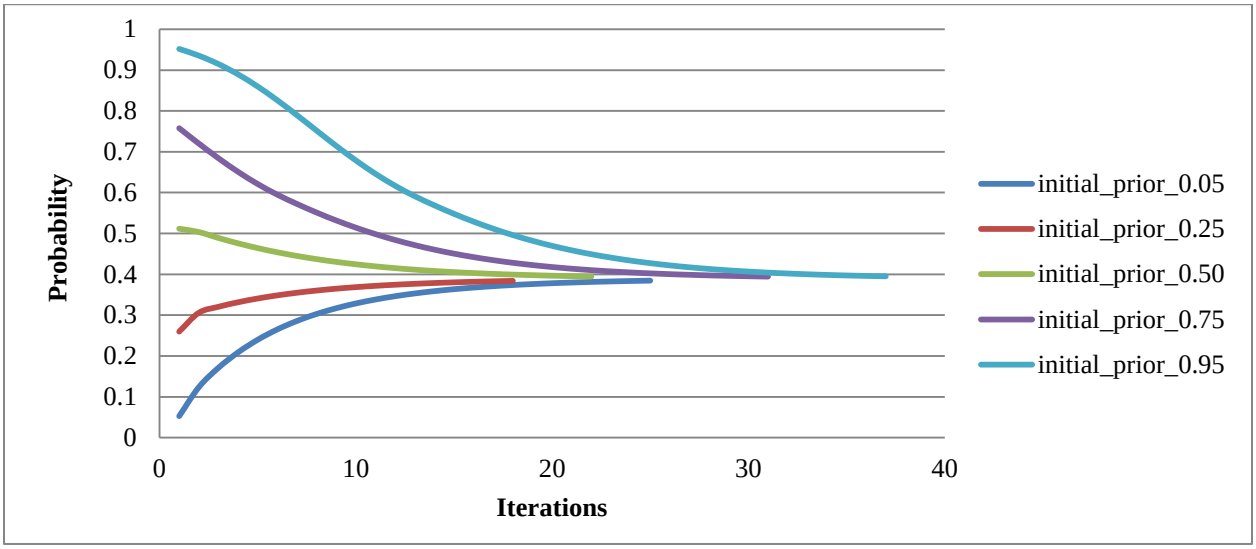


Figure 7.1. This figure shows the results of running EM with different prior probabilities. Each initial prior probability is a line in the graph. The x-axis represents the iteration number showing the learned probability after each iteration has completed. Once EM has converged there are no more data points for a given line; therefore some lines will end earlier than others.

Before using the learned prior for Bayesian hypothesis testing, another important question to ask is, what is the confidence interval on the learned prior? It is critical to have an accurate prior to perform Bayesian hypothesis testing. To answer this question we extend the methodology proposed by Deng and implement the bootstrap method to calculate confidence intervals for parameters learned by EM proposed by (Basford et al., 1997).

The bootstrap method works by resampling from the original dataset of twenty-two experiments with replacement. Each bootstrap sample is the same size of the original sample. EM is run on the bootstrap sample to learn a prior probability from the data in the bootstrap sample. This is repeated for a large number of times, and each prior probability is stored. Finally, the list of all the probabilities from each bootstrap sample is sorted. We define S as the size of the list, U_i as the index of the upper confidence bound equal to ((1-0.025) x S) and L_i as the index of the lower confidence bound equal to 0.025 x S. The 95 % confidence intervals can be obtained taking the value at Li as the lower confidence bound and the value at U_i as the upper confidence bound.

We apply this bootstrap method to obtain confidence intervals on the twenty-two experiments. We also simulate having a large number of experiments available under the same distribution to see how many experiments are required for reasonable confidence on the learned prior. To do this we vary the list size of the bootstrap sampled. Figure 7.2 shows the confidence intervals for a varying number of experiments. The mean probability learned back is roughly 0.39 for all experiments; however the confidence intervals vary with the number of experiments used in the bootstrap.

Firstly we acknowledge that the confidence intervals for the prior learned on twenty-two experiments are quite poor [0.001, 0.86]. This indicates that it is too optimistic to learn a genuine prior with just twenty-two experiments. How many experiments are required for a reasonable confidence interval is somewhat subjective, however there seems to be a point of diminishing returns at around 200 experiments. Therefore it appears that there would need to be roughly 200 experiments of historical data to learn from.

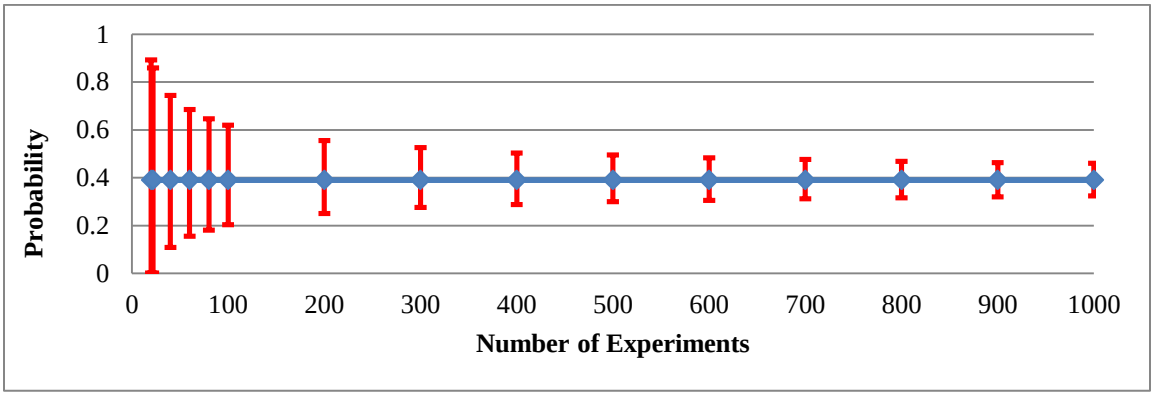


Figure 7.2. Confidence interval estimates on learned probability.

Applying Bayesian Hypothesis Testing

To apply Bayesian hypothesis testing we use leave-one-experiment-out cross-validation. We use twenty-one out of the twenty-two experiments in the dataset to learn a genuine prior using the methodology described in the previous sections. We then use that learned prior to evaluate the last experiment. This repeats until all twenty-two experiments have been evaluated with priors learned from the twenty-one experiments. We compare the results from both the Bayesian and Frequentist methods.

Table 7.1 summarizes the results for each of the twenty-two experiments. For the Frequentist results we report the p-value and the effect size (Cohen's D). The p-value is the chance that the differences between conditions in an experiment are due to chance. Therefore a p-value of 0.05 means that there is only a 5 % chance that the differences in conditions of the experiments are due to chance. This is typically the cut-off point most researchers use when determining if there results are statistically significant. When the p-value is not less than 0.05, we have failed to reject the null hypothesis. We cannot make any claims towards accepting the null hypothesis. Figure 7.3 also shows a graphical representation for the Bayesian probabilities for each of the twenty-two experiment.

In Bayesian statistics there are three probabilities that are reported. P(flat) is the probability of the result being zero, P(positive) is the probability of the result being positive, and P(negative) is the probability of the result being negative (Deng, 2015). In our context P(flat) represents the probability of the treatment having no impact on student learning, P(positive) represents the probability of the treatment having a positive impact on student learning, and P(negative) represents the probability of the treatment having a negative impact on student learning.

In addition to the three probabilities, it is common to report the Bayes Factor. The Bayes factor is the ratio of the posterior odds of the alternate hypothesis being true to the prior odds of the alternate hypothesis being true (Kass & Raftery, 1995). In general, a large Bayes factor indicates strong evidence towards the alternate hypothesis. For example, if the Bayes factor is ten, then this can be interpreted as the data is ten times more likely to occur under the alternate hypothesis. Kass & Raftery give general benchmarks on interpreting the Bayes Factor shown in Table 7.2.

When comparing both the Bayesian and Frequentist methods we look to see if there are any differences and where those differences are. After looking through all the results, there were five experiments (2-4, 10, 13) with reported p-values less than 0.05. Out of these five experiments only two (2, 3) had a P(positive) > 0.95 with the Bayesian method. Those two experiments were the only two experiments with either a P(positive) or P(negative) greater than 0.95. This suggests that the Bayesian method is more conservative than the Frequentist method on this dataset. Without knowing the ground truth values it is not possible to say which method is more accurate.

	Bayesian						Frequentist	
Experiment Number	**Learned Prior**	**Learned Variance**	**Bayes Factor**	**P(positive)**	***P*(flat)**	**P(negative)**	***p***	**Effect Size**
1	0.445	0.199	0.348	0.218	0.782	0.000	0.812	0.017
2	0.362	0.215	4.383	0.713	0.287	0.000	0.016	0.186
3	0.367	0.192	414.230	0.996	0.004	0.000	0.000	0.464
4	0.336	0.200	44.597	0.958	0.042	0.000	0.001	0.290
5	0.359	0.199	0.303	0.000	0.855	0.145	0.606	-0.029
6	0.370	0.200	0.483	0.206	0.779	0.015	0.871	0.018
7	0.378	0.201	0.484	0.000	0.773	0.227	0.745	-0.034
8	0.384	0.201	0.439	0.000	0.785	0.215	0.768	-0.028
9	0.389	0.201	0.468	0.008	0.771	0.222	0.8475	-0.020
10	0.381	0.203	2.491	0.606	0.394	0.000	0.038	0.200
11	0.384	0.204	0.594	0.000	0.730	0.270	0.799	-0.085
12	0.387	0.204	0.528	0.000	0.750	0.250	0.663	-0.050
13	0.382	0.204	3.016	0.651	0.349	0.000	0.029	0.277
14	0.382	0.205	0.962	0.000	0.627	0.373	0.136	-0.120
15	0.382	0.205	0.844	0.000	0.657	0.343	0.2244	-0.135
16	0.382	0.206	0.948	0.370	0.630	0.000	0.261	0.207
17	0.383	0.206	0.762	0.321	0.679	0.000	0.245	0.118
18	0.386	0.206	0.367	0.004	0.813	0.183	0.8727	-0.013
19	0.386	0.206	0.746	0.320	0.680	0.000	0.146	0.088
20	0.390	0.205	0.225	0.082	0.874	0.044	0.985	0.001
21	0.392	0.205	0.471	0.000	0.767	0.233	0.654	-0.044
22	0.389	0.206	1.745	0.000	0.473	0.527	0.0653	-0.187

Table 7.1. The results comparing both Bayesian and Frequentist hypothesis testing. The Bayesian method used an initial prior learned from EM as well as an initial variance. All three probabilities are reported for Bayesian statistics as well as the Bayes factor. The *p* value and effect size (Cohen's D) are reported for the Frequentist statistics.

Bayes Factor	Strength
1 - 3	Not worth more than a bare mention
3 - 20	Positive
2- 150	Strong
> 150	Very Strong

Table 2. Bayes Factor interpretation by Kass & Raftery (Kass & Raftery, 1995).

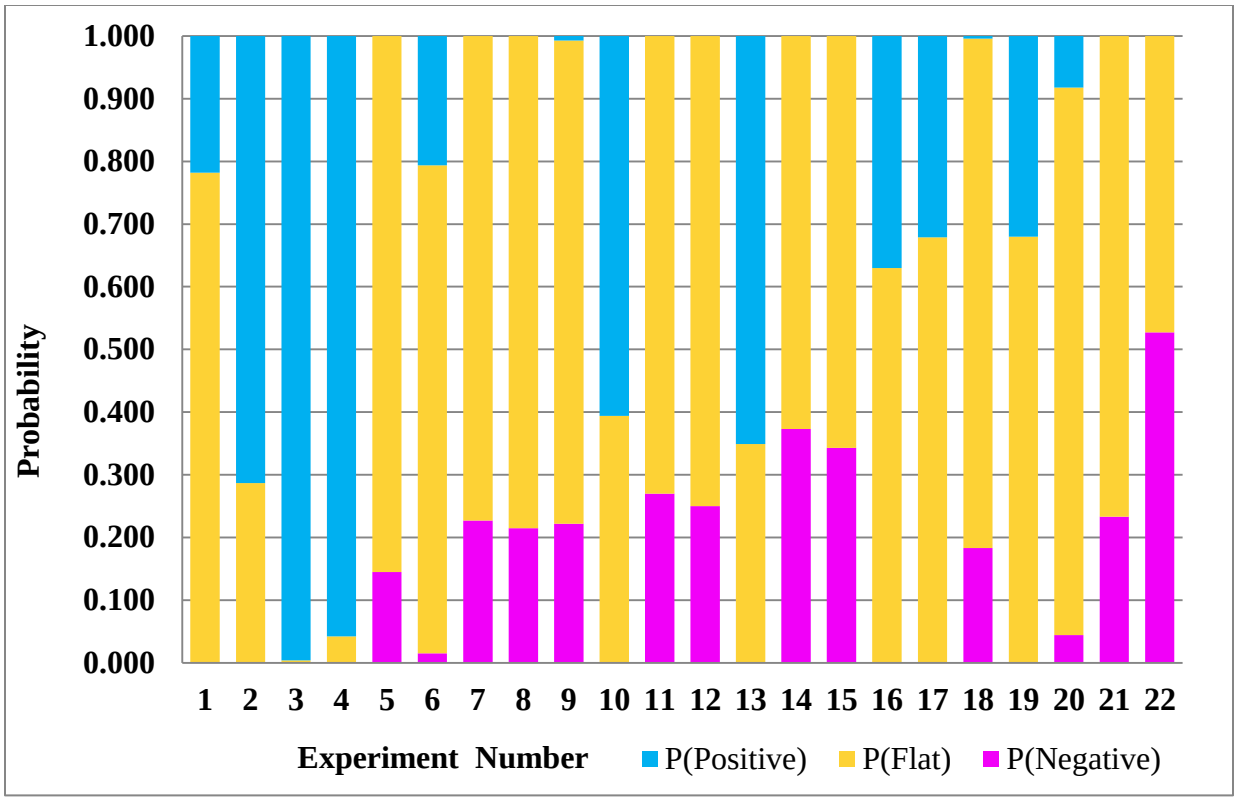


Figure 7.3. A stacked bar graph to show the three probabilities (positive, negative, flat/null) for each experiment.

It is also worth pointing out that although Bayesian methods can accept the null hypothesis, none of the experiments had a P(flat) > 0.95. It is debatable whether or not the 0.95 cut-off point should still be used; however we do so for comparison purpose with the Frequentist methods.

There are several P(flat) probabilities between 0.7 and 0.9. Although these probabilities are not yet large enough to accept the null, they provide promise that with possibly a few more samples the probabilities would be large enough to accept the null hypothesis; Thus turning a null result by Frequentist methods into a conclusive result by using Bayesian methods.

Overall the results reported in Table 1 are fairly standard results one would expect from an unbiased collection of randomized controlled A/B experiments. Most experiments result in a null result by both Bayesian and Frequentist methods. Most effect sizes and Bayes Factors are small, which is expected in this area of research. Although there were no exciting results, there were no mysterious results either. The Bayesian hypothesis testing methodology seems to perform just as well as the Frequentist methodology, despite the lack of confidence on the genuine prior.

Future Work

The work described in this chapter applies Bayesian hypothesis testing on existing experiment data. Since it was shown that Bayesian hypothesis testing can be used, given enough historical experiments, it is future work to continue to implement this framework into online tutoring systems to better manage the randomized controlled experiments. One area for future work is to conduct a sensitivity analysis to determine what impact the learned prior has on the statistical analysis. In this work, confidence interval on the learned prior were given, however the difference in statistical analysis was not explored in cases where the learned prior may exist on the extremes of the confidence intervals. Another area of future work is to extend the ASSISTments TestBed[8] to incorporate Bayesian hypothesis testing when running online controlled experiments (Heffernan & Heffernan, 2014; Ostrow et al., 2016). Several Bayesian advantages can then be gained, such as continuous monitoring and optional stopping (Deng, 2016). There already exists a template to implement these methods described in (Basford et al., 1997). It is future work to continue to integrate Bayesian hypothesis testing into online tutoring systems for better experiment methodology that could ultimately benefit thousands of students using these systems.

Contributions and Conclusions

This work makes a first attempt at applying Bayesian hypothesis testing to randomized controlled A/B experiments run inside online tutoring systems. We show that the methods can be applied in this context and extended the existing methodology to do so. We report on how to calculate confidence intervals on the prior probability learned from using expectation maximization on the effect sizes of treatments in experiments. We also show how many experiments would be required to have reasonable confidence estimates on the genuine prior learned from historical experiment data. We show that using Bayesian hypothesis testing generates results consistent with the Frequentist methods in addition to having a more intuitive probability interpretation.

[8] https://sites.google.com/site/assistmentstestbed/

Chapter 8: PeerASSIST

8.1. Introduction

A common problem shared amongst online tutoring systems is content creation. The issue of content creation has been studied in the past for both problem creation and hint creation. It has been estimated that an hour of online instruction can take up to 100-300 hours to create (Murray, 1999). Several different systems have created tools to expedite content creation. Some of these tools include the Cognitive Tutors Authoring Tool (CTAT) for the Cognitive Tutor Algebra (Koedinger et al., 2004), and the ASSISTments builder for the ASSISTments online tutoring system (Razzaq et al., 2009). Programs have also been created to automatically generate tutoring from past data. Examples of these programs include the Hint Factory for the Deep Thought logic tutor (Stamper et al., 2008), automatically generating hints for programming problems (Rivers et al., 2013; Piech et al., 2015), and machine-learning incorrect solution paths to manually generate buggy messages (Selent & Heffernan, 2014). Although these tools make content creation more efficient, they all still depend on the efforts of a content creator and/or past historical data and do not take full advantage of the power of the crowd in real-time.

Recent work has been done in this domain related to crowdsourcing. One example involves creating mathematical word problems and associated hints for the educational math game Riddlebooks (Chen et al., 2016). In this work successive task design was shown to be significantly more effective when used inside Mechanical Turk to create accurate hints. Another example is the construction of expert knowledge bases for intelligent tutoring systems using crowdsourcing (Floryan & Woolf, 2013). In this work, student data from problems on human biology from the Rashi inquiry system was used to create an Evolving Expert Knowledge Base (EEKB). This knowledge base was shown to be less effective than a human expert constructed knowledge base but required 100 hours less time to create from non-experts. A third example was for demonstrating the feasibility of crowdsourcing videos on problems related to solving logarithms from people using Mechanical Turk (Whitehill & Seltzer, 2017). In this work it was shown that people who watched crowdsourced videos had greater learning gains than those who watched an unrelated video; meaning that the crowdsourced videos were more effective than

nothing. It was also shown that people who watched the best crowdsourced videos did not have significantly different learning gains than people who watched a video from Khan Academy.

This chapter introduces a system called PeerASSIST to provide a solution to crowdsource tutoring content (Heffernan et al., 2016). PeerASSIST is a system which crowdsources work done by students and redistributes that work as a form of tutoring to their peers who need help. It is specifically designed and integrated into the ASSISTments online tutoring. ASSISTments has over 3,000 active teachers, 58,000 active students, and over 10 million problems solved over the past year. The ASSISTments online tutoring system has a large amount of standard content aligned to the common core state standards. This content typically has well-made tutoring strategies, which are beneficial to students. However a large amount of assignments that teachers assign are customized to their specific lessons. This content does not have any pre-existing tutoring strategies. In this scenario the teacher must be responsible for either creating a large amount of good tutoring strategies, which can be extremely time consuming, or not create any tutoring strategies at all, which can make the assignment for difficult for students.

It is also possible that some of the existing tutoring strategies created by teachers and other content experts are not as helpful as intended. This can be related to the phenomenon of the "expert's blind spot", where experts overlook important details. This phenomenon is explored in more depth in (Nathan, 2001) and (Nathan, 2003). The PeerASSIST system provides the infrastructure to gain this knowledge since the content will be generated by the non-experts, who will give more attention to details that may be overlooked by the expert.

Typically students must submit their work in order to receive credit. The idea of PeerASSIST is to reuse previously submitted student work to provide to other students who are struggling on the same problem. This leverages the existing work submission features of the platform provides an integrated mechanism to crowdsource tutoring from the students who use the system, without additional burden.

The issues and challenges faced by online tutoring systems provide an ideal environment to implement a solution using crowdsourcing. Six factors for successful crowdsourcing are discussed in (Sharma, 2010). These factors can be applied to the domain of tutor strategy generation in online tutoring systems. The first factor is "Vision and Strategy". This refers to the common goals that are shared by the creators of the crowdsourcing system and the crowd and how clearly the vision is presented and understood by the crowd. In reference PeerASSIST, the

common vision is to help students who are struggling to learn more efficiently. Since we do not directly interact with student through the system, we share our vision by communicating with teachers (via email), who then pass the information on to the students. It is necessary for teachers to share our vision in order for the PeerASSIST system to be used. It is helpful for us to have the students share the same vision as that will likely lead to students submitting better quality work.

The second factor is "Human Capital", which is about the abilities of the crowd. In order for crowdsourcing to be successful the crowd needs to have the basic skills to complete the tasks. Our crowd is made up of the students who use ASSISTments. Since these students are learning and practicing the content that they are providing work for, the skill level will vary among students. Some students will understand the material and are also capable of creating good quality work to be used as hints to share with others. Some students will not be able to create useful hints. Since students will often be required to submit work anyway, there will be minimal to no extra effort to create hints. This design helps us effectively manage the "human capital".

The third factor is "Infrastructure". By design crowdsoucing is not meant to be capital intensive as it leverages the work of the crowd. However in order to provide the system, which the crowd uses an initial infrastructure must be created. PeerASSIST was created by graduate students, where the capital was provided through grants. Since the system has been built, the initial capital spent on building the infrastructure was not an impeding factor.

The fourth factor is "Linkages and Trust". This factor is about the ease of communication and information sharing between members of the crowd. In our context there are a few different types of communications addressed. One type communication is the relation between the PeerASSIST/ASSISTments team and the teachers who use those services. We are fortunate to have good relations with several teachers. This communication is typically done through email, or rarer meetings in person. Another type of communication is between the teachers who use the system. PeerASSIST currently does not provide a means for teachers or students to directly communicate with each other in our system. Unfortunately this minimizes the amount of collaboration between teachers and students. Our teachers trust ASSISTments, however a few teachers were apprehensive about using PeerASSIST due to content control. There have been several features built into PeerASSIST in order to control the content that gets distributed and we believe the system is safe to use.

The fifth factor is “Eternal Environment”. This factor relates to external support for the crowdsourcing project. Although Sharma discusses this factor in terms of government support of business, it is very different in our context. In our context, this is mostly related to the teachers support of the PeerASSIST system. Teachers will contribute to the classroom environment, where teachers will encourage or enforce that students submit work.

All five of these factors contribute to the sixth and most important factor of “Motive Alignment of the Crowd”. This factor is about having the motives of the crowd be aligned to the goals of the crowdsourcing system. Unfortunately and realistically the majority of students (our crowd) have no motivation to help their peers. Their main goal is to finish the required work as quickly as possible, which is not aligned with our goal of providing students with more helpful forms of tutoring to improve learning efficiency. Mechanisms of dealing with this issue have been discussed, such as a rewards system or more selectively choosing the crowd. Currently we are moving towards more selective crowd selection, by having teachers hand-pick “good” students to generate helpful hints. These students are more likely to have their motives aligned with our goals. These students are also less likely to need help. There is a clear distinction between help-givers and help-receivers. It may be best to only include students who are likely to be help-givers in our crowd.

Several considerations were made when thinking about the design of the PeerASSIST system. Many of these considerations addressed the issues mentioned by (Saxton & Kishore, 2013). They analyzed 103 crowdsourcing websites and discussed three major issues related to crowdsourcing. The first major issue is related to the product or service being outsourced. In our case this refers to the peer hints which are being collected as student work. Saxton and Kishore mention that the product being outsourced is something that a computer cannot do but may vary in level of human complexity with the main point being to diversify the product/service. There has been some automation of hints to some degree. However the levels of automation, quality of the hints, and breadth of the domain have been limited.

The second major issue discussed was the level of collaboration. Saxton and Kishore claim that the level of collaboration is not important, but how the collaboration is controlled is important. This is also ideal for ASSISTments, where the level of direct collaboration between students will minimal. Students do not interact with each other through ASSISTments. PeerASSIST will allow students to see and rate tutoring generated by their peers, but there will be no explicit

collaboration in the tutor beyond that point. We focused the design of PeerASSIST on controlling the "collaboration", or when/what hints other students can see. Several rules were created to control when explanations are displayed. These rules will be discussed later

The third major issue is about managerial control systems. The idea is not for the system to make the crowd better but to use the best aspects of the crowd. This applies to PeerASSIST regarding the quality of the hints. Many students will not improve the quality of their work. In fact, from my own observation, student grades tend not to vary much over time. Good students tend to remain good students and bad students tend to remain bad students. Therefore is it important to have the PeerASSIST system work with all types of students to extract the most useful information out of them. This overlaps with personalized tutoring as well, where one hint may work well for one student but not well for another. It may also be the case where one hint creator does a good job for a specific type of student. The PeerASSIST system can take advantage of student features to supply students with the best forms of crowdsourced tutoring. This form of crowdsourcing is called learnersourcing as defined in (Kim, 2015).

Learnersourcing is basically crowdsourcing where the crowd consists of learners. In this case the learners will be the students using ASSISTments. Students will engage in a mix of active and passive learnersourcing. Active learnersourcing is when information is gathered by prompting students for it where passive learnersourcing is when information is gathered while students do their regular tasks inside ASSISTments. The type of learnersourcing students will engage in was hard to classify as either active or passive. It is active in the sense that students are now either required or have the option to submit work in ASSISTments when they normally were not. This can be considered an active form of learnersourcing because it is outside the normal behavior of the student-tutor interaction that existed in ASSISTments. This can be considered a form of passive learnersourcing if we decide to redefine what the normal interactions in ASSISTments are. If we choose to let the submission of student work be considered a normal interaction that is newly introduced to our system, then we can consider the learnersourcing as passive since students are submitting their work regardless of if/how it is used as peer explanations.

The following section will discuss the design and capabilities of the PeerASSIST system. This will include some of the important design decisions and challenges faced. To get an idea of how the system is being used, various statistics on the system usage are presented. These statistics can

be useful to gain insight on what to expect when designing such systems in the domain of online tutoring systems. Lastly, a randomized controlled experiment is presented to determine the early effectiveness of the PeerASSIST system.

In this chapter, the capabilities and underlying infrastructure of the PeerASSIST system are explained. PeerASSIST is a system built as an extension to the ASSISTments online tutoring system, which crowdsources student work and distributes that work to peers in need of help in a controlled and optimal manner. A brief overview of the PeerASSIST system is given here, with more complete details provided in the following sections. Teachers can enable PeerASSIST for assignments for students to work on. Students will have the option to show and share their work for the problems in the assignment. When a student submits work it becomes available to be distributed to other students. This addresses the crowdsourcing aspect of PeerASSIST, where entire classes of students are contributing their work, which can be used as a form of tutoring for their peers. When a student answers a problem incorrectly and receives a score of zero on the problem, another student's work will be displayed to the current student. The work that was distributed was chosen from the pool of previously collected student work. A framework for running multiple simultaneous randomized controlled experiments was designed and implemented to function on a separate server. This server is called the PeerASSIST server and is responsible for choosing which student work to distribute to other students. Several Multi-Armed Bandit policies are used to choose which student work to distribute based on a set of dependent measures to determine "goodness" of each student work. This chapter is ordered as follows. The first section of the chapter discusses how PeerASSIST is integrated with the ASSISTments tutoring system. How PeerASSIST works both from a user's perspective and from a system design perspective will be explained. The second section will focus on just the PeerASSIST server, which is responsible for choosing which student work gets distributed to other students. Since much of the time spent on the project was on programming a scalable, maintainable, and reusable platform, a large amount of this section will discuss important implementation details on the inner workings of PeerASSIST.

A high level overview of the system architecture is shown in Figure 8.1. Prior to the existence of PeerASSIST, ASSISTments consisted of a single server with a single database located on the same machine as the server. In the diagram, these components are labeled "ASSISTments Server" and ASSISTments Database". Students, who do classwork or homework, request the

assignment data from the ASSISTments server/database and also send back data as actions are performed and recorded in the tutor. The new PeerASSIST system handles all requests for peer assistance. When a student starts a problem, a separate request is also sent to the PeerASSIST server. The PeerASSIST server will map that user to a given experiment policy if that user has not already been mapped. This mapping will be stored in the PeerASSIST database. An experiment in this context refers to a mapping from a user to a given multi-armed bandit policy used to distribute work. Once a user is mapped to an experiment, they remain mapped for the duration of the experiment. The PeerASSIST server will run database queries to obtain necessary data from both the PeerASSIST database and the ASSISTments database. This data will be used by the multi-armed bandit policies to determine which student work to send back to the student. Due to performance and system integration details, student work is sent down to the student's client machine at the time that the problem is loaded and not at the time help is requested (note that student's do not explicitly request help in the current implementation). Although work is sent to the student's machine, this work is not necessarily displayed to the student, because the student may answer the problem correctly and not need any help. Data on distributed work is only stored if the work is actually seen by the student.

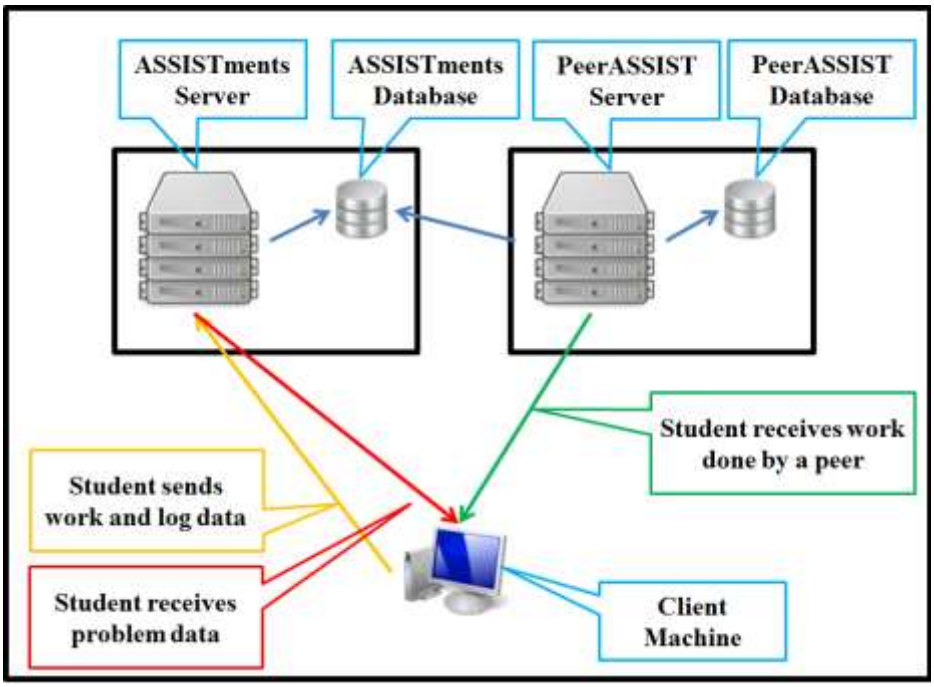


Figure 8.1: PeerASSIST system architecture

8.2. PeerASSIST Integration inside ASSISTments

This section describes the additional features added inside ASSISTments to support PeerASSIST. This includes a walkthrough of how PeerASSIST is used inside ASSISTments from both the teachers' perspective and the students' perspective. The functionality and user interfaces within ASSIASTments are shown. Applicable UML diagrams (Use Case, Activity, and State Machine) are used to explain some of the functionality when needed. At the end of this section, an ER diagram of the ASSISTments database, along with a description of the tables and columns, is shown for all tables related to storing data related to the PeerASSIST system. To start using PeerASSIST or PeerASSIST related features, a teacher must first enable it through the teacher settings under the preferences tab inside on the ASSISTments website. Figure 8.2.1 shows an activity diagram of the process that a teacher goes through to enable one or more features of PeerASSIST and assign an assignment with PeerASSIST enabled. The first step of this process involves the teacher to enable the work box, PeerASSIST, or both options. These two actions are done through the teacher preferences page in the ASSISTments system shown in Figure 8.2.2. The order of these actions does not matter and both can be done in parallel as shown in the activity diagram. The screen shot shown in Figure 8.2.2 corresponds to the first activity in the activity diagram, labeled as "Choose PeerASSIST settings". The two black bars in the activity diagram mean that the actions "Enable/Disable PeerASSIST" and "Select Teacher Work Options" can be done in parallel independently of each other. At this screen, the functionality related to PeerASSIST is described in the Use Case diagram in Figure 8.2.3. There are several other settings that a teacher can manipulate from the settings page, however only the settings related to PeerASSIST are important in this context. The two main settings that a teacher can choose at this page are settings for the "work box", which allows students to enter their work on problems, and the settings for PeerASSIST, which allows students to see work done by their peers, who have the same teacher, when they reach a score of zero on a problem.

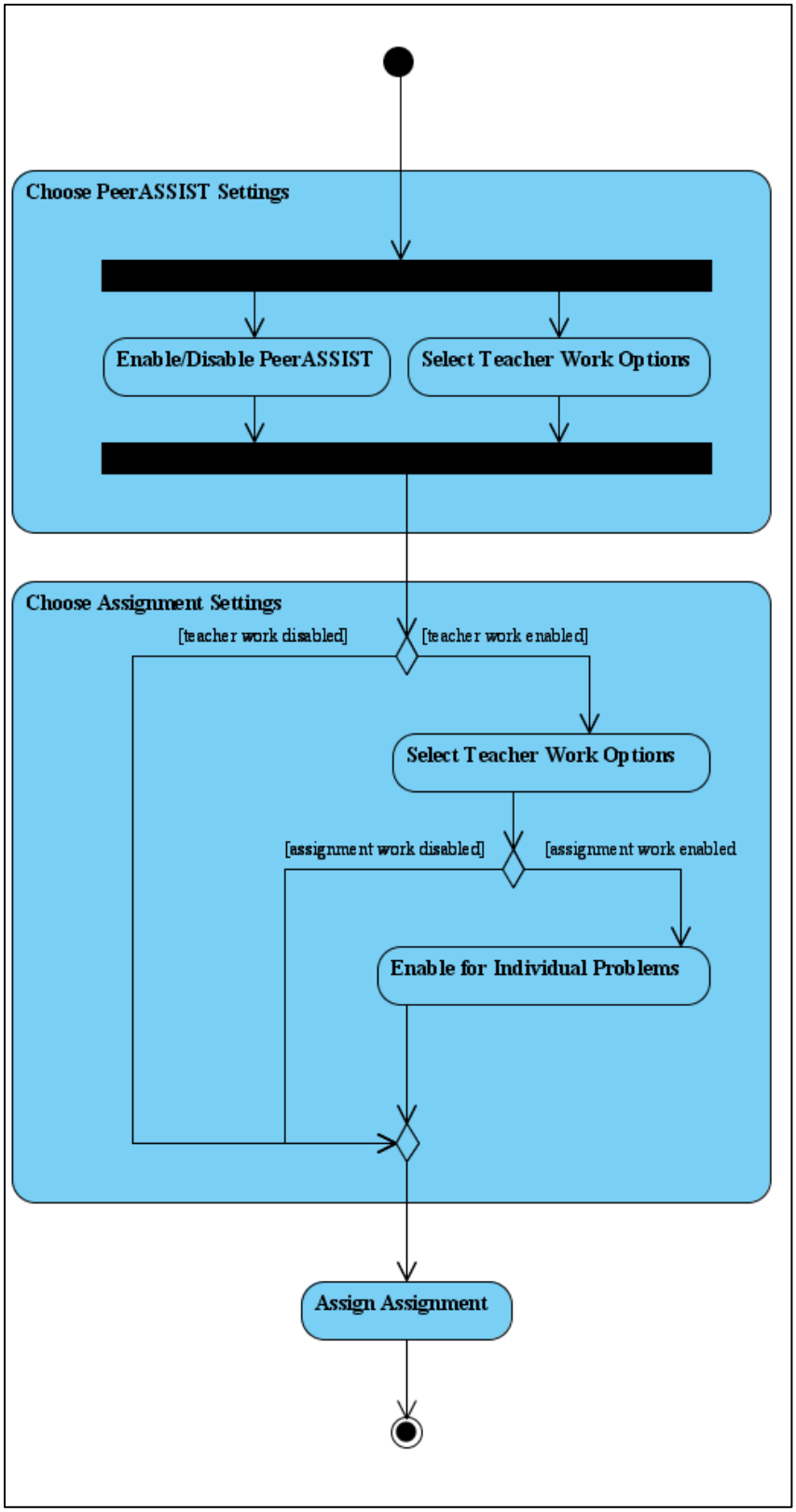


Figure 8.2.1. Activity diagram to describe the process of enabling PeerASSIST and assigning an assignment with PeerASSIST enabled.

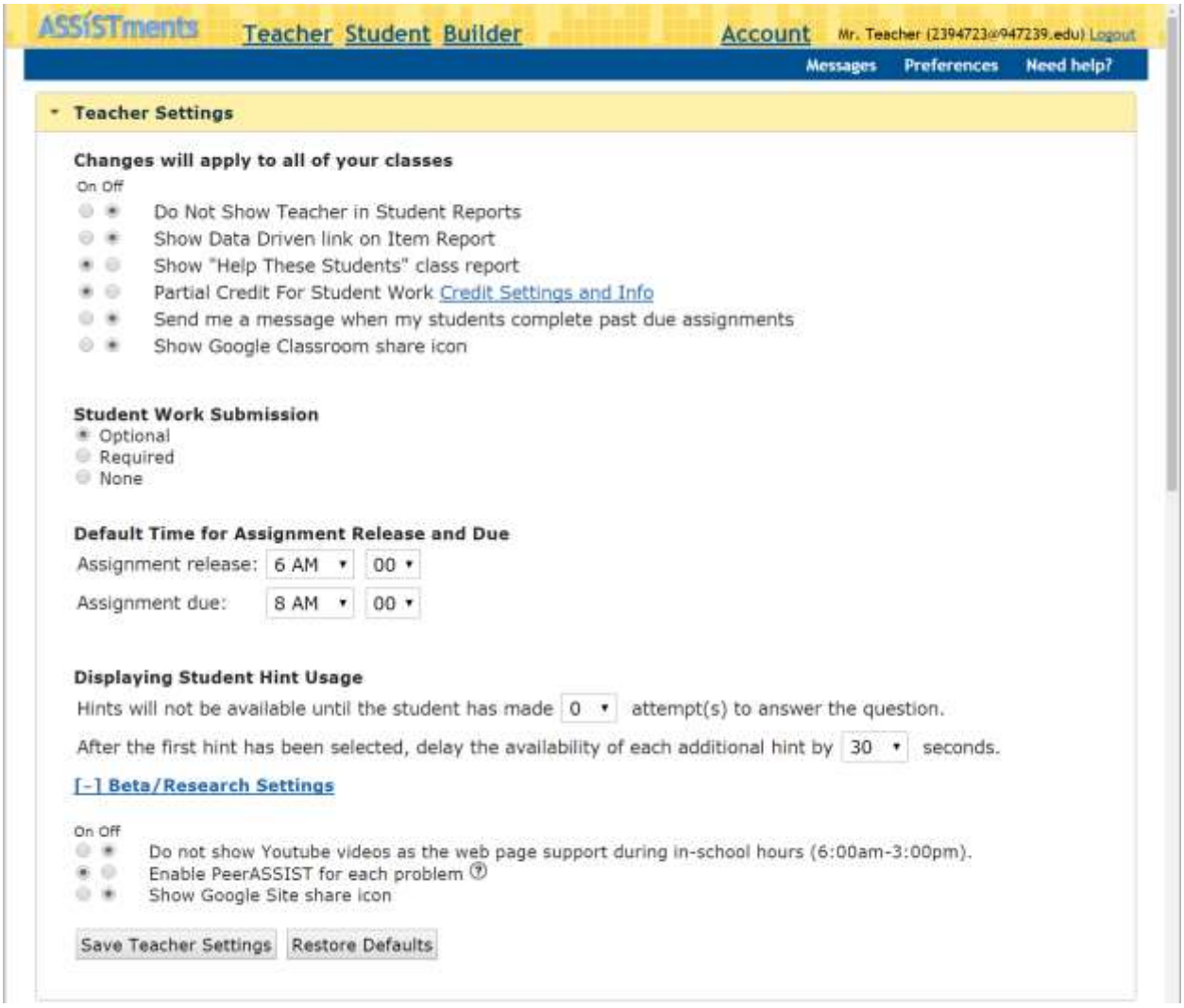


Figure 8.2.2. This figure shows an image of the teacher settings page under preferences inside the ASSISTments system. The Beta/Research settings are expanded to show how to enable PeerASSIST.

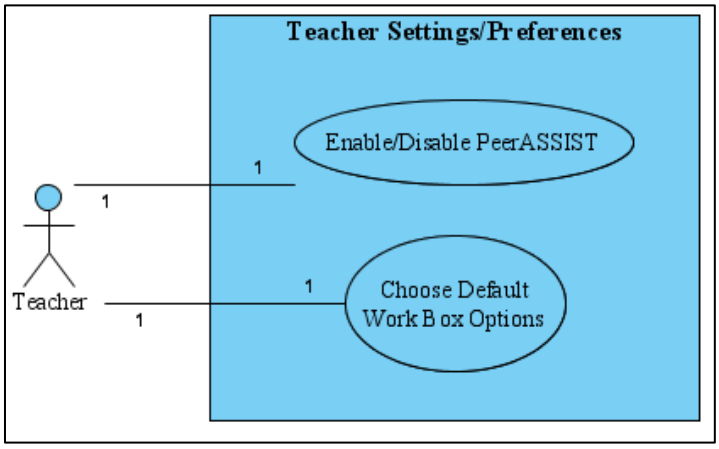


Figure 8.2.3. Use Case diagram of the functionality related to PeerASSIST on the teacher settings page.

Under the “Teacher Settings” tab, there is a radio button for the default student work submission. This option sets the default option that appears on the assign forms when a teacher assigns an assignment. There are three options for student work submission, “Optional”, “Required”, or “None”. If student work submission is optional, then the work box will appear on the students screen while they interact with the tutor. The specific details and capabilities of the work box will be described later in this section. Students will be able to submit their answer to the problem and also have the option to submit work as well. If student work submission is required, students must submit work in the work box before submitting an answer to the problem. A student will not be able to submit the answer to the problem unless they have entered text inside the work box. If no student work is necessary, the work box will not appear on the student’s screen and students will not have the option to enter work for problems.
There is a setting to enable PeerASSIST located under “Beta/Research Settings” expandable link. Turning this option on will enable PeerASSIST for assignments assigned after it has been enabled. There is a distinction between the work box and PeerASSIST. Although the two components are separate, they often work together. Enabling the ability for students to show their work for problems, allows students to show their work. Enabling PeerASSIST will allow students to see work done by their peers, if any exists. The two settings can be enabled and disabled separately, however enabling PeerASSIST without allowing student to show their work, will likely result in not having any PeerASSIST content being available since work cannot be redistributed if there is no way to submit the work to begin with. Therefore in order to gain any benefit from PeerASSIST, students must be able to show their work. There are a few special cases, where PeerASSIST can work without having the work box enabled. One of these scenarios, is where work is generated by teachers or students prior to a given assignment. Since students can see work done by other students who have the same teacher on the same problem, there may be options to use this for future research. One idea is to have “good” students, do an assignment and generate work as well. Another assignment can then be assigned with the same problems, but without the option to include work. When these students see any PeerASSIST, the content will be generated by only the “good” students. This is one method that can be used to probabilistically increase the chances of “good” content being distributed to other students using ASSISTments and PeerASSIST.

After a teacher has saved his/her settings, the next step is to assign an assignment for the students to work on. This relates back to the activity diagram shown in Figure 8.2.1 and refers to the "Choose Assignment Settings" activity. A Use Case diagram shown in Figure 8.2.4 shows the functionality related to PeerASSIST that a teacher can interact with at this page. There is additional functionality related to manipulating assignment options, however only the functionality related to using PeerASSIST is shown.

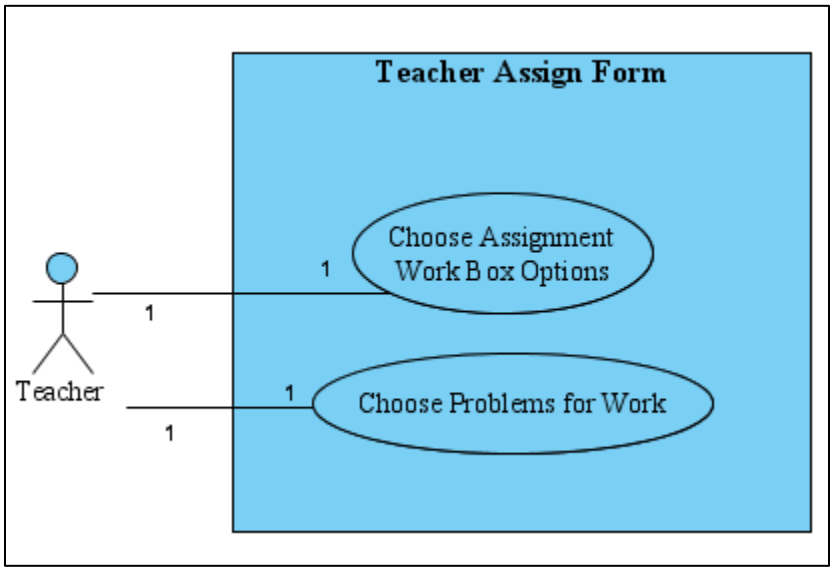


Figure 8.2.4. Use Case diagram for PeerASSIST functionality at the assign page.

On the assignment form shown in Figure 8.2.5 the work box options are shown again. The radio button is preselected for the option that the teacher has chosen as the default option under the teacher settings previously described. If the work option is either required or optional, teachers have the ability to choose specific problems that they want students to show work for. It is often too time consuming for students to submit work for all problems or it is not necessary to show work for all problems. This feature allows teachers to disable the work box for problems of their choosing. There are a few cases where the system will override the teacher's choice. For essay problems, there will never be a work box because the essay box is very similar and having two boxes is unnecessary. Also the work box will never be shown for scaffold problems because of system limitations, simplicity for students, and the idea that all work should be shown only for the original problem. Note that there is no option for PeerASSIST on the assign form because PeerASSIST is currently a setting that is either enabled or disabled for all assignments.

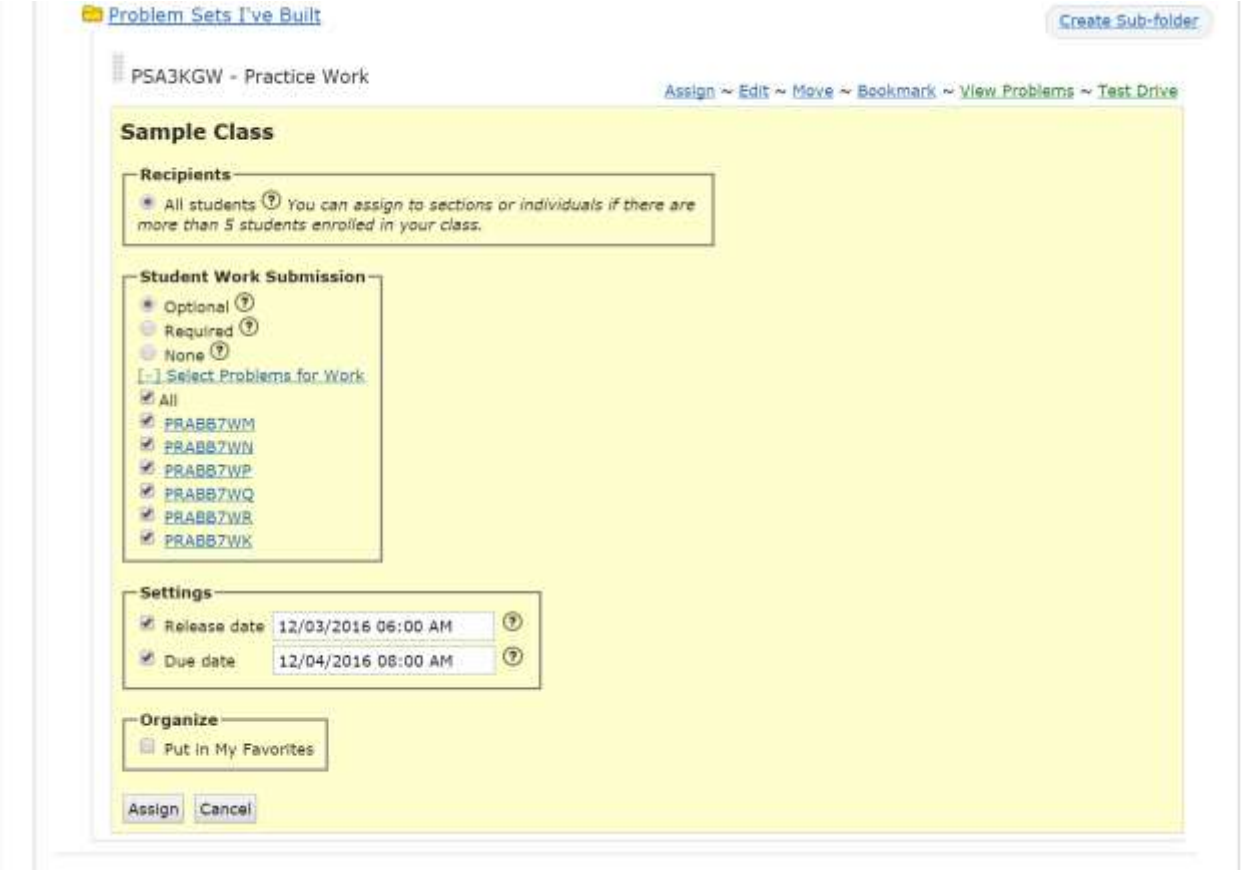


Figure 8.2.5. An image of the assign form inside of ASSISTments.

Once an assignment is assigned, students can work on the assignment. The code base used to generate the student views and functionality is a separate code base than the other webpages on the ASSISTments website. All other pages are programmed in Ruby, where the tutor is programmed in Java using the Google Web Toolkit (GWT) library (Smeets. 2008). The Google Web Toolkit allows for Java code to be written once and compiled into several forms of Javascript used by the different browsers. There are several advantages of using GWT instead of directly writing Javascript. One advantage is that Java is a strongly typed language where all variables have a specific declared type. This helps prevent casting/conversion confusion when working with variables of different types. Java is a compiled language and several errors can be checked at compile time. If any compilation errors occur, the output files to run the program will not be generated. Javascript is an interpreted language and is not compiled, therefore no compilation errors can be checked before execution. The main advantage for using GWT is that different browsers (mainly Internet Explorer) function differently. Instead of writing special

versions of Javascript for each browser, a single version of Java code can written using the GWT libraries and then compiled into Javascript for each browser. This means that multiple versions of Javascript do not have to be individually maintained, which can be both tedious and error-prone.

Figure 8.2.6 shows an example view of what a student sees as they solve problems inside of ASSISTments. In this example both the work box and PeerASSIST are enabled. In this hypothetical example, a fictitious student, named Ashley, is working on problems on solving two-step equations.

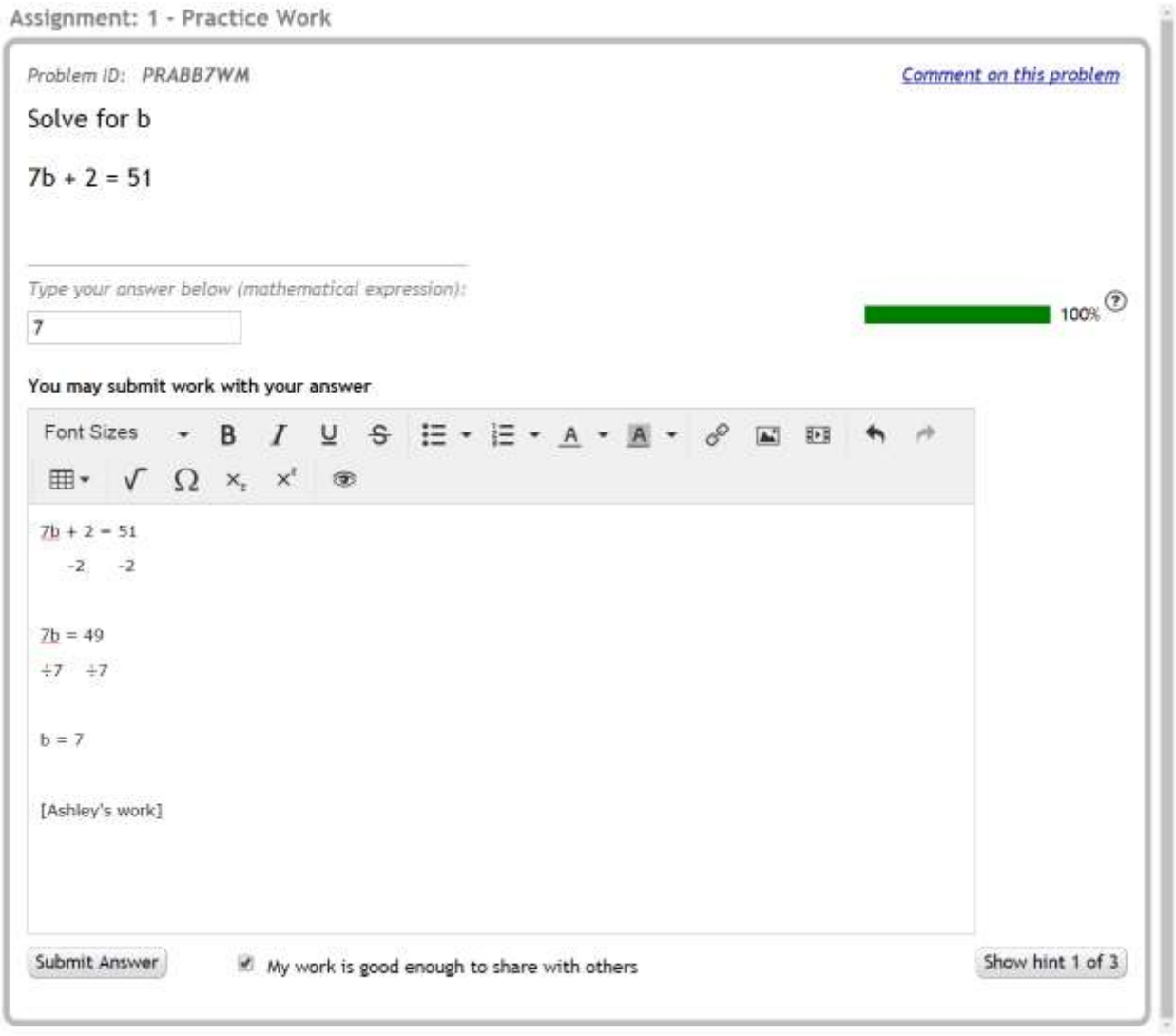


Figure 8.2.6. An image of a student ("Ashley") solving a problem and showing her work.

A TinyMCE box is provided for students to enter their work on problems. This allows students to enter work in the form of stylized text, images, videos, and equations. Providing students with these tools enables the creation of higher quality work although it is most common for students to just enter plain text, since it is simpler and quicker to do.

There is text above the box, which provides students instruction. This text in shown in Figure 8.2.6 currently states, "You may submit work with your answer". This text can be modified to elicit different types of responses from students. The modifications are not intended to be something the teacher can do but something that researcher can programmatically do to run different types of experiments with different messages. This text can be considered a type of external script where different instructions can lead to different responses. For example the script, "Show your work" may produce a response different than "Explain your work". Both scripts are only three words long and only the first word is different, however the responses elicited from students may be different just by changing one word. Students who are told to explain their work may put more effort into explaining the steps of the problem rather than merely showing the steps without any explanation. An explanation accompanying the work may be more beneficial to peers when used as tutoring content, and may also be beneficial to teachers who can more easily understand what students' misconceptions are.

Typically these scripts are more complex. Fisher and Schank have much more complex theories behind scripting, which is broken down into four components and seven principles (Fisher et al., 2007; Schank, 1999). In a typical environment using scripting there are usually several scripts involved, both internal and external. In our case we use a very simple non-interactive version of a script. Kollar et al. show that high structured external scripts helped learners regardless of their level of internal scripts (Kollar et al., 2004). Although this will likely not be as powerful as a fully implemented interactive scripting prompt, we hope it will be sufficient enough to provide some benefit. The benefits of minimal scripting are discussed in (Dillenbourg, 2002). It is stated that overscripting can interrupt the natural interactions and problem solving process. This is also stated in a positive manner discussed in the following paragraph below. Dillenbourg also states that the scripts can increase the cognitive load on students. I believe this is a major concern with middle school students, who can't handle much cognitive load. A complex scripting scheme could do more harm than good.

Weinberger et al. have shown the effects of different scripts in the context of discussion boards used in a study where students analyzed case problems on attribution theory (Weinberger et al., 2004). The goal of this study was to improve knowledge convergence focusing on process convergence and outcome convergence using social and epistemic cooperation scripts. Knowledge convergence is the amount of knowledge concepts that all the learners share. Learners who all know the same concepts would have a high knowledge convergence, and learners who all know different concepts would have low knowledge convergence. Process convergence is when learners communicate individual concepts which are then internalized by other learners. Outcome convergence is when an individual leaner can apply the knowledge gained and shared via the collaboration group. Social scripts are scripts that help leaners handle the social tasks of collaborative learning and epistemic scripts focus on helping learners deal with leaning task and stay focused on the goals. Weinberg shows that epistemic scripts without social scripts produced higher process convergence and social scripts without epistemic scripts produced higher outcome convergence. The main point to take away from this is that different forms of scripting can have different effects on students.

In the example shown in Figure 8.2.6 example Ashley has entered work in the work of simple text for this problem. To encourage accountability and for the purposes of clarity in this example, Ashley has signed her work as "Ashley's work". Ashley has entered a correct answer and has provided work for it. Ashley has also checked the box that says "My work is good enough to share with others". When Ashley submits her answer to the problem, her work will also be submitted with it. Ashley's work, as well as most other data associated with the problem, is sent up to the ASSISTments server at ten second intervals. Batching the data together minimizes the bandwidth usage of the network. This helps keep the requests hitting out server lower and more importantly helps reduce the usage of the school networks which are often perform poorly. Ashley's work will be added to a pool of work, which can be used by the PeerASSIST system to redistribute as hints to other students. Student work can be considered for redistribution if it meets a set of criteria. Firstly, the work must be associated with a correct answer. If a student does not answer the problem correctly on the first attempt, their work will not be considered for redistribution. The reason for this is to prevent incorrect work from being distributed. Although it is not implemented yet, it is possible for students to first answer the problem incorrectly and then answer the problem correctly and fix their work. This scenario is being considered for

future work, to include work that was submitted with a correct answer although the problem was not initially answered correctly.

The second criterion for work to be considered for redistribution is if the work is sharable. Work is considered sharable if the student has selected the checkbox which states that "My work is good enough to share with others". This provides a means to filter out poor quality work and also provides a way for students to consent to sharing their work with other students. The option of having the checkbox selected or deselected by default is explored in section 8.4.

The third criterion is that work can only be redistributed to other students who share the same teacher (excluding a student from seeing their own work). This is for control over the content that gets redistributed. Teachers may be alienated or scared to use PeerASSIST, without knowing what students are generating the content. Since ASSISTments and PeerASSIST is being used by children in grades K-12, content control is an important issue. Several teachers have expressed fear of the content that is distributed. This allows teacher have more control over the content that is distributed. If any issues arise, the teacher can talk to the students directly.

A preapproval feature was suggested where student work will not be redistributed unless it was previously approved by a teacher. This feature was not implemented for the following reasons. Two common scenarios are when students are doing classwork or homework. If students are doing classwork, it is unlikely that a teacher would be focusing on reading and preapproving content in real-time during the class. Teachers will often have to focus on managing the class and will not have time to approve work. If students are doing homework, it is unlikely that a teacher will be monitoring and approving work throughout the afternoon and night. For these reasons, preapproval on student work was an idea that we believed would not work effectively, therefore was not implemented.

This also leads into the fourth criterion, where work can only be distributed if it has not been flagged as inappropriate by another student or teacher. Any work that was been flagged as inappropriate will not be considered for redistribution. If that work has already been distributed, then it will still be available to those students who it had been distributed to. Due to implementation details and experiment consistency, it is not possible to remove peer hints that have already been distributed. Teachers also have the ability to unflag student work. This was implemented to handle the scenario where a student may be flagging valid work inappropriately. The teacher can unflag the work. Work that has been both flagged and unflagged is valid for

redistribution. In addition to manually flagging content, student work is scanned automatically by the system and automatically flagged by the system if it is determined to contain inappropriate language. A list of over 500 "bad words" is used to check against the words in the work submitted. If any words match words in the bad words list, then the content is automatically flagged as inappropriate by the system. This preemptive flagging helps prevent any inappropriate content from being distributed to a single student and maintains the integrity of the system.

Continuing with this example, another student ("Josh"), who is in the same class as Ashley, is working on the same problem. Josh is not having as much success as Ashley. Josh has been entering incorrect answers on the problem. Once Josh's credit score has reached zero, any available PeerASSIST content will be displayed. If there is no content to display, Josh will not see anything. If Josh does not see any PeerASSIST content, the attempt to see the content will be logged in the ASSISTments database. This allows for data analysis for the intent to treat. If Josh does see PeerASSIST content, that will also be logged as well as what content was shown. PeerASSIST is based on the partial credit settings of the teacher. Teachers can set how many attempts students have on a problem before their scores reaches zero. Students prefer partial credit instead of immediately getting the entire problem wrong after one incorrect attempt. Partial credit has also been shown to better model student performance (Wang et al., 2010; Wang & Heffernan, 2013; Ostrow et al., 2015, VanInwegen et al., 2015). Therefore PeerASSIST will only be shown once a student has reached zero credit for the problem. It is likely that PeerASSIST content will contain the answer to the problem; therefore it is only shown when a student has reached a score of zero. It may also annoy students if it pops up too early or on the first attempt without giving the student a chance to correct their mistake on their won.

Figure 8.2.7 shows an image of Josh's screen where Josh has received a score of zero and is being presented with Ashley's work. Josh has the ability to rate the work as "good" or "bad" by clicking on either the "thumbs-up" or "thumbs-down" buttons. This rating will be used as one factor that is considered by the multi-armed bandit algorithms running on the PeerASSIST server, in order to determine which peer hints to redistribute to other students. Hints with a higher rating will have a higher probability of being distributed than hints that have a lower rating. Josh also has the ability to flag Ashley's work as inappropriate and provide a reason why. Josh's teacher will be notified via the internal commenting system inside of ASSISTments and

Ashley's teacher will be notified of the flagged content. Currently both Josh and Ashley share the same teacher, however this was designed to be scaled outside of the classroom, where students could potentially see content generated by students from another teacher. Therefore anytime content is flagged, the teacher of the flagger is notified and the teacher of the student who generated the content is notified.

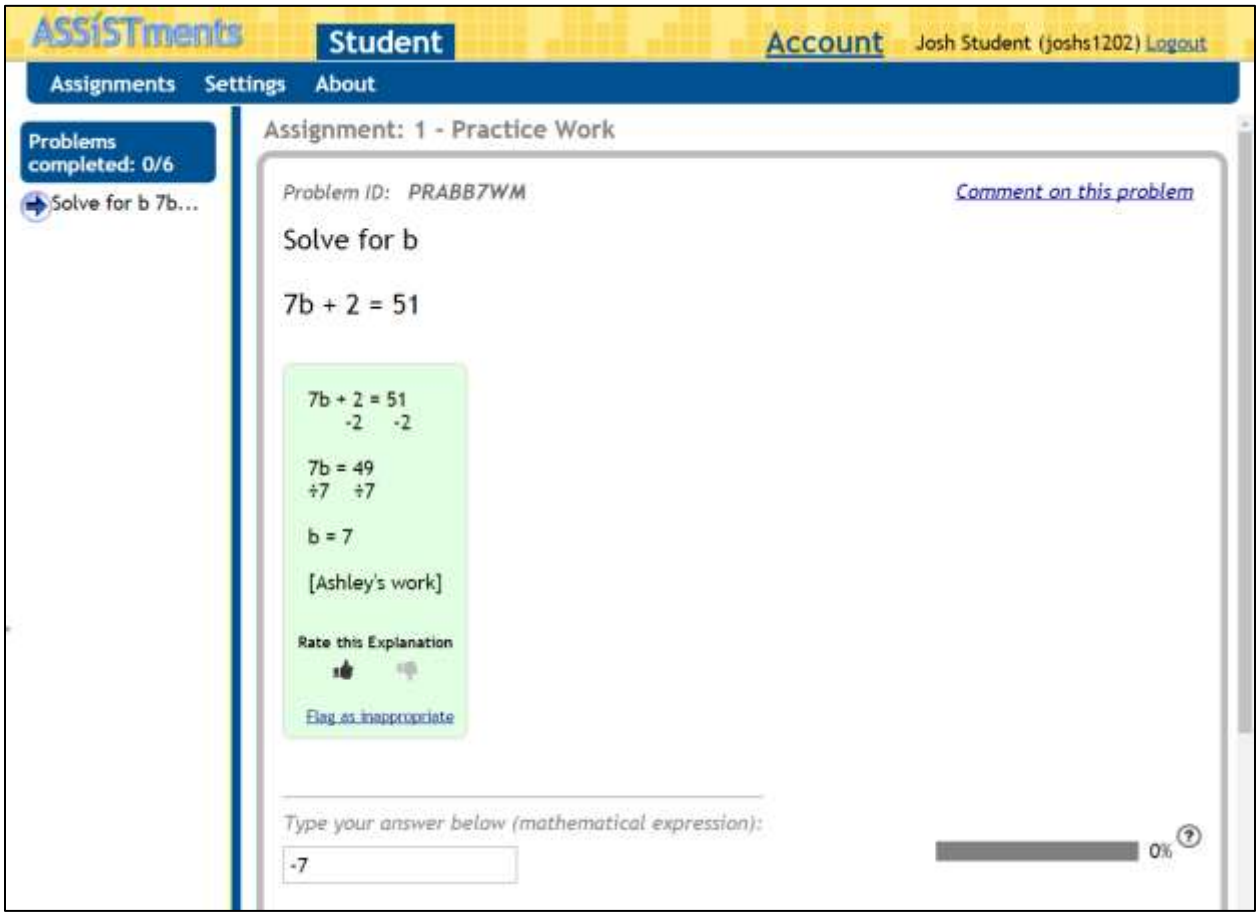


Figure 8.2.7. An image of a student ("Josh") who is receiving a peer hint (work done by "Ashley")

Figure 8.2.8 shows a Use Case diagram that summarizes all the functionality that students can do related to PeerASSIST. Students can submit their answer, which includes submitting their work if work was allowed. Students can also select whether they want to allow their work to be shared with others. If students are presented with hints created by their peers, they can rate the hints and flag the work. If work is flagged a reason can be provided on why the work was flagged.

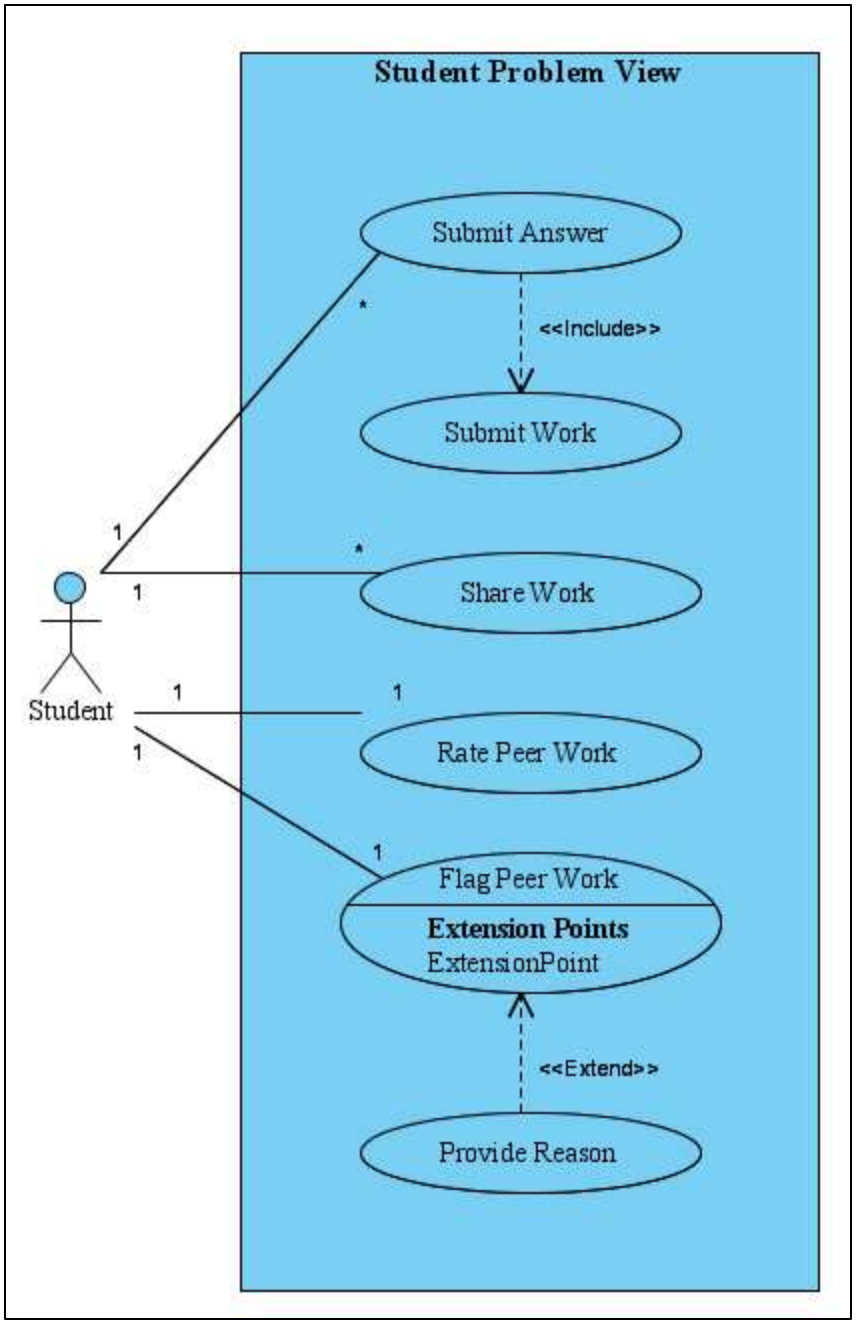


Figure 8.2.8. This figure shows a Use Case diagram for all the functionality students have access to from the tutor view that is related to PeerASSIST. Note that the "extends" key word is used to represent optional functionality and the "includes" key word is used to represent mandatory functionality.

Teachers can also see the work that students have submitted. A "Work Report" was built into the ASSISTments website to display details of student work to the teacher. Figure 8.2.9 shows an example of the teacher Work Report. This continues from the example being used with Ashley and Josh. On the work report, teachers have the option to anonymize or unanonymize student named. This is so that teachers can present their screens in front of the class without student's knowing who did what. Figure 8.2.9 shows the students work for Ashley and Josh; however their names are anonymized and only shown as "XXXXXX". On this report teachers can select to see work for a given class section and problem number. For each student work entry, the answer is shown, as well as the work. From this screen teachers can also rate and flag the student work. Teachers can see the total number of up-votes and down-votes that a given students' work has received, which includes the teaches' vote as well. Teachers can see if the student chose to have their work sharable to other students. Although the next question performance for students who received work is not shown in the image, it has been recently implemented and would appear right before the sharable column.

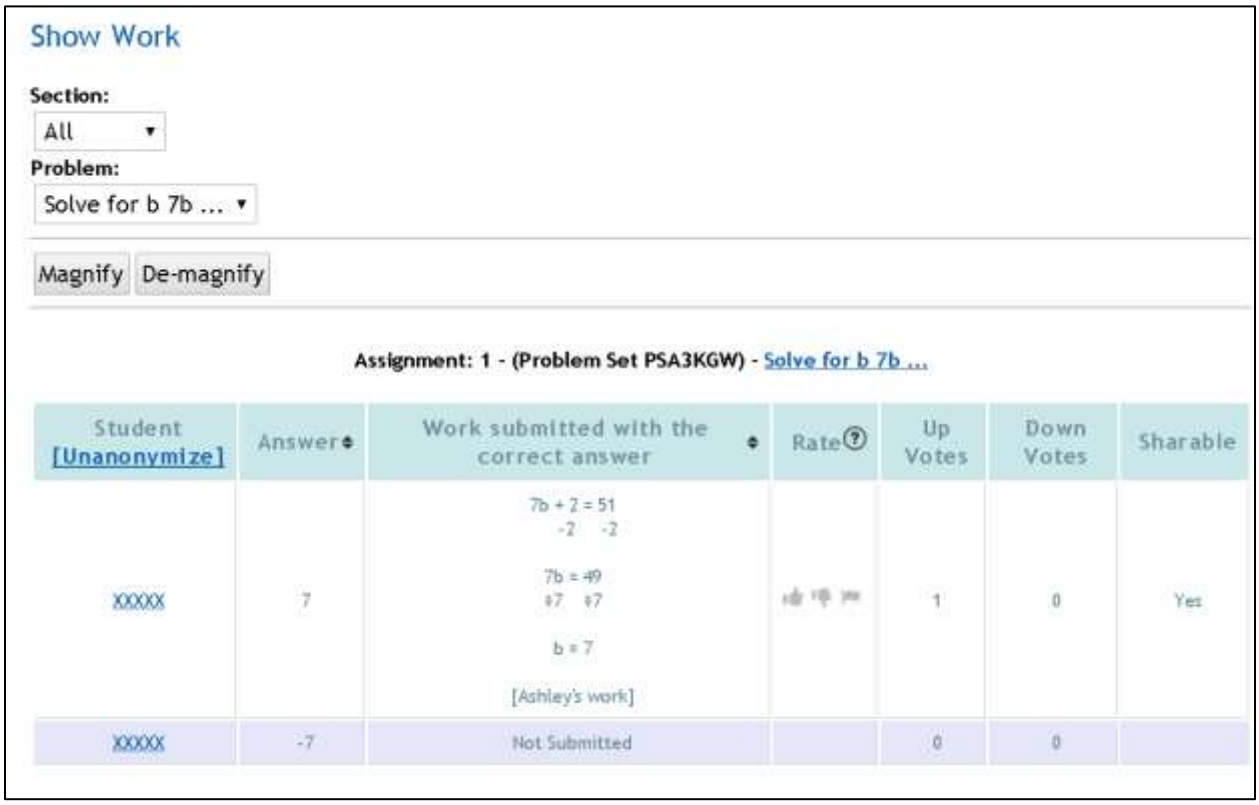


Figure 8.2.9. Teacher Work Report

Teachers also have the ability to flag students work from the work report. Teacher flagging appears simple to the teacher, where they can simply click on the flag image button. Internally the flagging process is quite complex to ensure that information related to flagging and unflagging is not lost. Figure 8.2.10 shows a state machine diagram for how the flagging process and associated data is managed internally.

In the first state the student work has never been flagged. Three pieces of information are shown. "explanation_flagged" is a Boolean variable that represents if the student work has ever been flagged either by the student, teacher, or the system. Once this variable is set to true, it can never be set to false. "explanation_unflagged" is a Boolean variable that represents if the student work has ever been unflagged by the teacher. Student work can only be unflagged, if it had been previously flagged. Once this variable is set to true, it also can never be set to false. "unflag_active" is a Boolean variable that represents if the unflag entry is active. This variable can only be set to a true value if the student work has been unflagged. This variable can be toggled on/off. If this variable is set to 'true', then the unflagged status is active and the student work has been unflagged and is not considered inappropriate. If this variable is set to 'false', then the unflagged status is not active and the student work has been flagged but not unflagged; thus being considered inappropriate.

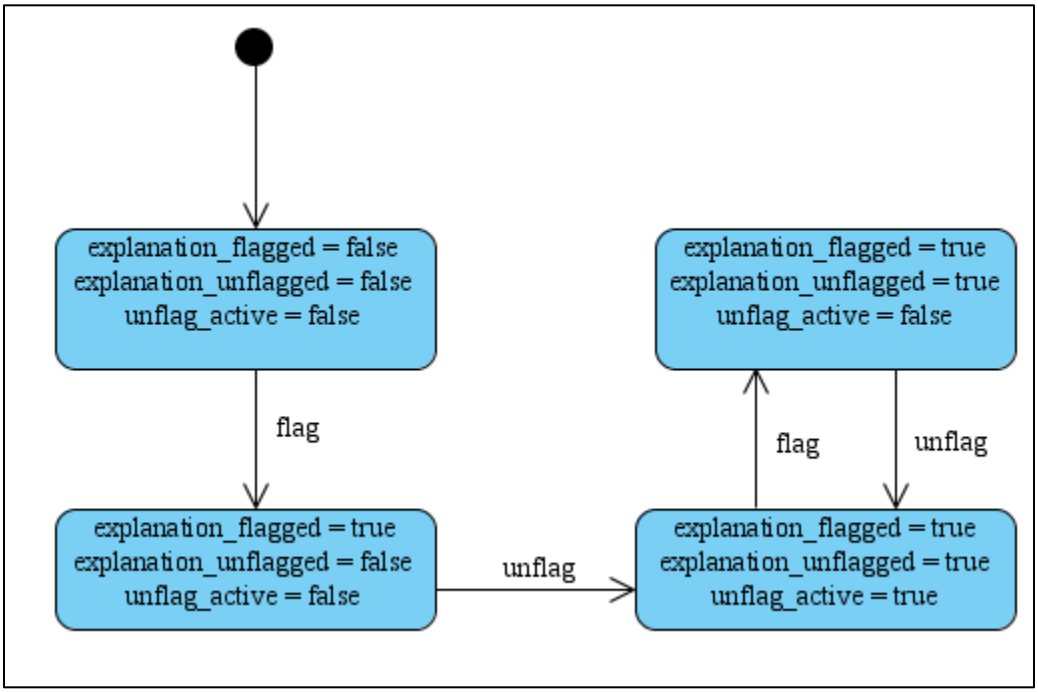


Figure 8.2.10. State machine diagram to model the internal state of the flagging process.

The state machine starts with work that has not been flagged. Each state is made up of three variables, “explanation_flagged”, “explanation_unflagged”, and “unflag_active”. The specific details of the database tables are not discussed here; however it involved both the explanation_flaggings table and the explanation_unflaggings table. At this initial state, the teacher has only one option/ state transition, which is to flag the given work. When work is flagged the “explanation_flagged” variable becomes true. In this state the teacher will see the flag image, filled in with black, indicating that the work is currently flagged. From this state the teacher has the option to unflag the work. When work is unflagged, the “explanation_flagged” variable is not changed. The work has been flagged and that cannot be undone. However since the work has been flagged it can be unflagged. Therefore the next state is when a teacher chooses to unflag work that they have already flagged. When work is unflagged the “explanation_unflagged” variable is now set to true as well as the “unflag_active” variable. This means that the work has been unflagged and the unflagging is active. From this state a teacher can then flag the work again. Since a given user can only flag work once, and the work has already been flagged by the teacher, attempting to flag the work again does not change the “explanation_flagged” variable. In this case, the “unflag_active” variable is changed from true to false. This means that the explanation was flagged, unflagged, and then flagged again. Since the unflagging is no longer active, the work goes back into a state where the work has been flagged. This complex internal implementation keeps the flagging infrastructure consistent, while maintaining UI simplicity for the teacher. The teacher just simply sees the toggling of a button and may use it as desired. In the state machine diagram the student work is considered appropriate in states two and three.

The work report functionality is summarized in the Use Case diagram in Figure 8.2.11. This diagram shows the main functionality and things that a teacher can do from the work report. Anonymization and magnification are not includes in the diagram, but the teacher can choose these options. This functionality is not specifically related to the work report and therefore was left out of the Use Case diagram.

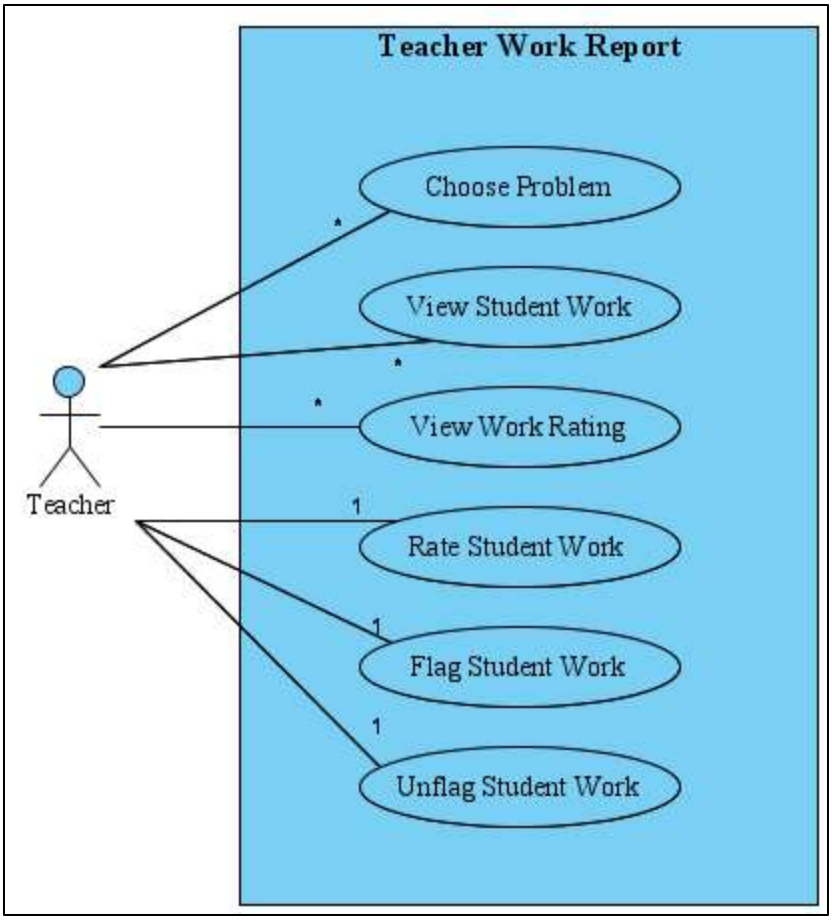
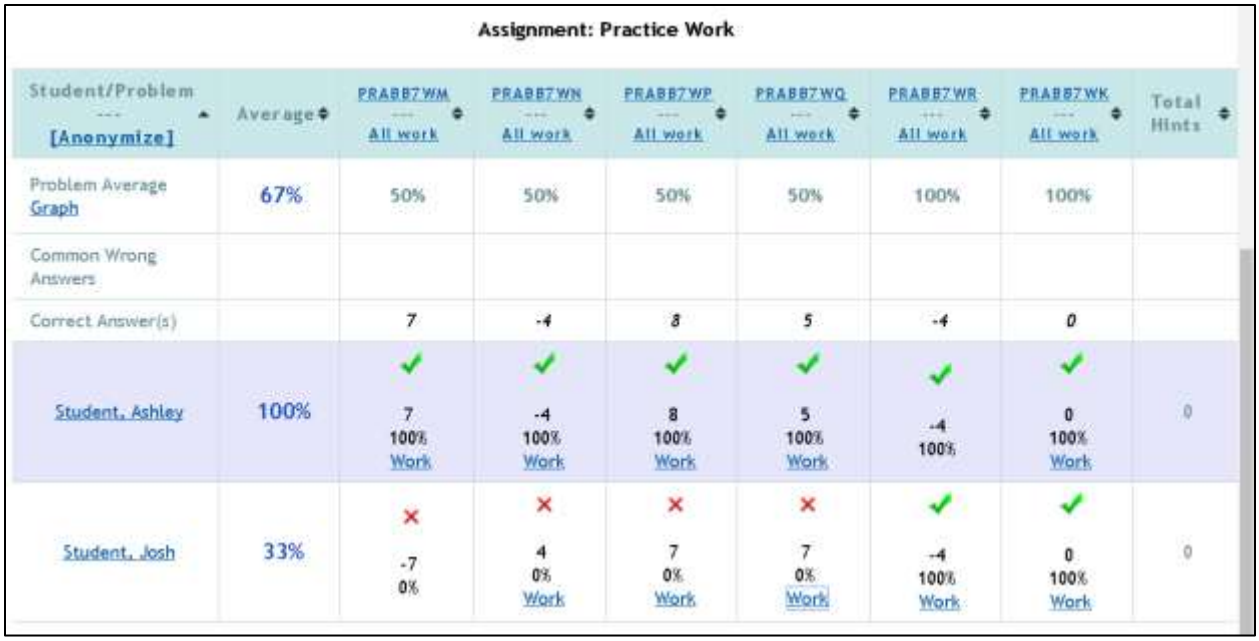


Figure 8.2.11. Use Case diagram for the teacher work report.

Teachers can also see work on the existing item report shown in Figure 8.2.12. This was built into the item report for expediency and convenience for the teacher. If a teacher just wants to quickly see single instances of student work, the teacher can click the work link in the cell of the item report. A popup dialog will appear, showing the student's work and will have the buttons for the teacher to rate or flag the work. Work links will only appear in cells where the student has shown work on the problem. A teacher can also click the "All work" link at the top of a column for a given problem. When this link is clicked, the work report page will load with the current problem being selected.



Assignment: Practice Work

Student/Problem [Anonymize]	Average	PRABB7WM All work	PRABB7WN All work	PRABB7WP All work	PRABB7WQ All work	PRABB7WR All work	PRABB7WK All work	Total Hints
Problem Average Graph	67%	50%	50%	50%	50%	100%	100%	
Common Wrong Answers								
Correct Answer(s)		7	-4	8	5	-4	0	
Student, Ashley	100%	✓ 7 100% Work	✓ -4 100% Work	✓ 8 100% Work	✓ 5 100% Work	✓ -4 100%	✓ 0 100% Work	0
Student, Josh	33%	✗ -7 0%	✗ 4 0% Work	✗ 7 0% Work	✗ 7 0% Work	✓ -4 100% Work	✓ 0 100% Work	0

Figure 8.2.12. Teacher item report with students who have done work on problems.

In addition to the workflow and functionality, database design is described here. Figure 8.2.13 shows an ER diagram for the table additions in the assistment_production database to support the PeerASSIST system. Note that not all tables in the assistment_production database are shown. There are over 250 tables in the database and most of those tables are not applicable to PeerASSIST. Some referenced tables are also not included, such as the users table. Although some PeerASSIST specific tables reference columns in other tables that are not shown, I felt it was not necessary to show those tables. The focus is not to show the complete ER diagram for the assistment_production database, but to focus on the tables that are specific to PeerASSIST data. Therefore some foreign key constraints are absent from the diagram. Also note that the problem_logs table was included as its references are important, however only the id column is shown because the other 20+ columns are not important in this context. All column data types were represented as accurately as possible, however due to limitations of the modeling tool; there may be slight differences between the ER diagram and the actual database. These differences are mostly in terms of column data types. A list of tables and column descriptions are provided below. Descriptions for the columns, (“id”, “created_at”, “updated_at”, and “deleted”) are defined here and omitted below to prevent duplication.

Common Columns

1. id (integer) – The unique id for the table entry and the primary key for the given table.
2. created_at (timestamp) – A timestamp to store when a given table row was created.
3. updated_at (timestamp) – A timestamp to store when the table row was last updated.
4. deleted (boolean) – A Boolean value to represent whether a row was deleted. This allows for the soft deletion of data, where the data is represented as having a deleted status, but is not actually physically removed from the database.

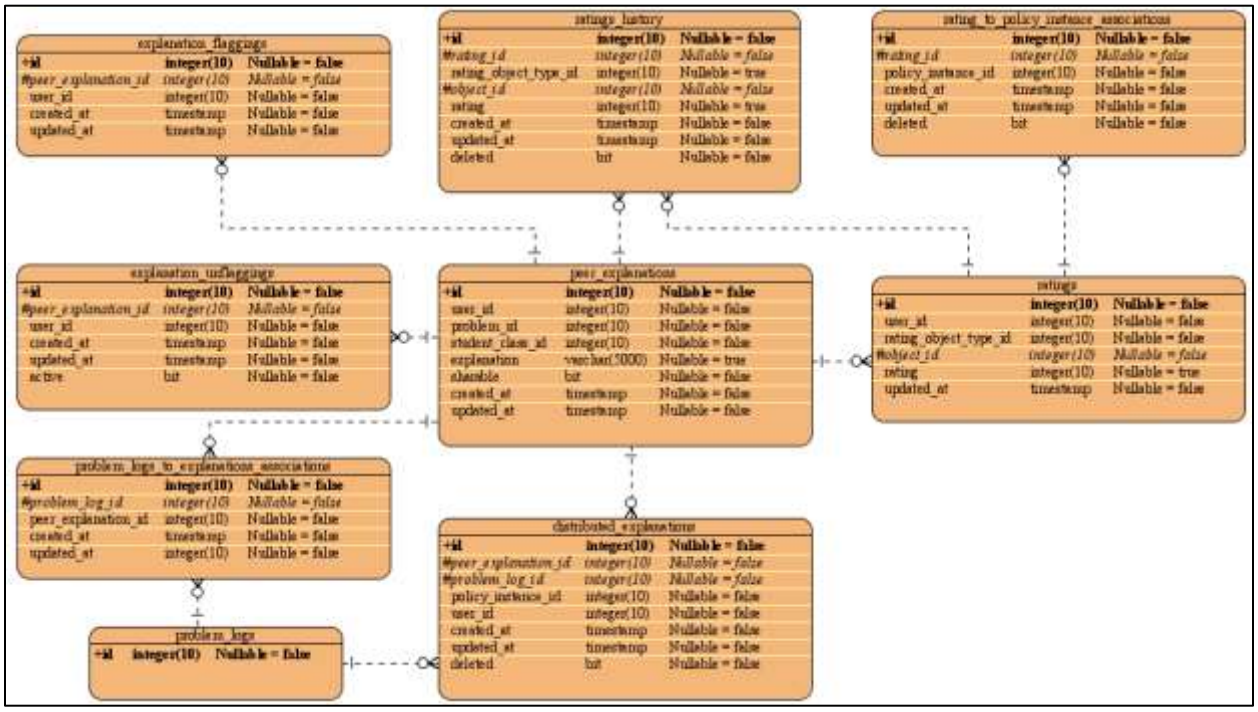


Figure 8.3.1. ER diagram highlighting the tables related to PeerASSIST data in the ASSISTments database.

peer_explanations

The peer_explanations table is where the student work is stored. Although there has been some debate over the correct terminology for the student's work, the database table name refers to them as explanations. This may differ from the terminology used in non-technical areas of this chapter.

Columns

1. user_id (integer) – The id of the user who created the work.
2. problem_id (integer) – The id of the problem that the work was created on.
3. student_class_id (integer) – The id of the student class which the student belongs to. This id is derived from the class that the given assignment was assigned for, since students can belong to more than one class.
4. explanation (text) – The student's work. This is the HTML characters including the text the student submitted as well as any markup characters and image or video links. It is limited to 5,000 characters, which was determined as a reasonable upper limit for the student's work. The limit is imposed to prevent an "infinite" amount of data from being stored, in the scenario where a student submits a large amount of content which is likely not a legitimate explanation. This event has already occurred, where students has tried to submit a large amount of text by repeated copying and pasting. This data is prevented from being inserted into the database so no other students are affected.
5. sharable (boolean) – A column to represent whether or not the student has allowed their work to be shared with other students. If the value is false, the work will not be considered for being redistributed to other students.

problem_logs_to_explanation_associations

The problem_logs_to_explanation_associations table is a table to maintain the lniking between the problem_logs table and the peer_explanations table. In other words, this table maintains the link to know what problem log a given student work was created for. It was unclear to developers, on the future usage of the peer_explanations table. Currently student work must be associated with a given problem log entry, however in the future that may not be true. Therefore a direct foreign key constraint was not added to the peer_explantions table. Instead an intermediate table (this one) was created with foreign keys to both the peer_explanations table and the problem_logs table.

Columns

1. problem_log_id (integer) – The problem log id associated with the student work.
2. peer_explanation_id (integer) – The id of the students work (peer_explanation) associated with the problem log.

problem_logs

The problem_logs table is an existing table in the ASSISTments database. This is where all student data on each problem is logged. Due to its importance, it was included in the ER diagram and the descriptions. There are over twenty columns in this table, however the columns are not important but the relationships with the other tables are. It is possible for an entry in this table to be physically and permanently deleted. In this scenario, the data is completely lost. This implementation was created prior to my involvement and a correct reimplementation will likely take over 100 hours or work. A simpler solution (although not entirely correct) would be to add a database trigger, which inserts any rows that are about to be deleted into a separate table for archiving. The reason for allowing for the deletion of data is if any errors, teacher mistakes, or other issues arise with a given assignment. It functions as a usability convenience. Teachers have the ability to delete the assignment and all the problem logs associated with it. This was unfortunately implemented as a real physical deletion in the database instead of a soft deletion. When a problem log is deleted any associated entries with student work are also deleted. Since the problem log no longer exists, the work associated with that problem log should also not exist. This is a minor detail to keep in mind when understanding the structure, constraints, and triggers of the database related to the storing of student work data. It is also important to know when doing data analysis on the multi-armed bandit policies. It is possible for an action that the policy has taken, to no longer exist. It may also cause reproducibility problems with data inconsistencies. These issues are not specific to the PeerASSIST system, and affect all data mining with ASSISTments data. It will be left as future work (for someone else) to fix this issue.

distributed_explanations

The distributed_explanations table is where all student work that was seen by other students is stored. If a student sees work done by another student, an entry is added to this table.

Columns

1. peer_explanation_id (integer) – This is the id of the student work that was shown to the current student. If there was no work to show, this column is left blank. This can be used to measure the intent to treat because it keeps track of when a student would have received another student's work if such work existed.

2. problem_log_id (integer) – This is the id of the problem log entry for which a student received another student's work for.
3. policy_instance_id (integer) – This is the id for the policy instance which was used to distribute the student work. More information on this is discussed in section 8.3. If an error occurs that is not handled by a specific policy instance, this id will be set to zero. Some examples of such errors include failure to connect to the PeerASSIST server, loss of network connectivity, or server errors where a policy instance could not be chosen. Currently there are no known programmer errors; therefore server errors are unlikely to happen other than timeouts related database queries at peak times.
4. user_id – The id of the user who received student work.

explanation_flaggings

The explanation_flaggings table stores if student work has been flagged as inappropriate. A student, teacher, or the system can flag student work. Once work has been flagged it cannot be flagged again by the same user. Flagged work will not be considered for redistribution, unless it has also been unflagged.

Columns

1. peer_explanation_id (integer) – The id of the student work being flagged.
2. user_id (integer) – The id of the user who flagged the student work. An id of zero means that the system has flagged the work as inappropriate.

explanation_unflaggings

The explanation_unflaggings table stores data on if an explanation, that had been previously been flagged, has been unflagged. Currently only teachers have the ability to unflag an explanation.

Columns

1. peer_explanation_id (integer) – The id of the student work that is being unflagged.
2. user_id (integer) – The id of the user who has unflagged the student work.
3. active (boolean) – Whether the unflag status is active. If the unflag status is active, the student work will be considered for distribution because it has been unflagged by a teacher after being flagged. The teacher's decision is trusted. If the unflag status is

inactive, it means that the explanation has not been unflagged (technically, it has been flagged, unflagged, and then flagged again by the teacher) and will not be considered for distribution.

ratings

The ratings table was a previously existing table in the ASSISTments database. Until now the ratings table had been basically not utilized. It seemed to be originally designed to support ratings on any type of content. Instead of creating another ratings table specifically for PeerASSIST, this ratings table was incorporated into the design. Reusing existing implementations when applicable is typically good software practice. Note that this ratings table holds data for ratings on content other the student work, which can easily be distinguished.

Columns

1. user_id (integer) – The id of the user who has rated the content.
2. rating_object_type_id (integer) – The id for the type of content being rated. Currently ids one and two are used. An id of one means that the content is a webpage and an id of two means that the content is student work.
3. object_id (integer) – The id of the content being rated.
4. rating (integer) – The rating of the content. Currently ratings must be an integer value. The previous implementation had stored ratings as integers. PeerASSIST currently stores three rating values. A value of -1 is stored if the content was rated as "bad". A value of 0 is stored if the content was rated and then that rating was cleared by the same user. A value of 1 is stored if the content was rated as "good". If desired, this column can be harmlessly converted from an integer to a decimal value for future use.

rating_to_policy_instance_associations

The rating_to_policy_instance_associations table stores what ratings were given from content distributed by a specific policy. This information is not recoverable without this table. Since students often do the same problem more than once, it is possible for students to receive peer student work multiple times on the same problem but distributed from different policies. The ratings table merely holds the rating on the content for a specific user. Therefore in order to know which policy a rating for content was generated from, this table is necessary. This

information is required for some of the experimental framework to function. As one option for data inclusion, a given policy instance may only use data that had been generated by the same policy instance. Therefore the policy instance, which the content was distributed from and a rating generated from, must be known. This table also provides a nice shortcut for some acquiring some simple statistics on ratings for each policy instance.

Columns

1. rating_id (integer) – The id of the ratings table entry.
2. policy_instance_id – The id of the policy instance.

ratings_history

The ratings_history table stores the history of any changes made to the ratings table. This is important for recovering data at specific points in time. Many decisions are made based on the ratings of student work at a given point in time. Since a rating can be changed any number of times, it is important to know what the ratings were at any given time. The ratings table merely holds the most updated version of the ratings and does not store prior values of ratings; therefore the ratings_history table is necessary. Every time a rating is changed a database trigger is used to create an entry in the rating_history table for the prior value of the rating before it was changed. This table stores the complete history of all ratings that have been changed. This means that future researchers will always be able to recover the ratings at a specific point in time.

Columns

1. rating_id (integer) – The id of the ratings table where the rating was updated.
2. rating_object_type_id (integer) – The type of content (same as the ratings table).
3. object_id (integer) – The id of the content being rated (same as the ratings table).
4. rating (integer) – The rating for the content (same as the ratings table).

8.3. PeerASSIST Server

8.3.1. Introduction

This section describes the internal processes of the PeerASSIST server. This server is used to receive requests for peer hints and choose which hints to redistribute. Section 8.3.2 outlines the high level process used to choose the hints that are distributed. This was built as a framework for running simultaneous experiments as scale, which use and compare multi-armed bandit algorithms. This was inspired by work done by Kohavi et al., and Bakshy et al., at Bing and Facebook respectively, for creating systems for doing experimentation at scale (Kohavi et al., 2013, Bakshy et al., 2014). The framework and bandit implementations are described in section 8.3.3. Currently the server consists of roughly 15,000 lines of code and is running error free.

8.3.2. Peer Work Choosing Process

When a student starts a problem an HTTP request is sent to the PeerASSIST server to choose peer work that the student will receive when the student reaches a score of zero for the problem. This request contains four parameters (class id, assignment id, problem id, and user id) which are necessary to determine what student work to choose to redistribute. NGINX (our load balancer) receives the request and forwards it to the PeerASSIST server (Reese, 2008). The activity diagram in Figure 8.3.2.1 shows the process of how student work is chosen.

The first step is to determine if the current student has already been mapped to existing peer work that was previously distributed to the current student. A student will already be mapped to peer work when the request comes from a student who is resuming the problem on another local storage location.

For better performance, we store information on the problem inside the browsers local storage. When a student resumes a problem from the same local storage location, no requests need to be sent to the server to retrieve information. All the information is stored on the student's machine and can be loaded directly from that. The local storage location will change if the student resumes the assignment on a different machine, different browser, or different mode (incognito/private).

It is possible for a student to start a problem and be assigned to peer work to see, then leave and resume the problem somewhere else. This student may or may not have seen the work, however

the work should not change when a student resumes progress or many experiments could be invalid because the work could change every time a student resumes his/her progress. In this scenario the peer work that gets distributed needs to be the same peer work that was distributed to the student the first time they requested it for that assignment and problem. The mapping for this is stored in the peer_work_user_experiment_mapping table inside the PeerASSIST database. By storing this information, information will stay consistent for all resume actions.

If a student has already been mapped to peer work, it is sent back to the student's client machine and no further processing is required by the PeerASSIST server. If the work had already been displayed to the student, it will continue to be displayed, if the work had not been displayed, then it will remain hidden until the student reaches a score of zero for the problem. When a student has not been previously mapped to peer work, a peer work needs to be chosen to be distributed. The next step in the process is to determine the multi-armed bandit policy to use to determine what peer work to distribute.

The second step is to determine if the current student has already been mapped to an existing multi-armed bandit policy. If the student has already been mapped t a policy, then the policy that the student is already mapped to is used to determine what peer work to send back to the current student. If a student is not currently mapped to a policy, then a policy is chosen from a list of available policies. The algorithm to determine which policy to choose can be changed as desired inside the PeerASSIST server code. Currently, policies are chosen randomly and uniformly from the available list of policies stored in the database.

Once a policy has been chosen, that policy will be used to choose the peer work that gets distributed. A simple high level framework for this process was designed to standardize this process. Since this platform is designed for experimentation, it is expected to have several policies implemented within the system. Implementing these policies and integrating them into the system is greatly simplified by designing an understandable, maintainable, and extendable framework to follow. This framework is described in section 8.3.3.

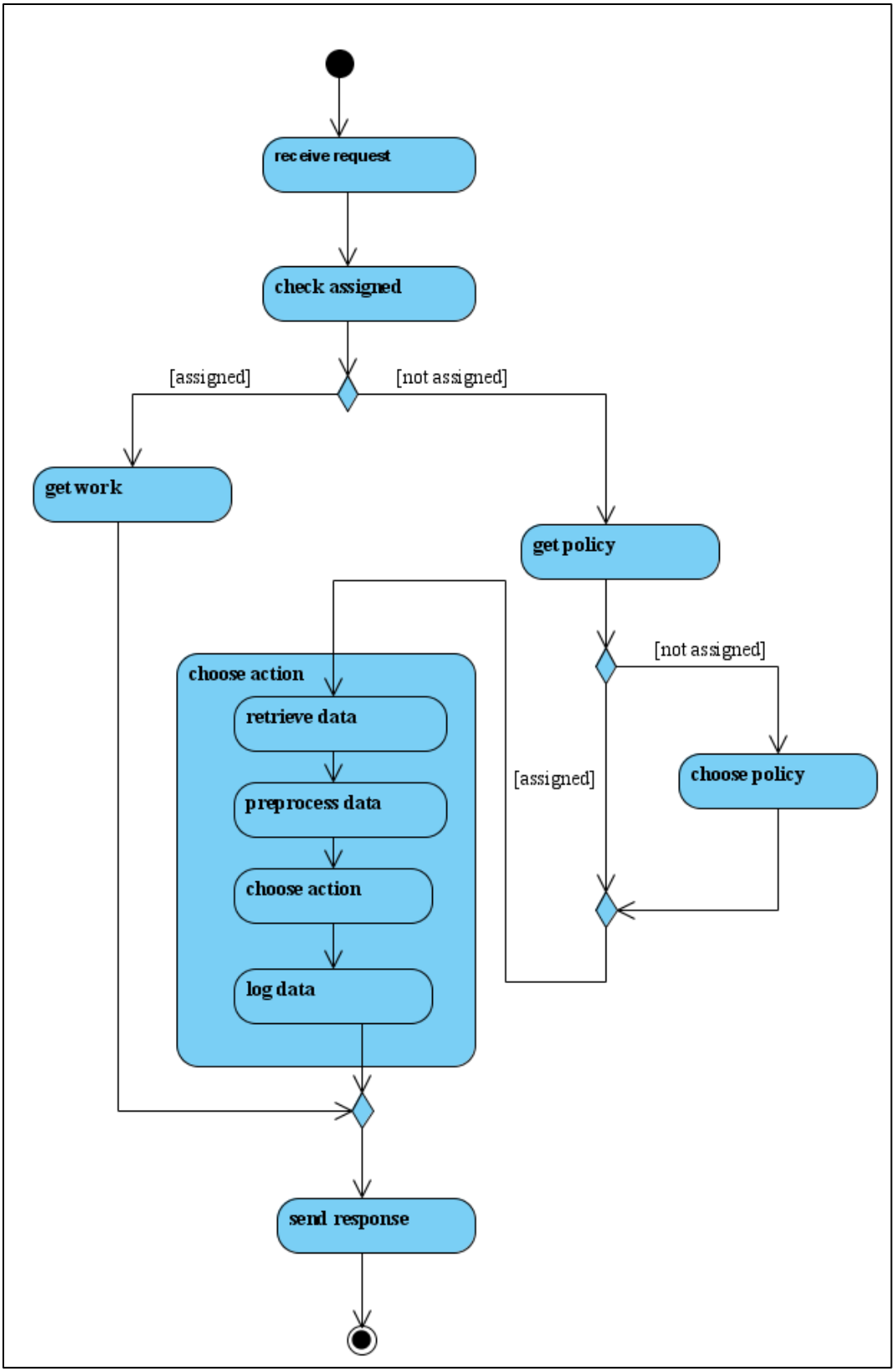


Figure 8.3.2.1. Activity diagram for the peer work choosing process

8.3.3. Multi-Armed Bandit Framework

8.3.3.1. Introduction

Multi-armed bandit algorithms have been used in several domains including educational system (Liu et al., 2014; Williams et al., 2016). In this section a simple framework is described for implementing multi-armed bandit policies inside the PeerASSIST server. This framework consists of four major processes, "Data Retrieval", "Data Preprocessing", "Action Choosing", and "Data Logging". These four processes are executes in sequential order as defined. Each multi-armed bandit algorithm will consist of their own "DataRetriever", "DataPreprocessor", "ActionChooser", and "DataLogger". An algorithm can reuse code from existing algorithms by extending the appropriate classes. The following sections describe these four processes in greater detail.

8.3.3.2. Data Retrieval

Data retrieval is the process where necessary data is queries from the necessary databases. This data will eventually be used by the bandit policy to determine which action (peer work) to choose. What makes this process difficult is that the necessary data is stored across multiple databases, where there is no good way to query both databases with a single query. The data comes from both the ASSISTments database and the PeerASSIST database. Therefore an elegant and simple custom middleware was designed and implemented to provide programmers an easy way to query across multiple databases.

An example of a real query shown in Appendix A is used to explain how the middleware works. Notice how this query contains variables such as ":userId", ":problemId", ":policyInstanceId", ":classId", ":piece1", and ":piece2". These variables embedded in the query can be replaces with the appropriate values when working with the Spring Framework (Johnson et al., 2004). These variables are replaced with a simple programmatic substitution before executing the query. What makes this implementation different than a typical implementation is that the substituted values for variables such as ":piece1", and ":piece2" come from the results of another query, which runs on a separate database. Note that variables such as ":userId" and ":problemId" expect a single value to be substituted, however variables such as ":piece1", and ":piece2" expect a list of integers to be substituted in this example.

The middleware provides a mechanism to run a query (child query) and substitute the results from that query into a variable of another query (parent query). The whole query is broken up into a hierarchical tree structure of query pieces. Figure 8.3.3.2.1 shows the tree structure for the query in Appendix A. Each node at a given depth level of the tree can execute independently of the other nodes at the same depth level. The middleware executes all nodes at the same depth level in a separate thread for faster parallel execution. A given node in the tree can only be executed, once all of its children have finished executing, so that the results of the child queries can be substituted into the parent query. The main purpose of this design is that it allows for nodes in the tree to run on separate databases. An addition feature is that subqueries can run in parallel execution, although this is typically not an optimal way to run the queries.

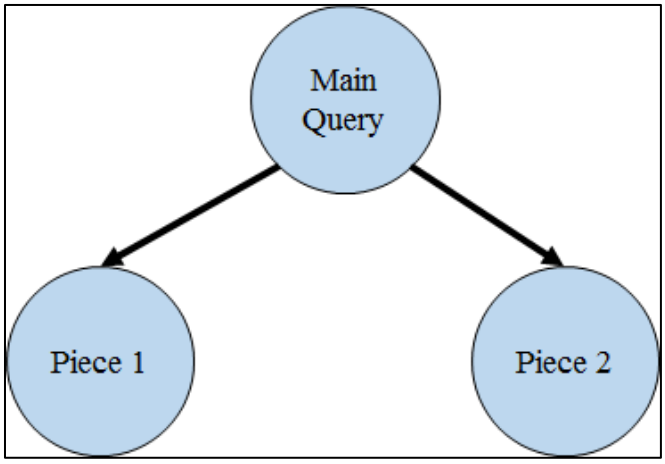


Figure 8.3.3.2.1. Query Execution Tree Hierarchy

Note that there are two major limitations for this middleware. One limitation is that joins across multiple databases are not supported. This feature has not yet been implemented. Another limitation is that there is a limit to the number of characters that a query can contain. Therefore if there are a large number of rows in the results of the subquery, substituting all of those characters into the parameter for the parent query can result in the total character count of the query to exceed limitations. These limitations should be kept in mind when using the middleware.

The result of the query is in in the form of one-row-per-peer-work and consists of the dependent measures for a peer work. These measures include student rating, teacher rating, and next question correctness statistics. This data is in a raw format, which was directly obtained from the database. Various multi-armed bandit policies may require the data to be in a different formats

for use. To make the code modular and easier to maintain, the data format that the bandit policies require is maintained separately from the data format directly obtained from the database. An intermediate preprocessing step is used to convert the raw data into a format that the multi-armed bandit policies will use. This design allows for data retrieval and query generation to be separate from the bandit policies.

Currently no contextual features are used by the implemented multi-armed bandit policies. However the data retrieval and preprocessing for these features has already been done. When these features are needed they will be available. Features that are currently available include student features, school features, problem features, and explanation features. These features are described in section 8.3.5.

These features do not need to be kept updated in real-time and therefore a caching mechanism was built for optimized performance. The original data for these features comes from the ASSISTments production database. Once this information is retrieved for the first time it is stored in an equivalent table on the PeerASSIST database. Storing the information on the PeerASSIST database will be faster than querying the production database. One reason is because the ASSISTments production database is typically busy handling requests from all the users of the ASSISTments website. The database and hard drive are more heavily utilized than the PeerASSIST database. A second reason why storing this information on the PeerASSIST database is faster is because the data will be located on the same physical machine. This means that there will be no network latency when transferring data from the database to memory. Executing multiple queries for features on the same database may cause performance to degrade to an unacceptable level. Caching the data on the PeerASSIST database is not just recommended but most likely required in order to function properly.

Data for each feature is stored in a corresponding table in the PeerASSIST database. Each table has its own cache expiration timestamp for when the data is no longer valid. When feature data is requested, the expiration timestamp for the table it is requested from is compared to the current time. If the data is still valid, the data will be queried from the PeerASSISt database. If the data has expired or does not exist, the data is queried from the ASSIStments production database and then cached in the PeerASSIST database. Queries for each feature are executed in parallel to optimize performance. Once all queries have completed the server will continue to the preprocessing phase.

8.3.3.3. Data Preprocessing

Data preprocessing is an intermediate step between retrieving the raw data and using the multi-armed bandit algorithms. This step is essentially the "bridge" connecting the algorithms to the data. PeerASSIST was designed so that the data retrieval process and multi-armed bandit algorithms could be implemented separately from each other. The reasons for this is because both processes require completely different knowledge to implement and are best to be maintained separately. The algorithms may work best from data in a specific form, which may be completely different from the raw data format returned by a database query. Instead of trying to modify the data format from one of the processes separately, the data preprocessing step handles it. Since the multi-armed bandit algorithms require data in order to function correctly, this intermediate step is required to transform the raw data retrieved from the databases into data that can be used by the multi-armed bandit algorithms. The raw data retrieved from the database in in the form of one row per student work. Dependent measures are aggregated into counts. Most multi-armed bandit policies expect the data to be in the form of a history of action-reward pairs.

Table 8.3.3.3.1, 8.3.3.3.2, and 8.3.3.3.3 show an example of the raw data from executing the database query in Appendix A. Table 3.3.3.1 shows the first set of columns with information on the given student work. Table 8.3.3.3.2 is an extension of the three rows in Table 8.3.3.3.1 and shows all columns related to the student rating. Table 8.3.3.3.3 shows the columns related to how well students did on the next question after being presented with the given peer hint. This shows a few examples of student work submitted for problem 1214705 and only includes columns important for this example. The column for the actual student work is excluded. It is excluded because it is currently irrelevant for determining the quality of the explanation and does not format well. Teacher rating is one of the three dependent measures; however it is excluded because all rows do not have any teacher ratings.

id	user id	sharable	student class id	problem log id	problem id
274382	405949	t	44004	188518935	1214705
220571	406091	t	44004	187323439	1214705
211711	405868	t	44004	187091501	1214705

Table 8.3.3.3.1. Example raw data for student work

id	distributed count	student up votes	student down votes	student cleared votes
274382				
220571	2			
211711	5	1	0	0

Table 8.3.3.3.2. Example raw data for student work showing the student votes

id	next question correct count	next question incorrect count	next question count	finished count
274382				
220571	1	1	2	2
211711	4	1	5	5

Table 8.3.3.3.2. Example raw data for student work showing the next question correctness

In all three tables, the unique id for an instance of student work is represented as the "id" column. In Table 8.3.3.3.1, the "user id" column is a unique id for the user who created the work. The "sharable" column represents a Boolean value for whether or not the student is allowing their work to be shared with others. "student class id" represents a unique class id for the class that student was in, which was assigned an assignment that the student showed work for the problem on. "problem log id" is the unique id for the student-problem instance where the student submitted work and "problem id" is the problem the work was submitted on. This information is necessary for searching for valid work to redistribute.

Table 8.3.3.3.2 shows the vote counts for each student work. The "distributed count" is the count of the number of times the given peer hint has been seen by other students. "student up votes", "student down votes", and "student cleared votes" show the count of the number of up votes, down votes, and cleared votes respectively.

Table 8.3.3.3.3 shows how well students do on the next question after seeing the given peer hint. It includes counts for the number of times students started and finished the next question. Out of all the students who have finished the next question, counts are given for the number of students who have answered the next question correctly and incorrectly.

This raw data is then transformed into data more applicable for use by multi-armed bandit policies. From this data a "fake" action-reward history is generated. Since the rewards (teacher rating, student rating, and next question performance) are asynchronous, it is important that only aggregated information is used. Although the total reward for a given instance of student work is made up of the three dependent measures, a single reward instance is not. Each reward function, functions separately.

To transform raw counts into a sequence of action, each reward function is transformed separately and stored as three separate sequences of actions. For each student rating, the counts of up votes and down votes are retrieved. Missing votes are derived by subtracting the combined counts of the up and down votes from the distributed count. Actions are created for each up, down, and missing vote with rewards on the scale of [0, 1]. Down votes are given a reward of zero and up votes are given a reward of 1. Missing votes are assigned a value of -1, which is then handled by the specific multi-armed bandit policies. Typically missing values are replaced with either an average of the up/down votes or a default missing value. Teacher rating and next question performance are derived in similar ways as the student rating. The result of the transformation is shown in Table 8.3.3.3.4, for the three instances of student work and the two reward functions (teacher rating excluded). The data is stored programmatically in this format, which is then used as the input action-reward history for the multi-armed bandit policies.

id 274382		id 220571		id 211711	
student rating	next question correctness	student rating	next question correctness	student rating	next question correctness
		-1	1	1	1
		-1	0	-1	1
				-1	1
				-1	1
				-1	-1

Table 8.3.3.3.4. Transformed data into an action-reward history of each instance of work and each reward function.

8.3.3.4. Action Choosing

The next step in this process is to choose an action (instance of student work) to redistribute. This process is runs one of the multi-armed bandit policies that was chosen and returns the result. The implemented policies and how they work is described in section 8.3.4. The final result of this step is a single action, which will be sent to the student.

8.3.3.5. Data logging

Data logging is the final step in this process. Depending on the policy which was run, there may be specific information related to that policy which needs to be logged. The final step in this process is to log any necessary information before responding to HTTP request. Currently the only information that all policies are logging is which instances of student work were chosen. This information is important, because it is good to know at what time the choice was made. When a student sees a peer hint, that information is logged in the ASSISTments production database at the time the student saw the peer hint. That time can be very different than the time that the peer hint was chosen to distribute to that student. To be able to know the sequence when peer hints were assigned, the information must be logged at the time of the assignment by the PeerASSIST server. A common question can then be answered which is, "How many alternative choices were there at the time this peer hint was chosen for redistribution?" It was not possible to answer this question without logging the timestamps of hint assignment server-side.

8.3.4. Implemented Policies

The multi-armed bandit problem is when a person goes into a casino to play slot machines with multiple arms where each arm has a different payout rate. The problem is that the person needs to determine which arm to pull that gives the best payout rate, without any prior knowledge, to maximize their profits (or minimize their losses). The arms of the slot machine are considered bandits because they are essentially stealing money from the person (Lai & Robbins, 1985). The original story is slightly unrealistic in the current day, since a real casino typically has several rows of slow machines with a single lever (arm) to pull (not multiple arms), where all the payouts are known to be the same (not different). A slightly more accurate interpretation can be to think of a casino with a row of slot machines where each slot machine has a single arm and a

different payout rate. In this example each slot machine can be considered a bandit and the object would be to play the slot machines in a way to maximize profits (minimize losses).

This story has a major application in research in reference to experimental design. Herbert Robbins is given credit for the origin of the multi-armed bandit experimental design strategy, although it was not named as such in his paper in 1952. At that time, the design of experiments had the number of samples fixed before the experiment was actually run. This was because statistics were not as robust back then and working with a varying number of samples was more difficult. Robbins introduced the idea of sequential design of experiments where the sample size is not predetermined and is a function of the samples themselves (Robbins, 1952). In reference to the multi-armed bandit story, the number of populations is equivalent to the number of arms that a slot machine has. A sample from a chosen population is analogous to a pull on a chosen arm of a slot machine.

The problem of sequential design was proposed as the following in the 1952 paper. There are two or more normally distributed populations with unknown means and variances where the goal is to estimate the difference in means. The two issues involved in sequential design are how many samples to use and which groups to assign the samples as they are collected at different points in time. Another way to think of it is what group to assign each sample as it is collected and when to stop collecting samples. The paper refers to the time at which the samples are collected as stages, suggesting that the samples are collected in groups and not one at a time. The question is then proposed for how many samples should be collected for each stage (Robbins, 1952). The questions of which populations to choose from at a given time and how many sample to choose are difficult questions to answer. The following quote by Peter Whittle in a discussion of John Gittins' paper shows how difficult the problem is.

"As I said, the problem is a classic one; it was formulated during the war, and efforts to solve it so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany, as the ultimate instrument of intellectual sabotage." (Peter Whittle in Gittins, J. C., 1979)

Over 30 years later, Lai and Robbins proved an optimal solution for the problem. The optimum solution multi-armed bandit problem starts with a set of populations that can be sampled from sequentially. The problem becomes how to optimize the greatest expected value of the sum of all the individual sample values. This problem is also equivalent to minimizing the total

expected regret. A key word is "expected" since the best choice is not known at the time, it is only estimated. The total regret is the sum of the regret for each choice for all samples. This problem statement makes a good analogy to playing slot machines because when you play slot machines you are not playing to maximize profits but really playing to minimize losses (because you're highly likely to lose). Lai and Robbins proved that the optimum asymptotic convergence to the lower bound on regret is O(log(n) and how to get a set of allocation rules that achieves the optimum convergence in their paper.

Asymptotic loss can be thought of the average loss you will receive after an infinite number of tries given that you do not know the optimal choice. Robbins gives a great example of asymptotic loss in his 1952 paper (Robbins, 1952). In his example there are two coins, each with some probability of heads with the goal to get the most heads. One set of rules is to randomly pick which coin to flip regardless of the outcome, and another set of rules is to switch coins when a tails comes up. The asymptotic loss per coin loss is based on the probability of heads for both coins. In the worst case, randomly choosing coins converges to an asymptotic loss per toss of 0.5 (50 cents per toss) and switching coins on tails converges to a worst case asymptotic loss of 0.17 (17 cents per toss). This happens in the situation where Coin A has a probability of heads = 0.58 and Coin B has a probability of heads = 0. Using the second set of rules, after an infinite number of tosses, if coin A is tails then coin B will be tried next, even though coin B has no chance of ever getting heads. In the case where both coin A and coin B have the same probability of heads, there will be an asymptotic loss of 0. In the case where coin A has a probability of heads = 1 and coin B has a probability of heads = 0, there will also be an asymptotic loss = 0. In this scenario, the first coin can be randomly chosen from coin A or coin B. If coin A is chosen then coin A will always be chosen since it will never have a tails value and therefore the coin choice will never change. If coin B is chosen, the value will be tails which results in the coin choice to be changed from B to A. As the number of tried approaches infinity (or even after the second try in this case) coin A will always be chosen and there will be no difference between the best choice and the current choice (coin A is both). Therefore the asymptotic loss is 0.

The algorithm uses confidence bounds to efficiently determine which population is the best. More specifically, their algorithm compares the upper confidence bound (UCB) of the "loser" populations to the confidence interval of the "leader" population. The leader population is

determined by the population that has the highest mean among all populations that have been sampled at least δn times, where n is the total number of samples and $0 < \delta < 1 / k$ where k is the number of populations. Delta is a fixed chosen number of samples for a given stage, where samples are taken in stages (not necessarily one at a time). The confidence interval calculation is also dependent on knowledge of the underlying distribution, which may be unknown (Lai & Robbins, 1985).

There are several advantages to using sequential design. Sequential design allows for an experiment to use a fewer number of samples and allows for the experiment to end earlier. Both time and sample (often people) resources are saved. Another advantage is that if one of the conditions is detrimental, it can be avoided. This is often the case with medical trials, where it is found that a treatment is harmful. There is no reason to keep giving people harmful treatment and it is unethical. Sequential design will prevent people from receiving harmful treatment. A disadvantage of sequential design is that constantly performing significance testing throughout the course of the experiment can result in a large type-I error of falsely rejecting the null hypothesis. This can be prevented with various forms of correction, but usually is not (Wegscheider, 1998).

The multi-armed bandit problem relates to the problem in our own domain of what tutoring content to assign students. Here the tutoring content to assign is analogous to a bandit or a slot machine arm. Each tutoring strategy represents its own population distribution, where the distribution contains the number of students for whom the tutoring is effective and the number of students for whom the tutoring is ineffective. There are many different types of tutoring strategies to choose from, each having a different level of effectiveness, just like each slot machine arm has a different payout rate. Assigning a tutoring strategy to a student is the same as sampling the population.

Some differences exist between the optimum algorithm proposed by Lai and Robbins and the current design of the experiment to assign tutoring to students in ASSISTments. Lai and Robbins represent their problem of sampling in stages, where each stage has some predetermined number of samples from a fixed group of populations. They also never explicitly define a delta parameter. In our domain, we expect to receive an increasing number of tutoring content in semi-real-time. Therefore our number of populations is constantly increasing during the experiment and is not fixed. We also will be sampling (assigning tutoring content to students)

individually and not in stages. The confidence interval calculation done by Lai and Robbins assumes the underlying distribution of the populations is from a common known distribution. In our experimental design, we cannot make such assumptions.

There have been several variations of the multi-armed bandit problem since it was first introduced in 1952. These variations include different types of reward functions (continuous, discrete, and Bernoulli), number of arms (finite or infinite), number of pulls (fixed horizon or horizon free), whether or not the reward functions can change over time (stationary or non-stationary).

Berry et al. explored the issue of bandits with an infinite number of arms (Berry et al., 1997). In their work, they focused on an infinite number of Bernoulli bandits with an infinite time horizon. They used the same idea as Robbins, which is to switch "arms" when a failure is encountered and investigated three different strategies based on this idea. For their proofs they assume that the reward probabilities are uniformly distributed among the infinite arms. They proved that for any strategy (given the previously mentioned set of assumptions), the best possible lower bound is $\sqrt{\frac{2}{n}}$ for the asymptotic failure proportion. This means that for some number of samples (n) the best possible proportion of failures to successes will converge to $\sqrt{\frac{2}{n}}$.

Their first strategy is called the 1-failure strategy, which is a variation of Robbins' example with two coins that extends to an infinite number of arms. The 1-failure strategy starts with the first arm available and then continues to use that arm until it produces a failure. A new arm is then used until it produces a failure. The algorithm continues forever as it keeps trying the same arm until failure and then switches to a new arm. Once an arm fails it is never used again. They proved that this strategy does better than any other algorithm with a k-failure strategy, where k is the number of failures required to switch arms. In other words, the best k for a k-failure strategy is when k=1. For the 1-failure strategy the asymptotic failure proportion is on the order of $\frac{1}{\ln(n)}$.

Their second strategy is called an m-run strategy. The m-run strategy is a modification to the 1-failure strategy, where the arm is sampled until failure or until m successes. If an arm succeeds m times, it will be used forever, if no arm succeeds m times after m arms have been tried, the best arm will be used for all future tries.

Teytaud et al. noted the deficiencies of the algorithms presented in Berry et al. (Teytaud et al., 2007). They noted that those algorithms only work with the assumption of a normally distributed reward function probability among the bandits and an infinite amount of time. When these assumptions are violated, their algorithms will perform poorly. Teytaud et al extend the algorithms in Berry et al to work without knowing the number of time steps. Teytaud et al claim that the algorithms in Berry et al depended on the number of time steps. However in Berry et al, they assume the number of time steps to be infinite as quoted, "We consider the horizon *n* to be infinite" (Berry et al., 1997). To clarify what appears to be a contradiction, Teytaud et al. use the concept of "anytime". An anytime algorithm is where the upper bound on expected regret is the same for all 'n'. The algorithm proposed by Berry et al. was proven to be optimal as an $n \rightarrow \infty$. However the algorithm does not converge to the optimal minimum regret for small n. Teytaud et al., propose an algorithm that is both optimal and for any number of 'n' sample sizes.

It is important to specifically look for the "anytime" property, since many of the older proofs and algorithms were merely concerned with asymptotic optimal convergence an $n \rightarrow \infty$. Although many algorithms claim to be optimal, they may require a large number of samples to converge to the minimum regret. In our case (experiments in ASSISTments), our number of samples will be small and therefore we need an algorithm that not only is optimal, but is also optimal for a small number of samples.

Teytaud et al. decided to separate the problems proposed by Berry where the uniform distribution is in the range [0, 1] or [0, ε], where $\varepsilon < 1$. They call these problems the Easy ManAB (EManAB) setting and the Difficult ManAB (DManAB) setting. They prove that the expected failure rate for any algorithm for these settings is $O(1/\sqrt{n})$ and $O\ N^{-1/4} / c$ respectively. Since there are two types of problems, Teytaud et al. provide an algorithm for each problem type. In their paper, they analyzed the FAILURE, FAILUCB AND FPU algorithms for the EManAB case and the MUCBT algorithm for the DManAB case. They show that the FAILURE, FAILUCB, and FPU algorithm worked well for the EManAB problems where the number of samples is infinite or roughly equal to the square of the number of populations. They also showed that the MUCBT algorithm performed all the other algorithms for DManAB problems and performed just as well for EManAB problems (Teytaud et al., 2007).

Wang et al. also explored the problem of when the number of arms is larger than the number of experiments (Wang et al, 2009). They made the assumptions that the reward function is

stationary. The main contribution of Wang et al. is that although the algorithms proposed by Teytaud et al. were optimal given any number of samples, they were not generalizable to non-Bernoulli reward distributions (they assumed uniformly distributed Bernoulli rewards). They modified the UCB-V algorithm, which they called UCB-V (∞), to work with an infinite number of arms. Typically the UCB algorithm family is not applicable to use for an infinite number of arms because it first executes an explore phase and then an exploit phase. Since it is impossible to explore an infinite number of arms, the UCB algorithm would not be applicable for the case of an infinite number of arms. However Wang et al. modified the UCB-V algorithm to switch between the exploring and exploiting phase. Their first algorithm uses a known fixed horizon, which they called UCB-F (fixed horizon). One of the drawbacks of their UCB-F algorithm is that the number of populations to explore must be fixed based on 'n' at the start of the algorithm. Therefore the algorithm will not work in the case where the time horizon is infinite or not known. To solve this problem Wang et al introduce UCB-AIR (Arm-Increasing Rule) to handle the case where time is infinite.

The UCB algorithm family started with Robbins and was later improved upon by Auer et al. Auer et al improved upon the asymptotically optimal regret bound first introduced by Lai and Robbins and simplified in Agrawal (Agrawal, 1995). They created an allocation strategy that has logarithmic regret for any number of samples instead of only converging asymptotically, which they called UCB1. Although UCB1 is asymptotically optimal, the constant is large causing the algorithm to perform with a larger average regret. UCB2 improves upon UCB1 reaching the best constant term possible. Auer et al also created a version of UCB1 for a normally distributed reward function called UCB1-NORMAL and a version of ε-greedy that improves the bound on regret by decreasing the exploration as time increases. They later improved upon the UCB1 allocation policy by adding an upper confidence bound on the variance. This algorithm (called UCB1-TUNED), performed empirically better than the other UCB algorithms, however a bound on the regret was not proven. UCB1-TUNED performed roughly as well as of ε_n-greedy, although ε_n-greedy can perform badly as time increases when a good rate of decay is not chosen. The performance of UCB-TUNED was empirically proven in (Auer et al., 2002).

Audibert et al. continued with the idea of using variance estimation to improve the performance of the UCB-TUNED algorithm (Audibert et al., 2007). They recognized that using empirical

variance estimation in their algorithm can allow for suboptimal arms to be detected sooner and lower the regret. Since the regret of UCB-TUNED was not previously proven, Audibert et al. created their own version of a UCB algorithm that used variance estimation proved that the regret is asymptotically logarithmic. They experimentally compared UCB1 to UCB-V, finding that UCB-V performed better. A point they noted is that when a suboptimal are has an average reward close to the average reward of the optimal arm, it becomes difficult for the UCB1 and UCB-V algorithms to determine the optimal arm. In this case it may choose a suboptimal arm and take a long time to correct to the optimal arm. This will not be a problem in the context of applying the algorithm in ASSISTments, since the exact best arm (tutoring strategy) is not important. Finding an optimal subset of arms will be sufficient.

UCB-AIR extends UCB-V by adding the decision on when to choose to sample from a new arm in the case where there are an infinite number of arms. This decision allows the UCB-AIR algorithm to be non-uniform where typical UCB algorithms are uniform. Figure 8.3.4.1 from (Wang et al., 2009) shows how the algorithm works.

UCB-AIR (Arm-Increasing Rule): given parameters μ^* and β of (1),

- At time n, try a new arm if

$$K_{n-1} < \begin{cases} n^{\frac{\beta}{2}} & \text{if } \beta < 1 \text{ and } \mu^* < 1 \\ n^{\frac{\beta}{\beta+1}} & \text{otherwise: } \mu^* = 1 \text{ or } \beta \geq 1 \end{cases}$$

- Otherwise apply UCB-V on K_{n-1} drawn arms with an exploration sequence satisfying

$$2\log(10\log t) \leq \mathcal{E}_t \leq \log t$$

Figure 8.3.4.1: UCB-AIR Allocation Strategy (Wang et al., 2009)

The variable μ^* is the best possible mean reward currently known in range of uniform probability distribution [0, 1] and β is the distribution shape parameter for the choices. The following example shows how to calculate μ^*. Consider the three slot machines below with different chances to win and different payout rates. μ^* can be calculated by finding the maximum average payout of the three machines.

Example

Slot machine 1 = 90% chance to win 50 coins

μ_1 = 0.90 * 50 = 45

Slot machine 2 = 80% to win 60 coins

μ_2 = 0.80 * 60 = 48

Slot machine 3 = 50% chance to win 70 coins

μ_3 = 0.50 * 70 = 35

μ^* = MAX(μ_1, μ_2, μ_3) = MAX (45, 48, 35) = 48

The β parameter is harder to describe, but represents the shape of the probability distribution among the machines. Together the α (not shown in the parameters for UCB-AIR) and β parameters, define the shape of the distribution. For example if alpha and beta parameters were both equal to one, then the distribution would be uniform for all the machines. This would mean that all machines have an equal chance of having a probability to win between [0, 1]. To get an idea on what distributions that different values of alpha and beta produce, the link http://www.xycoon.com/beta.htm , provides a way to try out and plot the probability density function for different values of alpha and beta. When beta is less than one, this would indicate several near optimal-arms.

Wang et al. never mention where the value of β comes from in their work, therefore it can be assumed that either β is known prior to the experiment or β can be estimated from the estimated payout rates for each machine. Once again, consider the previous example where there were three slot machines with an estimated mean reward of 45, 48, and 35 respectively. These values were estimates obtained from trying each machine some number of times. β can be estimated by applying equations 8.3.41, 8.3.4.2, and 8.3.4.3 using the estimated variance of the machines.

Equation 8.3.4.1[9]. $$\mu = \frac{\alpha}{\alpha + \beta}$$

Equation 8.3.4.2[9]above. $$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2 (\alpha + \beta + 1)}$$

Equation 8.3.4.3[10] $$\beta = \frac{(\sigma^2 + \mu^2 - \mu)(\mu - 1)}{\sigma^2}$$

[9] http://mathworld.wolfram.com/BetaDistribution.html

[10] http://stats.stackexchange.com/questions/12232/calculating-the-parameters-of-a-beta-distribution-using-the-mean-and-variance

Note that the μ used in the examples and the UCB-AIR algorithm refers to the mean payout of a single machine. The μ variable used in the Beta distribution is equivalent to the mean of the mean payouts for all the slot machines. Since the mean payouts of all of the slot machines and their variance are not known, $\sigma^2$and μ both have to be estimated from the data to obtain a β parameter to use in the UCB-AIR algorithm. The equations for the estimated means and variances of a single slot machine are shown in Figure 4. The mean is simply the sum of all the reward values divided by the number of plays. The variance is simply the sum of the squared difference between each reward and the average reward for a given machine. To calculate the mean and variance of all the slot machines, just apply the same formulae using the overall mean of all the machines and the means of the individual machines.

UCB-AIR starts by determining if a new arm should be used. For the very first pull, there is no data to estimate the parameters from, but it should be obvious that a new arm should be tried. For all other pulls a new arm should be tried in two cases. In the case where β and $\mu^*$are less than 1, a new arm should be tried if the number of arms tried is less than $n^{\frac{\beta}{2}}$. In the case where either β or $\mu^*$are greater than or equal to 1, a new arm should be tried if the number of arms tried is less than $n^{\frac{\beta}{\beta+1}}$. This rule allows for the UCB-AIR algorithm to work for an infinite or indeterminate number of arms. If there are no more arms to try at time n, then the algorithm can use UCB-V (see equation 8.3.4.4, Wang, 2009) for that time step.

Equation 8.3.4.4. $$B_{k,s,t} \triangleq \bar{X}_{k,s} + \sqrt{\frac{2V_{k,s}\varepsilon_t}{s}} + \frac{3\varepsilon_t}{s}$$

X_k = empirical mean for action k

V_k = empirical variance for action k

s = samples for action k

$B_{k,s,t}$ = upper confidence bound for action k at time t

When the UCB-V algorithm is run on the arms that have already been tried, the exploration sequence must satisfy the following inequality $2\log(10\log(t)) \leq \varepsilon_t \leq \log(t)$. In the inequality, ε_t represents the exploration sequence, which must be non-decreasing, and t represents time or the sample number. If the exploration sequence should not be too small or too large in order to maintain minimal regret. The larger ε_t is, the more the UCB algorithm will explore. A simple

way to find a valid exploration function set ε_t equal to the upper or lower bound and play with values of the constant to multiply the exploration sequence by. Since you may want exploration sequence to decrease over time, the 't' variable can be used in the sequence as well. Some graphs are shown below in Figure 8.3.4.1 and 8.3.4.2 for simple valid exploration sequence functions. The first graph compares 0.5 * log(t) to the upper and lower bounds. The second graph compared the average of the upper and lower bounds to the upper and lower bounds. From looking at the graphs, it is obvious that something looks really wrong. For values of n < 1000, the inequality $2\log(10\log(t)) \leq \varepsilon_t \leq \log(t)$ cannot be satisfied because $2\log(10\log(t))$ is greater than log(t). I have no explanation for why this is the inequality used since it does not work! The alternative solution is to use either log(log(t)) (lower bound suggested by Wang et al.) or log(t) (value suggested by Audibert et al.) to substitute ε_t for.

The exploration sequence is then incorporated into the bias sequence below. Whichever arm maximizes the formula below and has the highest upper confidence bound is chosen. A point to note is that although the term upper confidence bound is used for conceptual meaning. The number can be larger than 1. In fact the number representing the upper confidence bound can be infinitely larger than 1. Consider the example where at sample number 1 million, a new arm is chosen. Since this is the first time the arm is pulled, the term log(1,000,000) or log(log(1,000,000) is plugged into the exploration sequence, depending on which exploration sequence bound is chosen. This number will be much greater than 1. As $n \rightarrow \infty$, this number will also approach infinity.

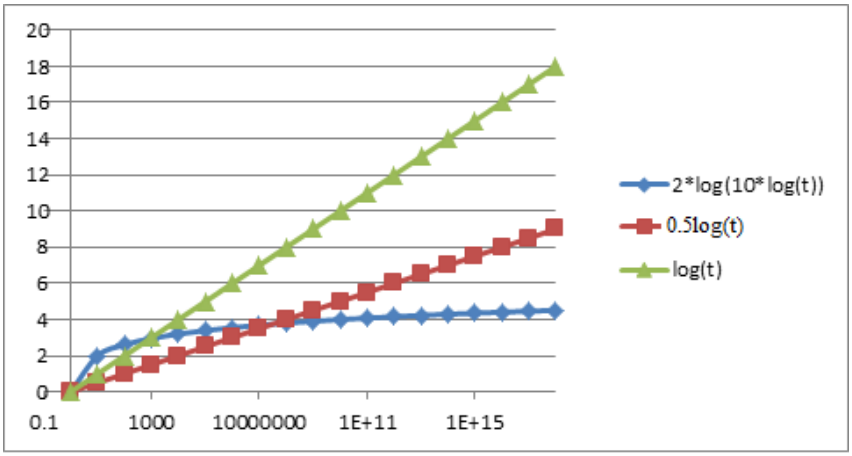


Figure 8.3.4.1: Red = 0.5log(t)

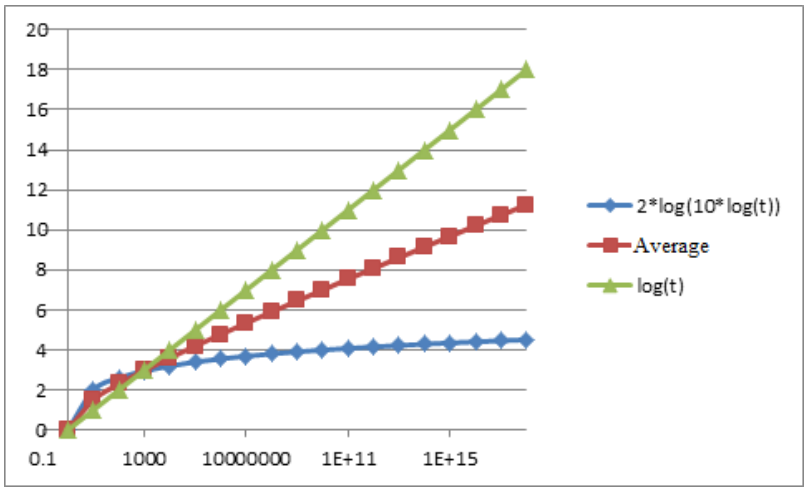


Figure 8.3.4.2: Average of the Bounds

For our implementation of a multi-armed bandit algorithm our "arms" or population will be the number of tutoring strategies available for a given problem. Since the system is designed to work in real-time, the number of tutoring strategies will be increasing as the experiment runs. Therefore we do not have a finite set of tutoring strategies. We must use a strategy that works with an infinite set of arms, although I prefer the word "indeterminate" instead of "infinite".

A semi-uniform algorithm will also be needed for the same reasons mentioned above. A semi-uniform multi-armed bandit algorithm alternates between the explore phase and the exploit phase. This can be seen as alternating between a tutoring strategy that is known to be good and newer strategies that have not been evaluated yet. A uniform approach first executes an explore phase and then an exploit phase. A uniform approach is not applicable to our situation because our tutoring strategies are not fixed at the time the experiment starts. Therefore using a uniform strategy would ignore all strategies that are created after the explore phase is complete. The algorithm we use must alternate between the two phases so that tutor strategies that are created later in time still have the possibility to be considered.

In our scenario the "arms" of the slot machine are our tutoring strategies, therefore the "pulls" are the assignments of tutoring to students. A single assignment can be thought of a single "pull" or a single sample of the population (tutoring strategies). The exact number of tutoring assignments is unknown, since the number of students who receive help is unknown. Therefore an anytime implementation of a multi-armed bandit algorithm is needed. An anytime algorithm

means that the number of "pulls" or samples is unknown, where a horizon free implementation means that the number of samples is infinite.

An optimal algorithm is also desired since a non-optimal algorithm choice is unethical to use in our context and may not work correctly. An optimal algorithm guarantees the fastest rate of asymptotic convergence to the optimal choice. In addition to the algorithm being optimal, it must also be anytime. An optimal algorithm merely guarantees that the asymptotic loss per sample will approach the minimum loss as $n \rightarrow \infty$. In our environment our 'n' is small, therefore we need the regret to be low throughout the samples and not just converging to optimality. An anytime algorithm has the property where the regret is uniform throughout the samples.

There has been some work done with non-stationary reward functions. Koscis and Szepesvári implemented a modified version of the UCB algorithm, which incorporated a discount factor on the reward functions. This discount factor allows more recent reward to be weighted more than rewards that happened in the past. They called their algorithm Discounted UCB. (Kocsis & Szepesvári, 2006)

Garivier and Moulines also implemented their own algorithm for non-stationary reward functions. They specifically explored the case where the reward function abruptly changes at some time unknown time epoch called a breakpoint (Garivier & Moulines, 2008) They called their algorithm Sliding Window UCB (SW-UCB) and proved that it is near optimal.

Although the issue of non-stationary reward functions has been addressed, the issue seems to be less popular. The additional algorithm requirements (anytime, optimal, infinite-arms) were not addressed in the two papers mentioned above. For these reasons the algorithm chosen to use for ASSISTments does not consider the issue of stationary. The problem is of lower importance, because it is only a problem when the reward function changes. Possible scenarios where the reward function can change are when there is a new school year or when the context switches from one classroom to another classroom. A higher priority was put on good multi-armed bandit algorithm working with other constraints (anytime, optimal, infinite-arms), before worrying about the issue of non-stationary. Therefore UCB-AIR was chosen as a multi-armed bandit policy to extend.

8.3.4.1. Weighted-UCB-AIR

One of the policies implemented in PeerASSIST is called Weighted-UCB-AIR. This is an extension of UCB-AIR into the multi-objective context. There has been prior work done related to Multi-objective multi-armed bandit policies. In multi-objective optimization problems, there is more than one objective that needs to be optimized. Sometimes these objectives may conflict. For example when purchasing a product there are two objectives, cost and quality. These two objectives will often conflict, where higher quality products cost more and lower quality products cost less. In our context, we currently use three objective functions (student rating, teacher rating, and next question correctness), which may not always be correlated. The goal is to optimize all objectives at the same time. In some scenarios, there may exist a dominance relation, where all objectives are superior for one solution. For example if there exists a product that is both the highest quality and the cheapest, that product will dominate the other solutions. In our context, there may exist a peer hint which has the highest student rating, teacher rating, and next question performance. In this case, that peer hint would be considered the optimal hint. Typically it is rare that one solution will dominate all other solutions. A more common scenario is where a solution may have one objective that is better than another solution. The set of all non-dominated solutions is called the pareto set. Figure 8.3.4.1.1, taken from (Narzisi, 2008), shows an example scatter plot with two objectives. The y-axis shows values for function 1 (f_1) and the x-axis shows values for function 2 (f_1). It is assumed that lower values are better. In this example, there are three types of points shown on the graph. The gray points represent dominated solutions, where there exist better solutions in both objective functions. The blue points represent non-dominated solutions, where there does not exist a better solutions in all objectives, however there may exist better solutions in one of the objectives. This set of non-dominated solutions is called the pareto set and makes up the solution space for the multi-objective optimization problem. Now the problem is how to choose a single solution from the pareto set to represent the single optimal solution.

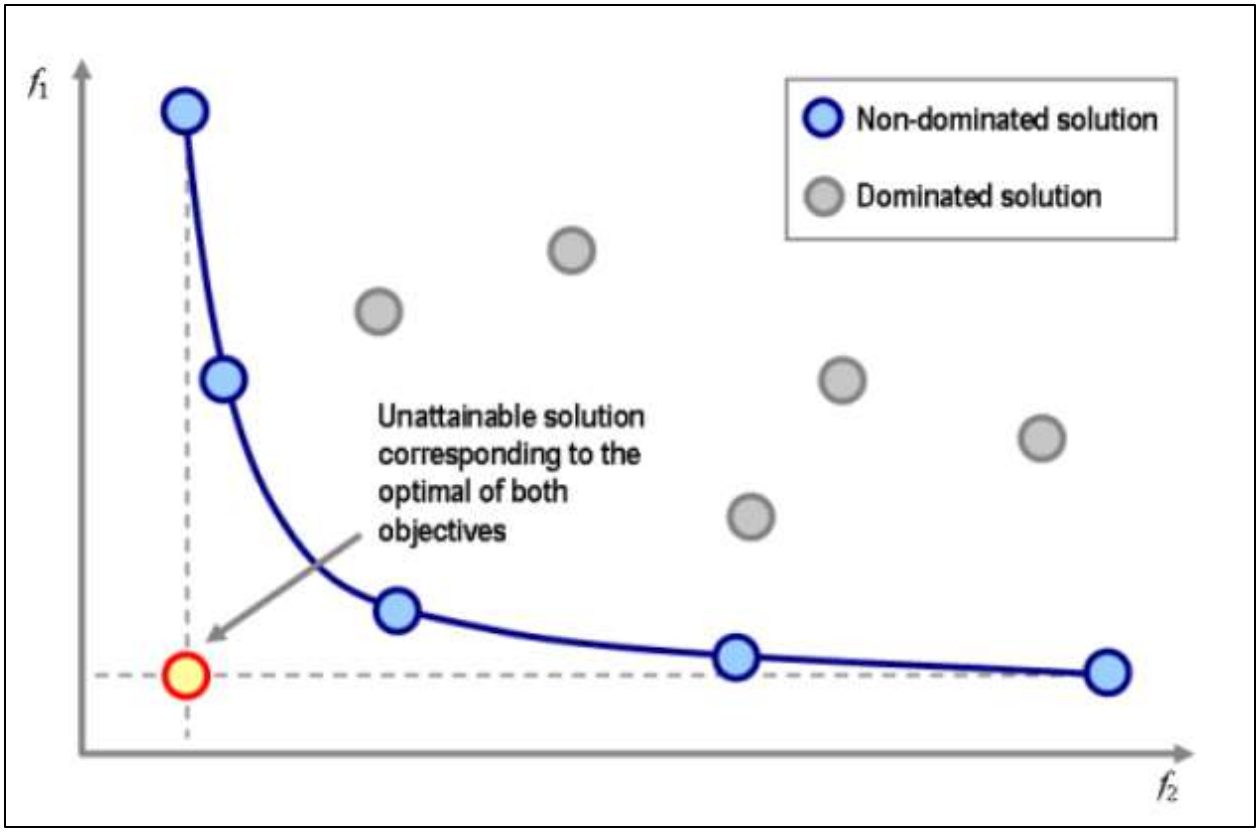


Figure 8.3.4.1.1. Example of the pareto set for a two-objective optimization problem. (Narzisi, 2008)

There are two main techniques for choosing a single optimal solution from the pareto set (Yahyaa et al., 2014). One solution is scalarize the reward vector using some scalarization function and then choose the best single scalarized reward (Eichfelder, 2009). The other solution is to minimize a pareto regret function (Drugan & Nowe, 2013; Drugan et al., 2014). For both approaches, several scalarization functions have been explored. Brys et al. explored various standard scalarization techniques in the application of when to engage a wet clutch (Brys et al., 2013). A wet clutch is a clutch that is typically used by off-road vehicles, where the plates are put in oil to prolong their use. The scalarization techniques that were compared are linear scalarization, Chebyshev scalarization, and time discount scalarization. The result of comparing the three different techniques showed the linear scalarization performed better than Chebyshev and time discount scalarization (Brys et al., 2013). It was able to identify better optimal points and a larger spread of them when compared to the other two methods.

Linear scalarization is where a weighted sum of the rewards for the objective functions is calculated. This method is simple, however has a few disadvantages. One disadvantage is that a linear combination of weights cannot always find optimal solutions in certain cases (Das & Dennis, 1997). Another disadvantage is that this technique does not capture the distribution of rewards (Das & Dennis, 1997). . This can be an issue in a scenario where there is a drastic difference in reward functions.

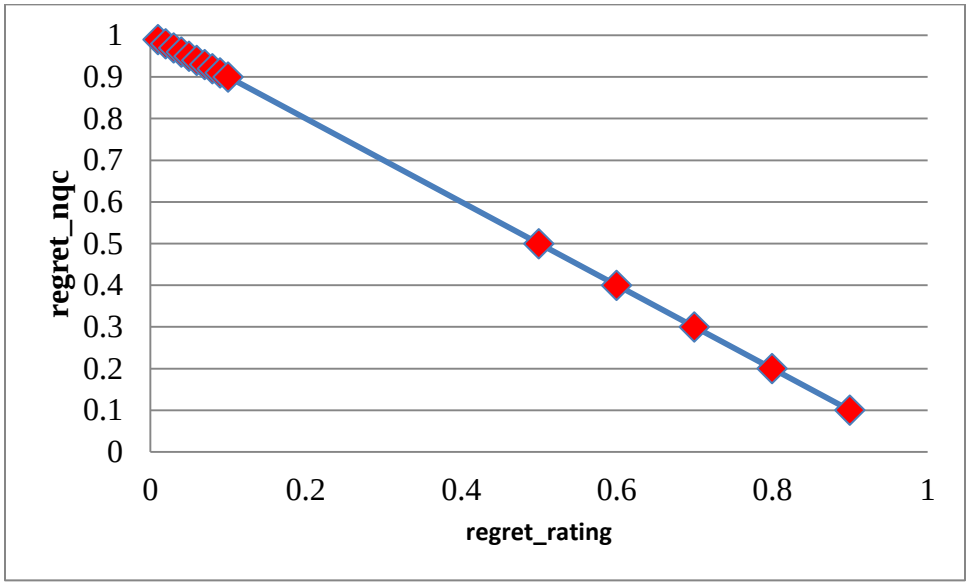


Figure 8.3.4.1.2. An example of a disadvantageous scenario for linear scalarization

To illustrate this example consider a group of peer hints with rating and next question correctness as reward functions. Figure 8.3.4.1.2 shows a hypothetical scatter plot of the distribution of peer hints. The x-axis shows the regret of the rating and the y-axis shows the regret of the next question correctness. Each data point represents a peer hint defined in terms of its regret (rating and next question correctness). In this example there are a large number of pareto optimal points with a low rating and high next question correctness in the upper right corner of the graph. This group of points is isolated from the other points in the graph. When a weighting function that weights both reward functions equally is applied to the points in this graph, the point at (0.5, 0.5) will minimize the regret function. In fact any function that weights rating between ~0.2 – 0.55, will result in the same optimal point. This is where the problem lies, where the distribution of possible weights does not correspond to the distribution of points. This

may result in incorrectly choosing the subjective preferences. Function-transformation has been suggested as a solution to help with initially choosing the subjective preferences (Marler & Arora, 2010).

Our choice for a multi-objective multi-armed bandit policy considered the following factors. Firstly, our problem is very unique since the reward functions are delayed and asynchronous. Our actions are "infinite", where the number of actions are not fixed at the start and can continue to increase as time continues. The uniqueness of our problem makes it hard to find an existing policy. Although these aspects have been studied separately, to my knowledge no research exists in the creation of a multi-objective multi-armed bandit policy that addresses all these aspects at once. For this reason, it was decided to extend the closely related single-objective multi-armed bandit policy UCB-AIR, which does address most of the previously mentioned issues.

In order to use UCB-AIR, it must be extended into the multi-objective context. Therefore a scalarization function is required. There are two big advantages of using a scalarization function (as opposed to a pareto regret function). One advantage is that the technique is simple, and simplicity is extremely important in our current development environment. Projects often have multiple students involved and students often come and go. Documentation is relatively non-existent. Simplicity is required for a project to live on.

A second big advantage is that a scalarization function can be on top of the traditional single-objective multi-armed bandit policies. These single-objective policies are well known and simple to implement. Having a generic scalarization function can provide a means to extend not just one, but many single-objective policies into the multi-objective context. This allows for our system to easily support new policies. A simple weighting of reward functions is not preferred due to the asynchronous nature of the rewards. For example, the teacher rating may have much fewer reward values than next question correctness. A ratio between the desired weighting and actual weighting is considered. A desired weighting is a subjective weight for the reward function. For example it is desired that the teacher rating is given 95% of the weight. An actual weighting is the weight based on the data, for the reward function. For example, if there is only one teacher vote and nine student votes, then the teacher vote will count only for 10% of the actual weighting. The weighting function is mathematically defined below.

Let $d \in D$ be the number of objective functions.

Let $|r_d|$ be the count of reward values for objective *d*.

Let $f(\pi_d)$ be the resulting value of running a single objective bandit policy for objective *d*.

Let w_d be the desired percent of weight to allocate to objective *d*.

Let $i \in A$ be the set of possible actions to choose from.

Let $i^* = argmax_{i \in A}$ be the best possible action.

A scalarization function for a given reward vector is defined as follows.

$$r = \sum_{n=1}^{|D|} r_{weighted_n}$$

$$r_{weighted_d} = w_{normalized_d} * f(\pi_d)$$

$$w_{normalized_d} = \frac{w_{combined_d}}{\sum_{n=1}^{|w_{combined_d}|} w_{combined_n}}$$

$$w_{combined_d} = w_{actual_d} * w_d$$

$$w_{actual_d} = \frac{|r|_d}{\sum_{n}^{|D|} |r|_n}$$

Choose i*

Intuitively the weighting function works as follows. For each reward function there exists a desired weighting for the function, where the sum of desired weights is equal to 1. There also exists the actual number of rewards samples for each reward function. The actual weight is just the percent of reward counts for a given reward function, where the sum of the actual weights will also equal 1. These two weights are multiplied together for each reward function in order to combine the weights. The result is a new set of weights, which does not sum to one. These new set of weights are then normalized so that they sum to one. The result is a weight value for each reward function, which represents a combination of the desired and actual weighting. The average reward value for each reward function is then multiplied by the weight the same way a basic weighted sum method would work. The final single reward value is the sum of all the reward values for each function from the previous step.

Table 8.3.4.1.1 shows example data rows for applying this weighting method with two objective functions. Since this method is applied on top of existing bandit policies, the average reward used as input to this function is the output value of the bandit policy being run in the single-objective context. A common problem with scalarization method is the impact of the users' desired weighting may not align with their true preferences. A common solution is to provide data or visuals to help assist with their decision. A large number of table entries are provided to represent a small piece of the potential multi-dimensional space of parameter values.

Count Reward 1	Count Reward 2	Average Reward 1	Average Reward 2	Desired Weight Reward 1	Desired Weight Reward 2	Scalarized Reward
1	1	0.25	0.75	0.25	0.75	0.63
1	1	0.25	0.75	0.50	0.50	0.50
1	1	0.25	0.75	0.75	0.25	0.38
1	1	0.50	0.50	0.25	0.75	0.50
1	1	0.50	0.50	0.50	0.50	0.50
1	1	0.50	0.50	0.75	0.25	0.50
1	1	0.75	0.25	0.25	0.75	0.38
1	1	0.75	0.25	0.50	0.50	0.50
1	1	0.75	0.25	0.75	0.25	0.63
1	5	0.25	0.75	0.25	0.75	0.72

1	5	0.25	0.75	0.50	0.50	0.67
1	5	0.25	0.75	0.75	0.25	0.56
1	5	0.50	0.50	0.25	0.75	0.50
1	5	0.50	0.50	0.50	0.50	0.50
1	5	0.50	0.50	0.75	0.25	0.50
1	5	0.75	0.25	0.25	0.75	0.28
1	5	0.75	0.25	0.50	0.50	0.33
1	5	0.75	0.25	0.75	0.25	0.44
1	10	0.25	0.75	0.25	0.75	0.73
1	10	0.25	0.75	0.50	0.50	0.70
1	10	0.25	0.75	0.75	0.25	0.63
1	10	0.50	0.50	0.25	0.75	0.50
1	10	0.50	0.50	0.50	0.50	0.50
1	10	0.50	0.50	0.75	0.25	0.50
1	10	0.75	0.25	0.25	0.75	0.27
1	10	0.75	0.25	0.50	0.50	0.30
1	10	0.75	0.25	0.75	0.25	0.37
5	1	0.25	0.75	0.25	0.75	0.44
5	1	0.25	0.75	0.50	0.50	0.33
5	1	0.25	0.75	0.75	0.25	0.28
5	1	0.50	0.50	0.25	0.75	0.50
5	1	0.50	0.50	0.50	0.50	0.50
5	1	0.50	0.50	0.75	0.25	0.50
5	1	0.75	0.25	0.25	0.75	0.56
5	1	0.75	0.25	0.50	0.50	0.67
5	1	0.75	0.25	0.75	0.25	0.72
5	5	0.25	0.75	0.25	0.75	0.63
5	5	0.25	0.75	0.50	0.50	0.50
5	5	0.25	0.75	0.75	0.25	0.38
5	5	0.50	0.50	0.25	0.75	0.50

5	5	0.50	0.50	0.50	0.50	0.50
5	5	0.50	0.50	0.75	0.25	0.50
5	5	0.75	0.25	0.25	0.75	0.38
5	5	0.75	0.25	0.50	0.50	0.50
5	5	0.75	0.25	0.75	0.25	0.63
5	10	0.25	0.75	0.25	0.75	0.68
5	10	0.25	0.75	0.50	0.50	0.58
5	10	0.25	0.75	0.75	0.25	0.45
5	10	0.50	0.50	0.25	0.75	0.50
5	10	0.50	0.50	0.50	0.50	0.50
5	10	0.50	0.50	0.75	0.25	0.50
5	10	0.75	0.25	0.25	0.75	0.32
5	10	0.75	0.25	0.50	0.50	0.42
5	10	0.75	0.25	0.75	0.25	0.55
10	1	0.25	0.75	0.25	0.75	0.37
10	1	0.25	0.75	0.50	0.50	0.30
10	1	0.25	0.75	0.75	0.25	0.27
10	1	0.50	0.50	0.25	0.75	0.50
10	1	0.50	0.50	0.50	0.50	0.50
10	1	0.50	0.50	0.75	0.25	0.50
10	1	0.75	0.25	0.25	0.75	0.63
10	1	0.75	0.25	0.50	0.50	0.70
10	1	0.75	0.25	0.75	0.25	0.73
10	5	0.25	0.75	0.25	0.75	0.55
10	5	0.25	0.75	0.50	0.50	0.42
10	5	0.25	0.75	0.75	0.25	0.32
10	5	0.50	0.50	0.25	0.75	0.50
10	5	0.50	0.50	0.50	0.50	0.50
10	5	0.50	0.50	0.75	0.25	0.50
10	5	0.75	0.25	0.25	0.75	0.45

10	5	0.75	0.25	0.50	0.50	0.58
10	5	0.75	0.25	0.75	0.25	0.68
10	10	0.25	0.75	0.25	0.75	0.63
10	10	0.25	0.75	0.50	0.50	0.50
10	10	0.25	0.75	0.75	0.25	0.38
10	10	0.50	0.50	0.25	0.75	0.50
10	10	0.50	0.50	0.50	0.50	0.50
10	10	0.50	0.50	0.75	0.25	0.50
10	10	0.75	0.25	0.25	0.75	0.38
10	10	0.75	0.25	0.50	0.50	0.50
10	10	0.75	0.25	0.75	0.25	0.63

Table 8.3.4.1.1. This table shows example data from applying the weighting function. Intermediate calculations are not shown due to spatial limitations. The final column represents the final scalarized reward as a result of applying the weighting method on the input values.

8.3.4.2. Thompson Sampling

Multi-objective Thompson Sampling is also implemented for PeerASSIST. This policy is fairly straight forward. It implements the multi-objective Thompson sampling described in (Yahyaa & Manderick, 2015). They describe the standard single-objective Thompson sampling and apply that for each reward function. A linear scalarization function is then used to scalarize the reward values. The action with the highest scalarized reward value is chosen.

Standard Thompson sampling works by modeling each reward distribution for a given action as a beta distribution (Beta(α, β)). Action that have not been sampled start with a uniform distribution (Beta(1, 1)). When an action is sampled and a reward is given, that reward is added to the distribution. For example if an action is sampled for the first time and the reward is 0.75, then 0.75 is added to the alpha parameter and 0.25 (1 - 0.75) is added to the beta parameter. The result is a new distribution of Beta(1.75, 1.25). As actions are sampled, the distribution becomes more like a normal distribution. If there are more successes than failures, the distribution will be skew right. If there are more failures than successes, the distribution will be skew right. More samples will result in a higher concentrated probability density at the estimated mean of the

distribution (high peak with short tails). To determine which action should be sampled next, a random sample is drawn from the distribution.

We modified this algorithm and substituted our weighting function in for the linear scalarization function. This is a nice proof of concept of the multi-armed bandit infrastructure and weighting function previously described. It shows that different bandit policies can be used in the framework and that those policies can easily be extended to the multi-objective context. One final tweak we made was changing the tuning constant to exploit more than explore. This is beneficial in our context of the policy, where there often will not be many samples to explore with in the current domain. The values of alpha and beta are multiplied by the tuning constant before updating the beta distribution after each sample. A higher multiplier will increase the exploitation rate. This is the same effect as "faking" more samples, where alpha and beta are increased at a faster rate.

8.3.4.3. "DoNothing"

The "DoNothing" policy does as the name suggested (nothing!). Students who are mapped to this policy will not receive any peer hints. This policy is implemented as a control for experimenting with other policies. In an experiment setting, the "DoNothing" policy will be active, as well as at least one other policy to compare to. When a student starts an assignment, they are mapped to one of the two policies randomly and probabilistically based on the experimenter's policy weighting. When students would normally see peer hints, they will see nothing if they are mapped to the "DoNothing" policy, however the information is recorded. This represents the event where a student would have received a peer hint if they were in the experiment condition of the experiment. This allows data from the policies to be compared to determine how much better one policy is to a control. The policy instance id is logged in the database, as well as the id of the work the student saw (or would have seen). From this information it can be determined which policy a student was mapped to and what peer hint the student saw (if any).

8.3.5. PeerASSIST Database ER Diagram

Figure 8.3.5.1 and Figure 8.3.5.2 show the ER diagram for the tables that make up the PeerASSTST database. Figure 8.3.5.1 shows tables and relationships for tables that are responsible for storing data on the multi-armed bandit policies and the mapping of those policies to students for experimentation. Figure 8.3.5.2 shows tables that were designed as a form of caching. The data related to dependent measures and student features is typically retrieved from the ASSISTment database. Running these queries on a database that is located on another server can be inefficient and not scalable for real-time performance. It is also typically unnecessary to repeatedly query features, such as student features or features of the student work because these features are unlikely to change often or at all. The gender of the student will likely not change every five minutes and currently the student work is immutable. A mechanism for querying the ASSISTment database and storing the information in the PeerASSIST tables was created so that the information from the ASSISTment database only has to be queried on a periodic basis. The information is stored in the PeerASSIST database and will be queried from that database until the content has expired. Reasonable expiration intervals have been determined for each table. If the content has expired, it will be queried from the ASSISTment database and copied over to the PeerASSIST database. Currently no multi-armed bandit policies are using features, however the database and code structure supports it for future use. A list of tables and column descriptions are provided below. Descriptions for the columns, (“id”, “created_at”, “updated_at”, and “deleted”) are omitted because those columns have been previously described.

Figure 8.3.5.1. ER Diagram for the PeerASSIST database on tables related to the experiment infrastructure.

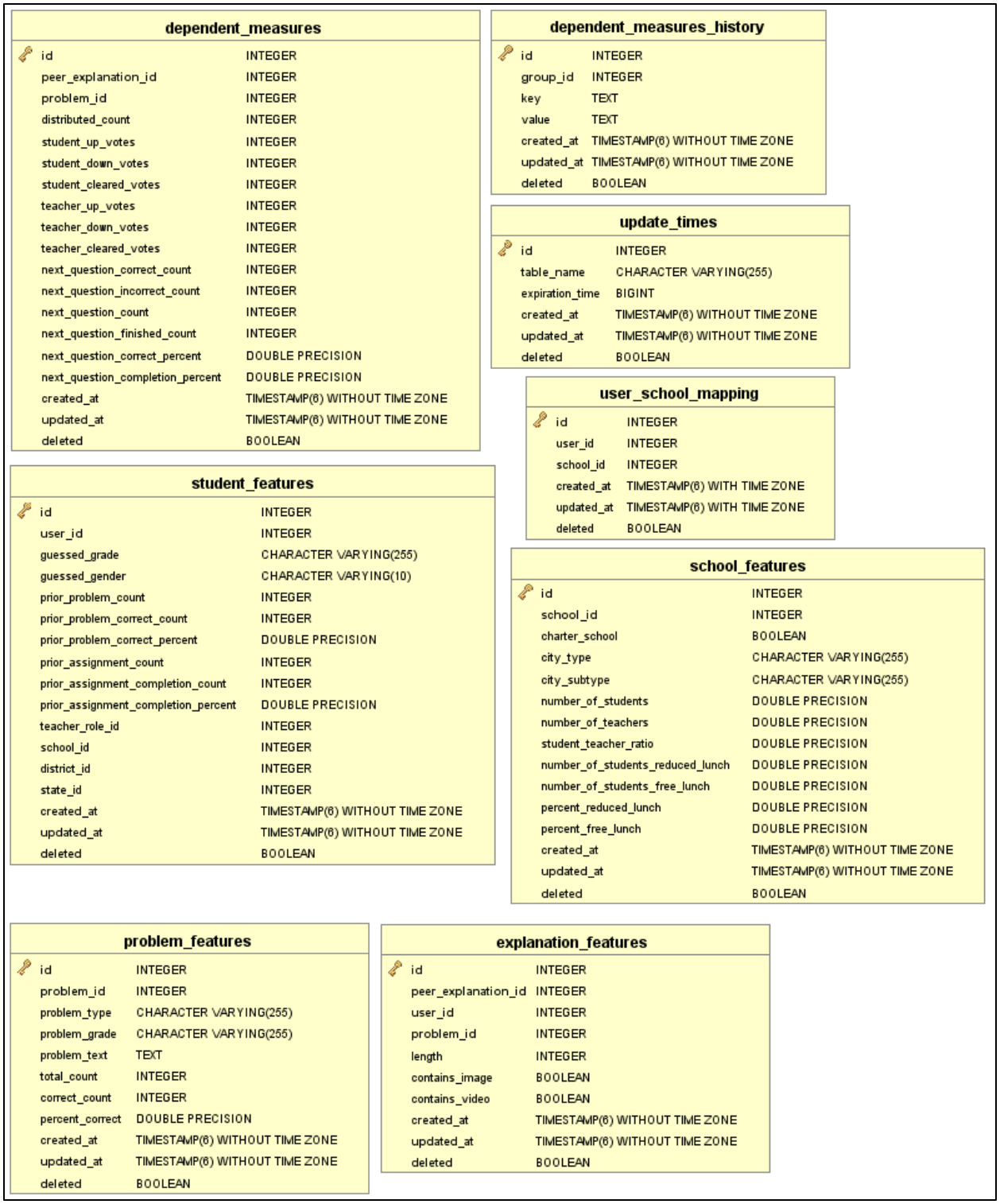


Figure 8.3.5.2. ER Diagram for the PeerASSIST database on tables related to caching features.

peer_work_distribution_policies

The peer_work_distribution_policies table stores the policies that can be used for distributing student work to other students. It is expected that associated programming is done to implement the policy listed in the database. Some policies that have been already implemented are "Weighted-UCB-AIR", "Thompson Sampling", and "Do Nothing". These policies are described in more detail in section 8.3.4.

Columns

1. name (text) – The name of the multi-armed bandit policy.

peer_work_policy_instances

The peer_work_policy_instances table stores information related to a specific instantiation of a policy. Since the same policy can be used more than once with different sets of parameters, a table to store data related to those specific instantiations is required.

Columns

1. policy_id (integer) – The unique id of the policy.
2. policy_data_properties_id (integer) – The unique id of the data properties. See the peer_work_policy_data_properties table for details.
3. policy_scope_id (integer) – The unique id of the policy scope. See the peer_work_policy_scopes table for details.
4. active (boolean) – A column to represent whether the given policy instance is currently active for potential use.
5. start_time (timestamp) – This column is for storing when a given policy instance was started. This is helpful to know what times a policy/experiment ran for.
6. end_time (timestamp) – This column is for storing when a given policy instance was ended. This can be used in combination with the "start_time" column to know the time intervals that a specific policy instance was running for.

peer_work_policy_data_properties

The peer_work_policy_data_properties table is used to store what data a given policy instance has access to when retiring data for use by the policy. There are currently three different data properties available, "instance", "policy", and "all". The "instance" property means that the

given policy instance can only use dependent measure data that was generated from student work distributed by the same policy instance. The "policy" property means that the given policy instance can only use dependent measure data that was generated from student work distributed by the same policy. The policy instances can be different as long as all instances are referring to the same distribution policy. The "all" property means that the given policy instance can use all dependent measure data generated from student work distributed by any policy instance.
The reason to have different data properties is because some experiments may or may not want to "start over". For example, a given policy instance may have been running for a year. There might be a large amount of data generated on the student work distributed by that policy instance. If a researcher proposes another policy, they may want to either start over without using any existing prior data or they may want to start with data from where the first policy left off. For these reasons, different policy data properties are available to support more flexible experimentation options.
Columns

1. name (text) – The name of the data property.
2. description (text) – A short description of the property.

peer_work_policy_scopes

The peer_work_policy_scopes table us used to store how long a given policy instance will be mapped to a specific student. This mapping can also be considered a condition mapping, since the experiments that are innately supported by this system are all involved with which multi-armed bandit policy a student is assigned. There are three different scopes that are available. "assignments_scope", "class_scope", and "user_scope". Policy instances with assignment scope will remain mapped to a specific user for the duration of an assignment. A given user can be mapped to multiple different policy instances for different assignments. Policy instances with class scope will remain mapped to a specific user for all assignments that are assigned for a specific class. If a user belongs to two or more classes, it is possible for that user to be mapped to different policy instances with class scope with a different mapping for each class. Policy instances with user scope will remain mapped to the specific user forever until the policy instance is deactivated. Deactivation of a policy instance will take effect on start of a new assignment. Therefore a user will still remain mapped to a policy instance for the duration of all

currently active assignments even if a policy instance is deactivated. If a user is mapped to a policy instance with user scope, then that user cannot be mapped to any policy instances with lower scope. If a user is mapped to a policy instance with assignment scope, it is still possible for that user to get mapped to a policy instance with higher scope on another assignment. If there are multiple simultaneous mappings, the mapping with the lowest scope takes precedence. For example, let's consider a scenario where a user starts an assignment (assignment A) and is mapped to a policy instance with assignment scope (policy instance 1). The same user then starts another assignment (assignment B). Since the scope of policy 1 is only an assignment scope, the mapping for this specific user to assignment A has no impact on assignment B. Therefore there is currently no policy instance mapping for the user for assignment B. Assignment B can then be mapped to a policy instance with any scope. In this example, Assignment B is mapped to a policy instance (policy instance 2) with user scope. When the user continues to work on assignment A, the user will be mapped to the policy instance with the scope of the lowest precedence for that assignment, which is the assignment scope of policy 1. A similar scenario exists for assignment B, where the user will be mapped to policy instance 2. When the user starts a third assignment (assignment C), the user will remain mapped to policy instance 2, because that policy instance has user scope.

This scoping structure allows for simultaneous experiments to be run with different scopes without overlapping. Once a user is mapped to a policy, they will remain mapped for the scope of that policy. Currently a problem scope can be supported, but does not exist. The reason is because no possible experimental design could be thought of, where the policy instance is allowed to change for every problem.

Columns

1. name (text) – The name of the scope.
2. scope_order (integer) – The precedence order of the scope. It is very important to interpret this column correctly. The lowest scope has the highest precedence. assignment scope has a scope order value of three, class scope has a scope order value of two, and user scope has a scope order value of one.

peer_work_user_experiment_mapping

The peer_work_user_experiment_mapping table stores all mappings from users to policy instances (experiments). This is the most important table in the PeerASSIST database and without this table the entire experimentation framework collapses. Currently this is the only table in the PeerASSIST database which is storing data that is not saved anywhere else. This mapping can be inferred from the distributed_explanations table for users who experience the condition of the experiment (receive peer work). It cannot be inferred for users who do not experience the conditions of the experiment and who do not have an entry in the distributed_explanations table.

Columns

1. user_id (integer) – The id of the user.
2. class_id (integer) – The id of the class that the user is in of which the assignment was assigned for.
3. assignment_id (integer) – The id of the assignment.
4. policy_instance_id (integer) – The id of the policy instance id that the user is mapped to.

peer_work_experiment_groups

The peer_work_experiment_groups table is for storing data that is specific to a policy instance as the policy instance runs. Currently this table is not used, although including it in the infrastructure may be important for future uses. This table work in conjunction with the peer_work_experient_group_data table, where the actual data is stored. This table is merely holding the grouping of the data.

peer_work_experiment_mapping_to_group_associations

The peer_work_experiment_mapping_to_group_associations table stores the mapping from a user-experiment entry to a group of data specific for that experiment/policy instance.

Columns

1. user_experiment_mapping_id (integer) – The id of the mapping entry from user to experiment.
2. experiment_group_id (integer) – The id of the data group for a given policy instance.
3. problem_id (integer) – The id of the problem.

4. assignment_id (integer) – The id of the assignment.
5. policy_instance_id (integer) – The id of the policy instance.

peer_work_experiment_group_data

This is a table to support data in a generic format that is specific to a given policy instance. In the future various policy instances may need to store data as they function. This table will support that stored of data in a key-value format. All parameters will be stored as keys with the corresponding values stored as well. The specific implementation of the policy instances will be responsible for managing its own keys and values.

Columns

1. experiment_group_id (integer) – The id of the experiment group entry, which holds the grouping of data.
2. key (text) – The key of the parameter or variable being stored.
3. value (text) – The value of the parameter or variable.

history_choice_data

The history_choice_data table is not currently used. The original idea was to store all information in order to reproduce the multi-armed bandit policies. The data is recoverable elsewhere, although not easily. It is not possible to create such a table structure for the following reason. For every action choice (student work), there are a list of actions to choose from. These actions are generated by the students, where it is possible for every student to generate a potential action choice. Consider an assignment consisting of 10 problems and a class consisting of 100 students. This means that a student can generate work for each problem of the assignment, which leads to a potential 1,000 (100 x 10) student works that can be distributed. Since there are 1,000 works that can be distributed and 100 students that could potentially receive the work, there is an upper bound of 100,000 work distribution data rows to store for an assignment. This is the number of possible rows that would need to be stored in the database for a single assignment. This number is too large to feasibly store all the data, therefore the data is not directly stored, but can be queries in more complex ways from the existing information that is stored. Since this table is not used, the columns of this table are not described, because the

implementation is incomplete. The table is left as a place holder in the event that similar data storage needs that are possible to store arise.

dependent_measures

The dependent_measures table stores all the information currently being used as dependent measures for evaluating the quality of student work. These dependent measures include the student rating, teacher rating, and next question performance. All primitive data is included as well as the calculated derived data for these dependent measures. This table was designed as a form of caching for the dependent measures, however it is not currently used. Due to the real-time nature of the system, the dependent measures are frequently updated and the caching is expected to become outdated too quickly. Therefore the dependent measures are still queried from the normal production database on demand. This table and associated code still exists for potential future usage.

Columns

1. peer_explanation_id (integer) – The unique id of the student work.
2. problem_id (integer) – The unique id of the problem that the work was generated for.
3. distributed_count (integer) – The number of times the given student work has been given out and seen by other students.
4. student_up_votes (integer) – The number of times a student has given the given work a positive rating. Student vote counts are primitive data used in calculating the student rating for the work they received.
5. student_down_votes (integer) – The number of times a student has given the given work a negative rating.
6. student_cleared_votes (integer) – The number of times a student has cleared their rating of the given work.
7. teacher_up_votes (integer) - The number of times a teacher has given the given work a positive rating. Teacher vote counts are primitive data used in calculating the student rating for the work they received.
8. teacher_down_votes (integer) - The number of times a teacher has given the given work a negative rating.

9. teacher_cleared_votes (integer) - The number of times a teacher has cleared their rating of the given work.
10. next_question_correct_count (integer) – The number of times a student has answered the next question correctly after receiving the given student work.
11. next_question_incorrect_count (integer) - The number of times a student has answered the next question incorrectly after receiving the given student work.
12. next_question_count (integer) - The number of times a student has started the next question after receiving the given student work.
13. next_question_finished_count (integer) - The number of times a student has finished the next question correctly after receiving the given student work.
14. next_question_correct_percent (double) – The percent of finished questions answered correctly by students who have been given the current student work.
15. next_question_completion_percent (double) - The percent of questions that were completed by students who have been given the current student work.

student_features

The student_features table stores all the information related to features of the student. The information in this table is populated by queries run on the ASSISTments production database. This table was designed as a form of caching for student features which only need to be updated on a periodic basis. Features such as inferred gender are unlikely to change, and features such as prior performance do not need to be updated in real-time.

Columns

1. user_id (integer) – The unique user id of the student.
2. guessed_grade (text) – The grade of the student inferred from the grade of the most recent class (set by the teacher). Possible value = {K, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, Freshmen, Sophomore, Junior, Senior, Masters, PhD, N/A}.
3. guessed_gender (male/female/unknown) – The gender of the student inferred from their first name. A list of over 65k first names and their gender mapping is used to infer the student's gender. This list was compiled by Jörg Michael and modified for our programmatic use by Doug Selent (Michael, 2007).
4. prior_problem_count (integer) – The number of problems attempted by the student.

5. prior_problem_correct_count (integer) – The number of problems correctly answered by the student.
6. prior_problem_correct_percent (double) – The percent of problems correctly answered by the student.
7. prior_assignment_count (integer) – The number of assignments the student has started. This counts all class assignments with a start date prior to three days of the current date.
8. prior_assignment_completion_count (integer) – The number of assignments that a student has started and completed. This counts all class assignments with a start date prior to three days of the current date.
9. prior_assignment_completion_percent (double) – The percent of assignments that were completed by the student. This counts all class assignments with a start date prior to three days of the current date.
10. teacher_role_id (integer) – A unique id for the teacher of the student. If a student has multiple teachers, the teacher id of the class with the highest grade is used.
11. school_id (integer) – A unique id to represent the school that the student belongs to.
12. district_id (integer) – A unique id to represent the district that the student belongs to.
13. state_id (integer) – A unique id to represent the state that the student belongs to.

school_features

The school_features table is an extension of the student_feasture table. Since several students can belong to the same school, their school features will be the same. Rather than storing this information twice, the school feature are stored in this table and a mapping from user to school is kept to know which student belongs to which school. This table is also designed as a form of caching for school information. This information changes infrequently and therefore does not need to be updated as often.

Columns

1. school_id (integer) – The unique id of the school.
2. charter_school (boolean) - Whether the school is a charter school or not.
3. city_type (text) – The type of the city. Possible values include {Rural, Town, Suburb, City, N}.

4. city_subtype (text) – More type information extending from the city type information listed above. Possible values include {small, midsize, large, distant, remote, fringe}.
5. number_of_students - The average number of students that attend the school.
6. number_of_teachers - The average number of teachers that work at the school.
7. student_teacher_ratio - The ratio of students per teachers.
8. number_of_students_reduced_lunch - The number of students who have reduced lunch fees, indicating that these students likely come from poorer families.
9. number_of_students_free_lunch - The number of students who get free lunch, indicating that these students likely come from poorer families.
10. percent_reduced_lunch - The percent of students who have reduced lunch fees.
11. percent_free_lunch - The percent of students who get free lunch.

user_school_mapping

The user_school_mapping table stores the mapping between a student and the school that they belong to. This table is required to map a given student to their school features. It is rare that a student changes schools, therefore this information is also good to cache.

Columns

1. user_id (integer) – The unique id of the student.
2. school_id (integer) – The unique id of the school.

problem_features

The problem_features table stores features of the problem (difficulty, grade, etc…). This table is also used as a form of caching since these features do not need to be updated in real-time.

Columns

1. problem_id (integer) – The unique id of the problem.
2. problem_type (text) - The type of problem (short answer, multiple choice, etc...). Possible values include {numeric, numeric expression, fraction_reduced, choose_1, choose_n, iframe, fill_in_ignore_case, fill_in_1, open_response, racket, rank, algebra, old_algebra}
3. problem_grade (text) - The grade the problem is usually given in. This is inferred first by finding all assignments that have the problem in it. Counts of the grade for all classes

that those assignments belong to are taken. The grade is determined by the grade with the highest number of class counts. Possible values include {K, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, Freshmen, Sophomore, Junior, Senior, Masters, PhD, N/A}.

4. problem_text (text) - The text of the problem; i.e. the problem. This consists of HTML characters and can link to images and videos.
5. total_count (integer) - The number of times the problem has been answered by students
6. correct_count (integer) - The number of time the problem has been answered correctly
7. percent_correct (double) - The percent of the time the problem has been answered correctly.

explanation_features

The explanation_features table stores features of the student work. This table is also used as a form of caching since these features do not need to be updated in real-time. Currently student work is immutable, and there are no means provided for the work to be edited or modified. Therefore data on the features of the student work will never be out of date. This data is ideal for caching in the PeerASSIST database.

Columns

1. peer_explanation_id (integer) – The unique id of the student work.
2. user_id (integer) – The unique id of the user who created the work.
3. problem_id (integer) – The unique id of the problem that the work was created for.
4. length (integer) – The number of characters in the student work.
5. contains_image (boolean) – Whether or not the given student work contains an image. A simple regular expression is used to detect the HTML <img> tag to determine if the student work contains an image.
6. contains_video (boolean) - Whether or not the given student work contains a video. A simple regular expression is used to detect the HTML iframe tags with a YouTube URL to determine if the student work contains an image.

update_times

The update_times table stores the information on the cache expiration of data for the feature tables. This table stores an expiration time for each table. When a row in one of the feature tables has an updated_at timestamp, that when compared to the current time, is greater than the expiration time for the table, the data has expired. This table is used to set expiration times for all of the PeerASSIST tables acting as cache.

Columns

1. table_name (text) – The name of the table of which the expiration time applies to.
2. expiration_time (integer) – The expiration time for data in the given table. This time is represented in milliseconds. It is used by comparing the current time to the last timestamp of the data rows in the table. When the difference between the two timestamps is greater than the expiration time, the data has expired. Designing the caching infrastructure in this manner, makes it simple to extend, update, and maintain.

8.4. PeerASSIST Use and RCT

System Usage Statistics

This section explores the overall usage the system has seen to gain insight on how it is being used and what to expect as more users begin using PeerASSIST. Table 8.4.1 provides some summary statistics on the usage of the PeerASSIST system. The data on student work was collected from January 5, 2016, which is when students first had the ability to submit work inside the ASSISTments system. The data for receiving student work was collected from January 18, 2017, when PeerASSIST was first deployed. Currently 610 teachers have enabled the option for students to show their work on problems in at least one assignment. This is roughly 20% of the total active users. Out of the 610 teachers, 60 (~10%) are using PeerASSIST. This number is expected to increase as more teachers learn about PeerASSIST.

Student Work Collected	338,017
Sharable Work	310,856
Work Distributed	2,187
Flagged Work	263
Flagged by Student	12
Unique Problems with Submitted Work	28,445
Unique Students who Generated Work	18,829
Unique Students who Received Work	559
Unique Teachers who Require Work	610
Unique Teachers Using PeerASSIST	60

Table 8.4.1. A summary of usage statistics for work submission and PeerASSIST is presented in this table.

One insight gained is the influence of the default option to have student work sharable. This option was selected by default, meaning that all student work was sharable unless the student unchecked the checkbox. As a result nearly 92% of the work submitted was sharable. One issue that arose was that a large number of student work was poor. After speaking with some teachers, it was decided to switch the sharable option from an opt-out policy to an opt-in policy, as an attempt to reduce the amount of poor quality work. One teacher that I contacted, who was using PeerASSIST had recently disabled it because his students were submitting "nonsense". This change was recently deployed within the last two weeks. Out of the last 10,000 student work instances, only 315 of them were marked as sharable. Changing the default option for whether student work is sharable from an opt-out policy to an opt-in policy changed the percent of work that was considered sharable from ~92% to ~3%. This drastic change suggests that most students do not check or uncheck the check box regardless of whether they think their work is good enough to share with others. This information on the impact that default choices have will be considered when implementing future components of the PeerASSIST system.

One positive statistic to report on is the amount of work that was flagged as inappropriate. Since PeerASSIST is being used mostly with students who are in grades 6-8, content control was extremely important. Out of all the student work collected, there were only 263 instances that were flagged which less than 0.1%. In addition, all but 12 of these instances were automatically flagged by the system. Only 12 instance of student work were given out to students who flagged

them as inappropriate by students. Presenting these statistics to teachers will help alleviate fears of using PeerASSIST related to content control issues.

Another aspect explored is how many unique problems had student work distributed to another student and how many instances of student work were distributed to students for a given problem. These questions are related to the functionality of the multi-armed bandit policy. It is useful to know if the policies have enough samples of student work distributed in order to converge on the good student work. Figure 8.4.1 shows the count of student work distributed per problem on the x-axis and how many problems where that number of work was distributed on the y-axis. In order to determine which work is good, a larger number of samples of the student work must be given to other students. Unfortunately only 1-2 instances of student work are distributed for most problems. This is because for a given class and a given problem, only a few students are likely to need and receive student work. Although the multi-armed bandit policy is tuned to exploit, 1-2 samples is not enough to converge on an optimal instance of student work. The major benefit of using multi-armed bandit policies will be gained when PeerASSIST expands outside of a single classroom.

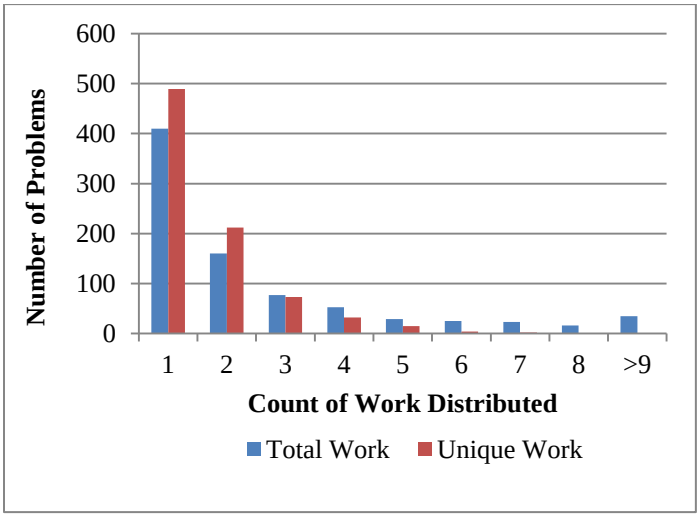


Figure 8.4.1. This figure shows the number of problems with various counts of student work that were distributed. A Column can be interpreted as "How many problems had exactly one student work distributed to another student?"

Students who receive work done by other students have the option to rate the work they saw. Students can only rate a given instance of work one time but may change their rating at any time. Figure 8.4.2 shows the percent of instances of distributed work that were voted on by students

and also shows the overall ratings of the student work. To make the distinction between the two pie charts clearer, the pie chart on the right shows the percent of work for different rating values where a rating is the sum of the positive (+1) and negative (-1) votes. The percent of these votes is shown in the pie chart on the left. Nearly 90% of the instances of distributed work were not voted on. About 8% of the distributed work was voted as negative and 4% was voted as positive. This suggests that most students will not rate the work they see and also suggests that the bad work outnumbers the good work 2:1 according to the student rating.

The majority (58%) of the work distributed received a rating of -1 and 6% received a rating of -2. These numbers were expected, since the majority of student work was expected to be poor. The main goal is to exploit the work that has a positive rating which consists of about 35% of the total work. These percentages are pessimistic in the sense that they come from the exploration phase of the bandit policy. Initially it is unknown which work is good or bad. As the number of work distributed increases, the higher rated work will be distributed much more than the lower rated work. These percentages will then favor more positive ratings and those ratings will be higher and will increase well beyond the maximum rating shown (3) in Figure 7. Another factor to consider is that the first work to be submitted will also be the first work to be chosen to be redistributed based on the fact that there is no other work to choose from. The distribution policy will rarely choose lower quality work over higher quality work.

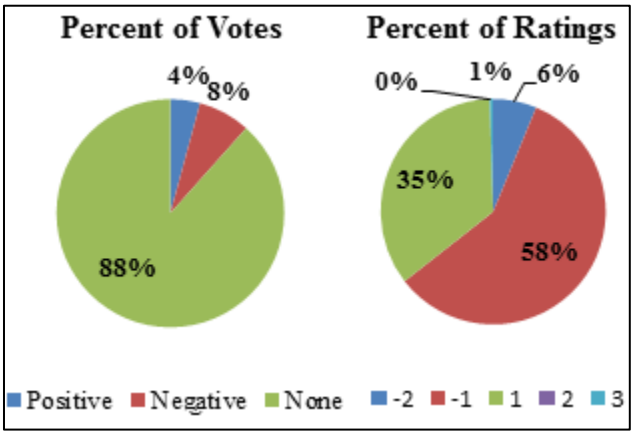


Figure 8.4.2. The pie chart on the left shows the percent of distributed work that was voted positively and negatively. The pie chart on the right shows the percent of student work ratings.

Experiment Design

A randomized controlled experiment was run, which leveraged the existing experiment infrastructure. The purpose of this experiment was to see if crowdsourced student work could be a potentially useful form of tutoring when compared to no tutoring at all. This is a common scenario for problems where there is no tutoring content. This experiment is a simple A/B design, where students were randomly assigned into the control condition (no tutoring) or the experiment condition (peer work).

Students in the control condition were simply mapped to a policy that returns no peer work. These students will never see any peer work; however the information is logged for the scenario where the students would have seen peer work if any had been given. Students in the experiment condition were mapped to a customized multi-objective multi-armed bandit policy called Weighted-UCB-AIR previously described in this chapter. This experiment ran inside the ASSISTments for any teachers who had PeerASSIST enabled. The data collected and the experiment results are discussed in the following section.

Experiment Analysis

There were a fewer number of teachers and students who participated in the A/B experiment. This is because the experiment was conducted in the early stages of the existence of PeerASSIST, therefore only a few teachers knew about the feature at that time. The data used for analysis was obtained from the students of three different teachers. Since these teachers assigned multiple assignments, students participated for all assignments. It is possible for a single student to participate in the experiment for several assignments. It is also possible that that same student was assigned into the control condition for one assignment and assigned into the experiment condition for another assignment. To avoid analysis issues, one assignment from each teacher was chosen at random to include for analysis. This means that a single student is only counted once and only belongs to a single condition.

A total of 100 students participated in the experiment on 1,500 problems. These students submitted 192 instances of student work, which were all sharable. A total of 44 (25 unique) instances of work were distributed to 31 unique students in the experiment condition. 18 out of

the 100 students answered all problems correctly and never received tutoring for the assignment. These students did not experience the condition of the experiment and were excluded from analysis. There remained 82 students, 44 in the control group and 38 in the experiment group.

Since it is possible to have an unlucky random condition assignment a $\chi 2$ test of goodness fit was performed to determine that the condition assignment was not biased, $\chi 2$ (1, N = 82) = 0.44, p = 0.51. Two different dependent measures were analyzed to compare performance between the two groups. The average next question correctness is used as one measure of effectiveness after the first problem that a student sees peer work on. This measure is also used for students in the control condition who would have seen work if they were in the experiment condition. This measure is used as it is less susceptible to noise and the differences in performance can be attributed to the effect of the intervention. Differences in assignment completion rates were also analyzed. Both statistics are reported separately.

For assignment completion, 38/44 (86%) students completed the assignment in the control group and 32/38 (84%) students completed the assignment in the experiment group. It is also possible that students in the experiment group did not see any peer work because none existed at the time it was requested. To analyze the intent to treat they were still considered as part of the experiment group, however those students who were in the experiment group and saw peer work were also included in the analysis. There were 30/31 (97%) students in the experiment group who saw peer work and completed the assignment. A $\chi 2$ test of independence was performed to determine that the completion rates were not different, $\chi 2$ (1, N = 82) = 0.076, p = 0.78. A second $\chi 2$ test of independence was performed to determine that the completion rates of the 31 students in the experiment group who actually saw peer work was not different $\chi 2$ (1, N = 75) = 2.33, p = 0.13.

Performance on the next problem after the first incorrect answer was compared between the two groups. Some students made the first mistake on the last problem and were excluded from the analysis. The control group and experiment group had 25/41 (61%) and 17/33 (52%) students answer the next question correctly respectively. This difference is not significantly different $\chi 2$ (1, N = 74) = 0.67, p = 0.41 although certainly not favorable. Although the randomized controlled experiment was unsuccessful at helping students, it did not (statistically significantly) cause any harm. Since this was the first attempted experiment run with the PeerASSIST system, it is viewed as an opportunity for improvement.

Future Work

There are several possible areas for future work for the PeerASSIST project. One area of future work is to address the issue of distributing a large amount of poor quality content. Although inappropriate content is automatically flagged, there is no mechanism to prevent poor quality (but still appropriate) content from being distributed. An attempt previously discussed was made to address this issue by changing the check box for "My work is good enough to share with others" from an opt-out policy, where the checkbox is selected by default and students can deselect it, to an opt-in policy, where the checkbox is not selected by default and students can select it. Initially 92% of the content was marked as "sharable". After the change from an opt-out policy to an opt-in policy, the percent of content that was marked as sharable is only 3%. The consequence was that when a student needed peer assistance, there was none to distribute. This essentially rendered PeerASSIST useless. It is future work to change the policy back to an opt-out policy, because students do not toggle the checkbox whether it is checked by default or not. This change will allow there to be enough student work collected in order for PeerASSIST to function again. In addition to this change, another change needs to be made to again try to address the issue of poor quality content being distributed to students. An idea originally suggested to me by Joe Beck, was to train a classifier on features of the student work submitted to detect which explanations are adequate to redistribute. I think engineering features of student work and creating a classifier to predict its quality is a great idea for future work. Adding another preliminary filter will help prevent poor quality work from being distributed to students. In this case, not being shown any work may be better than showing poor quality work.

In addition to the machine learning classifier to filter out poor quality work, the system could be improved to provide better mechanisms for students to submit higher quality work. Currently the TinyMCE box that is provided is slow to use. Students have to type their work and manually add symbols and change fonts as needed. This can be an extremely time consuming process, which is potentially why many students are generating poor quality work. A change needs to be made to our editor so that it is easier for students to produce higher quality content. One idea is to integrate Graspable Math into the work submission process (Ottmar, 2015). Graspable Math allows students to solve algebraic problems by simple moving the symbols on the screen with the mouse. After each operation, the problem is shown at its current step. This process is basically generating a step-by-step procedure for the work that the student used to solve the problem. This

is much easier than manually typing the work for each step and more user-friendly. Although Graspable Math is not applicable for all problems, it can still be useful for a large number of problems that student work on. An idea for future work is to integrate Graspable Math in a way where students can use it and the work can be saved as plain text.

Another area of future work is to continue work with the multi-objective multi-armed bandit algorithms that are being used to determine which student work are distributed. Due to the complex nature of the system, there are several aspects of these algorithms that can be examined. This includes topics related to the multi-objective aspect such as what reward functions to use, how many to use, and how to weight them. Currently two policies implemented are multi-objective Thompson sampling and multi-objective UCB-AIR. Both policies use my weighting algorithm to combine the actual weighting with the desired weighting. This is a more complicated form of linear scalarization, which is a common approach to scalarazing the reward vector in multi-objective multi-armed bandit problems. There is a second common approach which uses the Pareto set with a defined regret function. This approach has been empirically shown to perform better than scalarization methods (Yahyaa, 2015). Different scalarization techniques can be explored as well as topics related to the optimality of the algorithms under multiple asynchronous reward functions. Contextual bandits can also be considered for personalization based on a combination of student features, problem features, and features of the student work. A large amount of infrastructure already exists to retrieve several contextual features related to the student, school, problem, and student work. This data is available and ready for use. It can also be explored as to what types of explanations (in terms of their features) are most helpful to students. It would be a good area for future work to implement a policy that uses these features to see if this policy that uses contextual features can improve student learning more than a non-contextual policy.

Further experimentation using the PeerASSIST system also provides opportunities for future work. One research question that can be explored is "Does giving help to peers benefit the help-giver"? The idea behind this research question is that just believing that the student's work will be shown to help other students will cause the help-giving students to learn more just by generating the work. This question does not address the students who receive the help, it only focuses on the help-givers and if there is any benefit to them. There has been prior research that shows that giving help is beneficial to the help-giver (Webb & Palinscar, 1996; Topping et al.,

2004; Roscoe & Chi, 2007). This is different than receiving help, which has shown not to have consistent learning benefits and is a more complex process to be effective (Webb & Mastergeorge, 2003). King offers an explanation of why giving help is beneficial to the help-giver (King, 2002). She explains how answering questions (giving help) functions as a way for students to monitor their own understanding of the material and extend their knowledge beyond the material when answering questions. The extra focus on metacognition helps students improve their thinking. A simple randomized controlled experiment can be run, where one group of students (the control) solve problems in ASSISTments as they normally would. This group would be informed that the work they generate will not be shared with others. Another group (the treatment) would solve problems in ASSISTments and be required to show their work. They would be informed that their work would be shown to others who need help. The goal of this experiment would be to see if the students in the treatment group showed larger learning gains than those in the control group. Although this experiment has been done in other domains, it would be a good idea to evaluate its effectiveness inside of the ASSISTments online tutoring environment.

A final area of future work is to create a similar system called TeacherASSIST. PeerASSIST is meant to collect student work to redistribute to other students. TeacherASSIST will involve the teachers, where teachers can generate tutoring for problems, which can then be redistributed to students the same way PeerASSIST currently works. One main advantage of TeacherASSIST is there is much less concern about content control. We trust that teachers will create higher quality and appropriate content. Therefore we would be able to share content created by one teacher with other teachers and students much easier. This would improve crowdsourcing content, since it can be shared with a larger audience. Currently PeerASSIST only allows content to be distributed from students who have the same teacher. Therefore once an assignment is completed, it is unlikely those students will do the same assignment again. This means that all the work those students generated will not be used again, unless the teacher assigns the same assignment to the next class of students in the following school year. TeacherASSIST would solve this issue by allowing previously generated content to be used by everyone.

Contributions & Conclusion

The work presented here demonstrates an attempt to improve tutoring content in online tutoring systems. In chapters one and two we showed that Bayesian networks can be used represent learning maps to model student performance. We showed that these learning maps can be simplified by merging multiple skills into a single skill, without practical loss of predictive accuracy. We also showed that time series data is necessary to successfully merge skills if the values of the observable data are not (somewhat) equally distributed.

Several randomized controlled experiments were conducted using buggy messages generated from machine-learned incorrect processes inside the ASSISTments online tutoring system. Students who make the most common wrong answers often quickly fix their mistake without help and proceed through the rest of the problem set without any more mistakes. Students who make less common mistakes tend to struggle for several problems before completing the problem set. Students rarely make the same mistake twice and usually the second mistake is another common wrong answer. In one experiment, buggy messages significantly reduced hint usage, but did not improve learning when compared to the current hints inside the ASSISTments system. This implies that they are just as effective as hints, but not better than hints. In another experiment, video-messages (buggy messages in the form of 20-30 second videos) did not significantly affect student performance.

Alternative methods of statistical analysis were explored, such as Bayesian A/B hypothesis testing, as a more applicable method for analysis of experiments at large scale and in real-time. It was shown that using Bayesian hypothesis testing is a feasible alternative to the standard Frequentist null hypothesis testing, with several advantages that can be gained. We extended the Bayesian A/B testing framework to include confidence intervals on the learned prior probability that an experiment has differences between conditions. We concluded that twenty-two experiments are not enough to accurately learn a genuine prior (~200 are required), and both Bayesian and Frequentists methods gave similar results.

The PeerASSIST system was created and integrated into the ASSISTments online tutoring system to crowdsource and redistribute student work to other peers in need of assistance. This crowdsourcing system implemented several aspects of existing systems which run large-scale simultaneous experimentation (Bing, Google, and Facebook). Custom multi-objective multi-armed bandit policies were developed to balance the objectives of exploring to find higher

quality student work and also exploiting the current best known student work in the context of the ASSISTments online tutoring system (multi-objective asynchronous delayed reward functions with an infinite horizon and infinite actions). An experiment infrastructure was built into PeerASSIST to run experiments on the bandit policies as needed. This system is currently running and has been used by over 60 teachers. Over 300,000 instances of student work have been collected from over 18,000 students on 28,000 problems. There have been over 2,100 instance of student work that has been redistributed to ~560 unique students. Some useful information gained from this data collected is that the majority of student work is low quality, this work did not improve student learning performance, student work is rarely distributed more than 1-2 times, and student work is rarely inappropriate. The result of a randomized controlled experiment shows that students who use PeerASSIST do not have statistically significant performance differences than students who do not use PeerASSIST. It is expected that the content distributed by the PeerASSIST will improve over time as the system design is tweaked and due to the nature of the multi-armed bandit algorithms.

This work demonstrates the success of PeerASSIST both as a crowdsourcing system and an experimentation platform. It represents a successful system that crowdsources tutoring content from students in online educational systems and redistributing that content in real-time and at scale.

Acknowledgements

Acknowledgements are listed in alphabetical order by last name.

I would like to acknowledge Seth Adjei. Seth and I worked closely together on the first two chapters related to modeling skill graphs with Bayesian networks. We wrote two papers together, which are included in this document.

I would like to acknowledge all the contributors to the "ALI" project for their various contributions.

I would like to acknowledge Joseph Beck for his advice and suggestions related to my research methodology over the past five years.

I would like to acknowledge Vijaya Dometti for her work with me on Bayesian hypothesis testing. She was the first student who I have successfully advised as a faculty member. Our success would not have been possible without her persistence.

I would like to acknowledge Chris Donnelly for his work in getting TinyMCE working in the tutor. TinyMCE is the fancy text box that allows students to submit stylized text, math formulae, and images. This was a difficult task, which I doubt I could have figured out myself. Chris has also supported the programming environment that the PeerASSIST server uses and has helped me with system integration throughout this project.

I would like to acknowledge my advisor Neil Heffernan. Neil has helped me mostly on the business and social side by writing grants to support my work and introducing me to other researchers in the field. We have also worked together on designing the PeerASSIST system and several other research topics.

I would like to acknowledge Bohao Li for his work. Bohao helped code the program which automatically generates and evaluates Bayesian networks in the form of knowledge tracing

models. Bohao is a good programmer and a better team member. Bohao and I also wrote a paper on the predictive power of buggy messages which is included in this document.

I would like to acknowledge Anthony Bothelo. Anthony has graciously volunteered to maintain the project and keep ALI alive. He has done work related to queries on feature generation and is currently maintaining the system.

I would like to acknowledge David Magid. He has helped me with the programming design inside of the ASSISTments tutor and also supported the programming environment that the PeerASSIST server uses.

I would like to acknowledge Korinn Ostrow for her work related to the ALI project. Korinn volunteered to be in charge of quality assurance of the system and ensured its integrity and wrote the majority of the paper.

I would like to acknowledge Thanaporn ("March") Patikorn for his our work on TAME. Although I have not included our work in this document, we have worked together on two publications related to better detecting treatment effects on data from multiple randomized controlled experiments.

I would like to acknowledge Yan Wang for her help with programming the PeerASSIST system. Yan was responsible for the teacher views, where teachers can view student work. This was a huge help to me as I could not have completed the project in a timely manner without her.

I would like to acknowledge Xiaolu Xiong for his initial work on the PeerASSIST settings. Xiaolu programmed a first draft of the settings and the assign forms, which provided a starting point for me to take over.

I would like to acknowledge Siyuan Zhao for his work on the implementation and integration of Thompson sampling inside the PeerASSIST system. Siyuan has also graciously volunteered to take over the maintenance of PeerASSIST so that the project will live on.

References

1. Adjei, S. A. (2014). Refining Learning Maps with Data Fitting Techniques (Master's Thesis, Worcester Polytechnic Institute).
2. Adjei, S., Selent, D., Heffernan, N., Pardos, Z., Broaddus, A., & Kingston, N. (2014). Refining Learning Maps with Data Fitting Techniques: Searching for Better Fitting Learning Maps. In *Educational Data Mining 2014*.
3. Agrawal, R. (1995). Sample mean based index policies with O (log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 1054-1078.
4. Aleven, V., & Koedinger, K. R. (2000). Limitations of student control: Do students know when they need help?. In *Intelligent Tutoring Systems* (pp. 292-303). Springer Berlin Heidelberg.
5. Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2), 167-207.
6. Attali, Y. (yet to be published) Practicing for a Quantitative Reasoning Assessment: Effect of Question and Feedback Type. A yet to be published manuscript shared via email with Heffernan Oct 1, 2014.
7. Audibert, J. Y., Munos, R., & Szepesvári, C. (2007). Variance estimates and exploration function in multi-armed bandit. In *CERTIS Research Report 07–31*.
8. Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*(2-3), 235-256.
9. Baffes, P. T. (1994). Automatic student modeling and bug library construction using theory refinement (University of Texas at Austin).
10. Baffes, P. T., & Mooney, R. J. (1996). A novel application of theory refinement to student modeling. In *AAAI/IAAI*, Vol. 1 (pp. 403-408).
11. Baker, R. S., Corbett, A. T., & Aleven, V. (2008). More accurate student modeling through contextual estimation of slip and guess probabilities in Bayesian knowledge tracing. In *Intelligent Tutoring Systems* (pp. 406-415). Springer Berlin Heidelberg.
12. Baker, R., Walonoski, J., Heffernan, N., Roll, I., Corbett, A., & Koedinger, K. (2008). Why students engage in "gaming the system" behavior in interactive learning environments. *Journal of Interactive Learning Research*, 19(2), 185-224.

13. Bakshy, E., Eckles, D., & Bernstein, M. S. (2014). Designing and deploying online field experiments. In *Proceedings of the 23rd international conference on World wide web* (pp. 283-292). ACM.
14. Barnes, T. (2005). The Q-matrix method: Mining student response data for knowledge. In *American Association for Artificial Intelligence 2005 Educational Data Mining Workshop*.
15. Basford, K. E., Greenway, D. R., McLachlan, G. J., & Peel, D. (1997). Standard errors of fitted component means of normal mixtures. *Computational Statistics*, 12(1), 1-18.
16. Beck, J. E., Chang, K. M., Mostow, J., & Corbett, A. (2008). Does help help? Introducing the Bayesian Evaluation and Assessment methodology. *In Intelligent Tutoring Systems* (pp. 383-394). Springer Berlin Heidelberg.
17. Berry, D. A., Chen, R. W., Zame, A., Heath, D. C., & Shepp, L. A. (1997). Bandit problems with infinitely many arms. *The Annals of Statistics*, 2103-2116.
18. Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher*, 13(6), 4-16.
19. Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive science*, 2(2), 155-192.
20. Brown, J. S., & VanLehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive science*, 4(4), 379-426.
21. Brys, T., Van Moffaert, K., Van Vaerenbergh, K., & Nowé, A. (2013). On the behaviour of scalarization methods for the engagement of a wet clutch. In *Machine Learning and Applications (ICMLA), 2013 12th International Conference on* (Vol. 1, pp. 258-263). IEEE.
22. Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In *Intelligent tutoring systems*. Academic press.
23. Burton, R. B. (1982). DEBUGGY: Diagnosis of errors in basic mathematical skills. *Intelligent Tutoring Systems. Academic Press, London*.
24. CAST (2011). Universal Design for Learning Guidelines version 2.0. Wakefield, MA: Author.

25. Cen, H. (2009). Generalized Learning Factor Analysis: Improving Cognitive Models with Machine Learning. Doctoral Thesis. Carnegie Mellon University.
26. Cen, H., Koedinger, K., & Junker, B. (2006). Learning factors analysis–a general method for cognitive model evaluation and improvement. In *Intelligent tutoring systems* (pp. 164-175). Springer Berlin Heidelberg.
27. Chen, Y., Mandel, T., Liu, Y. E., & Popovic, Z. (2016). Crowdsourcing Accurate and Creative Word Problems and Hints. *In the proceedings of the third AAAI Conference on Human Computation and Crowdsourcing* (H-Comp-15).
28. Clements, D. H., & Sarama, J. (2004). Learning trajectories in mathematics education. *Mathematical thinking and learning*, *6*(2), 81-89.
29. Corbett, A. T., & Anderson, J. R. (1994). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, *4*(4), 253-278.
30. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to algorithms 2nd edition.
31. Das, I., & Dennis, J. E. (1997). A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. *Structural and multidisciplinary optimization*, *14*(1), 63-69.
32. DeMars, C. E. (2008). Scoring multiple choice items: A comparison of IRT and classical polytomous and dichotomous methods. In *annual meeting of the National Council of Measurement in Education, New York*.
33. Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological), 1-38*.
34. Deng, A. Lu, J., Chen, S. (2016) Continuous monitoring of A/B tests without pain: Optional stopping in Bayesian testing (ArXiv ver.)
35. Deng, A. (2015). Objective bayesian two sample hypothesis testing for online controlled experiments. In Proceedings of the 24th *International Conference on World Wide Web* (pp. 923-928). ACM.
36. Desmarais, M. C. and Gagnon M. (2005) Bayesian Student Models Based on Item to Item Knowledge Structures. Artificial Intelligence in Education. Lecture Notes in Computer Science. 4227:111-124

37. Dillenbourg, P. (2002). Over-scripting CSCL: The risks of blending collaborative learning with instructional design. *Three worlds of CSCL. Can we support CSCL?*, 61-91.
38. Dommeti, V., & Selent, D. (2017). Applying and Exploring Bayesian Hypothesis Testing for Large Scale Experimentation in Online Tutoring Systems. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale* (pp. 229-232). ACM.
39. Drugan, M. M., & Nowe, A. (2013). Designing multi-objective multi-armed bandits algorithms: A study. In *Neural Networks (IJCNN), The 2013 International Joint Conference on* (pp. 1-8). IEEE.
40. Drugan, M. M., Nowé, A., & Manderick, B. (2014). Pareto upper confidence bounds algorithms: an empirical study. In *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2014 IEEE Symposium on* (pp. 1-8). IEEE.
41. Efron, B. (2013). A 250-year argument: Belief, behavior, and the bootstrap. *Bulletin of the American Mathematical Society*, 50(1), 129-146.
42. Eichfelder, G. (2009). An adaptive scalarization method in multiobjective optimization. *SIAM Journal on Optimization*, *19*(4), 1694-1718.
43. Embretson, S. E. (1998). A cognitive design system approach to generating valid tests: Application to abstract reasoning. *Psychological Methods*, 3(3), 380-396.
44. Fischer, F., Kollar, I., Stegmann, K., & Wecker, C. (2013). Toward a script theory of guidance in computer-supported collaborative learning. *Educational psychologist*, *48*(1), 56-66.
45. Floryan, M., & Woolf, B. P. (2013). Authoring Expert Knowledge Bases for Intelligent Tutors through Crowdsourcing. *In International Conference on Artificial Intelligence in Education* (pp. 640-643). Springer Berlin Heidelberg.
46. Garivier, A., & Moulines, E. (2008). On upper-confidence bound policies for non-stationary bandit problems. *Arxiv preprint*.
47. Gagné, R. (1968). "Learning Hierarchies." *Educational Psychologist* 6:1-9.
48. Gertner, A. S., & VanLehn, K. (2000). Andes: A coached problem solving environment for physics. *In International conference on intelligent tutoring systems* (pp. 133-142). Springer Berlin Heidelberg.

49. Gierl, M. J., Leighton, J. P., & Hunka, S. (2007). Using the attribute hierarchy method to make diagnostic inferences about examinees' cognitive skills. *Cognitive diagnostic assessment for education: Theory and applications*, 242-274.
50. Gierl, M. J., Wang, C., & Zhou, J. (2008). Using the Attribute Hierarchy Method to Make Diagnostic Inferences about Examinees' Cognitive Skills in Algebra on the SAT. *Journal of Technology, Learning, and Assessment*, 6(6), n6.
51. Gittins, J. C. (1979). Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 148-177.
52. Gong, Y., Beck, J. E., & Heffernan, N. T. (2012). WEBsistments: enabling an intelligent tutoring system to excel at explaining rather than coaching. In *Intelligent Tutoring Systems* (pp. 268-273). Springer Berlin Heidelberg.
53. Heffernan, N. & Heffernan, C. (2014). The ASSISTments Ecosystem: Building a Platform that Brings Scientists and Teachers Together for Minimally Invasive Research on Human Learning and Teaching. *International Journal of AIED*, 24(4), 470-497.
54. Heffernan, N.T., Ostrow, K.S., Kelly, K., Selent, D., Van Inwegen, E.G., Xiong, X., & Williams, J.J. (2016). The Future of Adaptive Learning: Does the Crowd Hold the Key? *International Journal of Artificial Intelligence in Education*. Springer New York. DOI: 10.1007/s40593-016-0094-z.
55. Ioannidis, J. P. A. (2005A). Contradicted and Initially Stronger Effects in Highly Cited Clinical Research. JAMA: *The Journal of the American Medical Association* 294 (2): 218–228. doi:10.1001/jama.294.2.218.
56. Ioannidis J. P.A. (2005B). Why Most Published Research Findings Are False. *PLoS Med* 2(8): e124. doi:10.1371/journal.pmed.0020124.
57. Jarvis, M. P., Nuzzo-Jones, G., & Heffernan, N. T. (2004). Applying machine learning techniques to rule generation in intelligent tutoring systems. *In Intelligent Tutoring Systems* (pp. 541-553). Springer Berlin Heidelberg.
58. Johnson, R., Hoeller, J., Arendsen, A., Sampaleanu, C., Davison, D., Kopylenko, D., & Harro, R. (2004). Spring-Java/J2EE application framework. *Reference Documentation, Version*, *1*(7), 265-278.
59. Kass, R. E., & Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430), 773-795.

60. Kim, J. (2015). Learnersourcing: Improving Learning with Collective Learner Activity. MIT PhD.
61. Kocsis, L., & Szepesvári, C. (2006). Discounted ucb. In *2nd PASCAL Challenges Workshop*.
62. Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B., & Hockenberry, M. (2004). Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. *In Intelligent Tutoring Systems* (pp. 7-10). Springer Berlin/Heidelberg.
63. Koedinger, K.R. & Corbett, A.T. (2006). Cognitive tutors: Technology bringing learning science to the classroom. In K. Sawyer (Ed.), *The Cambridge handbook of the learning sciences* (61-78). New York: Cambridge University Press.
64. Kohavi, R., Deng, A., Frasca, B., Walker, T., Xu, Y., & Pohlmann, N. (2013). Online controlled experiments at large scale. *In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1168-1176). ACM.
65. Kollar, I., Fischer, F., & Slotta, J. D. (2007). Internal and external scripts in computer-supported collaborative inquiry learning. *Learning and Instruction*,*17*(6), 708-721.
66. King, A. (2002). Structuring peer interaction to promote high-level cognitive processing. *Theory into practice*, *41*(1), 33-39.
67. Kruschke, J. K. (2013). Bayesian estimation supersedes the t test. *Journal of Experimental Psychology*: General, 142(2), 573.
68. Lai, T. L., & Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, *6*(1), 4-22.
69. Lang, C., Heffernan, N., Ostrow, K., & Wang, Y. (2015). The Impact of Incorporating Student Confidence Items into an Intelligent Tutor: A Randomized Controlled Trial. *International Educational Data Mining Society*.
70. Langley, P. W., Ohlsson, S., & Sage, S. (1984). A machine learning approach to student modeling.
71. Leighton, J. P., Gierl, M. J., & Hunka, S. M. (2004). The attribute hierarchy method for cognitive assessment: A variation on Tatsuoka's rule-space approach. *Journal of Educational Measurement*, 41, 205-236.

72. Leszczenski, J.M., Beck, J.E. (2007). What's in a Word? Extending Learning Factors Analysis to Model Reading Transfer. *13th International Conference on Artificial Intelligence in Education, Educational Data Mining Workshop,* 2007. Los Angeles, CA.
73. Li, B., Selent D. (2014). The Hidden Value of Common Wrong Answers (Interactive Qualifying Project, Worcester Polytechnic Institute).
74. Li N., Cohen W. W., Noboru M. and Koedinger K. R. (2011) 'A machine learning approach for automatic student model discovery', in *Proceedings of the 4th International Conference on Educational Data Mining*, pp. 31–40.
75. Liu, Y. E., Mandel, T., Brunskill, E., & Popovic, Z. (2014). Trading off scientific knowledge and user learning with multi-armed bandits. In *Educational Data Mining 2014*.
76. Lobo, Jorge M., Alberto Jiménez-Valverde, and Raimundo Real. (2008) "AUC: a misleading measure of the performance of predictive distribution models." *Global ecology and Biogeography* 17.2 (2008): 145-151
77. Marler, R. T., & Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, *41*(6), 853-862.
78. Matsuda, N., Yarzebinski, E., Keiser, V., Raizada, R., Stylianides, G. J., Cohen, W. W., & Koedinger, K. R. (2011). Learning by teaching SimStudent–An initial classroom baseline study comparing with Cognitive Tutor. In *Artificial Intelligence in Education* (pp. 213-221). Springer Berlin Heidelberg.
79. Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2007). Predicting students' performance with SimStudent: Learning cognitive skills from observation. *Frontiers in Artificial Intelligence and Applications*, *158*, 467.
80. Michael, J. (2007). 40000 Namen, Anredebestimmung anhand des Vornamens.
81. Mislevy, R. J., Steinberg, L. S., & Almond, R. G. (2002). Design and analysis in task-based language assessment. *Language Testing*, 19(4), 477-496.
82. Munroe, R. (N.D.) Frequentists vs. Bayesians. http://xkcd.com/1132/.
83. Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education* (IJAIED), 10, 98-129.
84. Narzisi, G. (2008). Multi-objective optimization. *A quick introduction, New York University lectures*.

85. Nathan, M. J., Koedinger, K. R., & Alibali, M. W. (2001). Expert blind spot: When content knowledge eclipses pedagogical content knowledge. *In Proceedings of the Third International Conference on Cognitive Science* (pp. 644-648). Beijing: University of Science and Technology of China Press.
86. Nathan, M. J., & Petrosino, A. (2003). Expert blind spot among preservice teachers. *American educational research journal*, 40(4), 905-928.
87. Ostrow, K. S. (2015). A Multifaceted Consideration of Motivation and Learning within ASSISTments (Master's Thesis, Worcester Polytechnic Institute).
88. Ostrow, K., Donnelly, C., & Heffernan, N. (2015). Optimizing Partial Credit Algorithms to Predict Student Performance. *International Educational Data Mining Society*.
89. Ostrow, K. S., & Heffernan, N. T. (2015). The Role of Student Choice Within Adaptive Tutoring. *In Artificial Intelligence in Education* (pp. 752-755). Springer International Publishing.
90. Ostrow, K., & Heffernan, N. (2014). Testing the multimedia principle in the real world: a comparison of video vs. Text feedback in authentic middle school math assignments. In *Educational Data Mining* 2014.
91. Ostrow, K., Selent, D., Wang, Y., Van Inwegen, E., Heffernan, N., & Williams, J.J. (2016) Assessment of Learning Infrastructure (ALI): The Theory, Practice, and Scalability of Automated Assessment. *In Proceedings of the Sixth International Conference on Learning Analytics & Knowledge*. ACM.
92. Ottmar, E., Weitnauer, E., Landy, D., & Goldstone, R. (2015). Graspable mathematics: Using perceptual learning technology. *Integrating Touch-Enabled and Mobile Devices into Contemporary Mathematics Education*, 24.
93. Pardos, Z. & Heffernan, N. (2010). Navigating the parameter space of Bayesian Knowledge Tracing models: Visualization of the convergence of the Expectation Maximization algorithm. In Baker, R.S.J.d., Merceron, A., Pavlik, P.I. Jr. (Eds.) *Proceedings of the 3rd International Conference on Educational Data Mining*. Pages 161-170.
94. Patikorn, T., Selent, D., Heffernan, N., Beck J.E., Zou, J. (2017) Using a Single Model Trained across Multiple Experiments to Improve the Detection of Treatment Effects *Educational Data Mining 2017.*

95. Patikorn, T., Selent, D., Heffernan, N. T., Yin, B., Botelho, A. (2016) ASSISTments Dataset for a Data Mining Competition to Improve Personalized Learning. Poster at the *Conference on Digital Experimentation* at MIT 2016.

96. Pavlik Jr, P. I., Cen, H., & Koedinger, K. R. (2009). Learning factors transfer analysis: Using learning curve analysis to automatically generate domain models. In *Educational Data Mining 2009*.

97. Pavlik Jr, P. I., Cen, H., & Koedinger, K. R. (2009). Performance Factors Analysis--A New Alternative to Knowledge Tracing. *In the Proceedings of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*.

98. Piech, C., Sahami, M., Huang, J., & Guibas, L. (2015). Autonomously generating hints by inferring problem solving policies. *In Proceedings of the Second (2015) ACM Conference on Learning@ Scale* (pp. 195-204). ACM.

99. Popham, W. J. (2011). *Transformative assessment in action: An inside look at applying the process*. ASCD.

100. Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine learning*, *5*(3), 239-266.

101. Quinlan, J. R., & Cameron-Jones, R. M. (1993). FOIL: A midterm report. In *Machine Learning: ECML-93* (pp. 1-20). Springer Berlin Heidelberg.

102. Rafferty, A. N., & Yudelson, M. (2007). Applying learning factors analysis to build stereotypic student models. *Frontiers in Artificial Intelligence and Applications*, *158*(697), 51.

103. Razzaq, L. M., Feng, M., Nuzzo-Jones, G., Heffernan, N. T., Koedinger, K. R., Junker, B., & Upalekar, R. (2005). Blending Assessment and Instructional Assisting. In *AIED* (pp. 555-562).

104. Razzaq, L., & Heffernan, N. T. (2010). Hints: is it better to give or wait to be asked?. In *Intelligent Tutoring Systems* (pp. 349-358). Springer Berlin Heidelberg.

105. Razzaq, L., Patvarczki, J., Almeida, S. F., Vartak, M., Feng, M., Heffernan, N. T., & Koedinger, K. R. (2009). The Assistment Builder: Supporting the life cycle of tutoring system content creation. *Learning Technologies, IEEE Transactions on*, *2*(2), 157-166.

106. Reese, W. (2008). Nginx: the high-performance web server and reverse proxy. *Linux Journal*, *2008*(173), 2.

107. Rivers, K., & Koedinger, K. R. (2013). Automatic generation of programming feedback: A data-driven approach. *In The First Workshop on AI-supported Education for Computer Science* (AIEDCS 2013) (Vol. 50).

108. Robbins, H. (1952). Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers* (pp. 169-177). Springer New York.

109. Roschelle, J. Feng, M. Murphy, R. Mason, C. (2016). Online Mathematics Homework Increases Student Achievement. AERA Open, 2 (4) 2332858416673968; DOI: 10.1177/2332858416673968.

110. Roscoe, R. D., & Chi, M. T. (2007). Understanding tutor learning: Knowledge-building and knowledge-telling in peer tutors' explanations and questions. *Review of Educational Research*, *77*(4), 534-574.

111. Rosenthal, R. (1979). The file drawer problem and tolerance for null results. *Psychological bulletin*, 86(3), 638.

112. Saxton, G. D., Oh, O., & Kishore, R. (2013). Rules of crowdsourcing: Models, issues, and systems of control. *Information Systems Management*,*30*(1), 2-20.

113. Schank, R. C. (1999). *Dynamic memory revisited*. Cambridge, MA: Cambridge University Press. doi:10.1017/CBO9780511527920

114. Selent, D., & Heffernan, N. (submitted). PeerASSIST: A System to Crowdsource and Redistribute Student Work in Real-Time to Peers in Need of Assistance. In *Fifth AAAI Conference on Human Computation and Crowdsourcing*.

115. Selent, D., & Heffernan, N. (2015). When More Intelligent Tutoring in the Form of Buggy Messages Does not Help. In *Artificial Intelligence in Education* (pp. 768-771). Springer International Publishing.

116. Selent, D., & Heffernan, N. (2014). Reducing Student Hint Use by Creating Buggy Messages from Machine Learned Incorrect Processes. In *Intelligent Tutoring Systems* (pp. 674-675). Springer International Publishing.

117. Selent, D., Patikorn, T., & Heffernan, N. (2016). ASSISTments Dataset from Multiple Randomized Controlled Experiments. *In Proceedings of the Third (2016) ACM Conference on Learning@ Scale* (pp. 181-184). ACM.

118. Sharma, A. (2010). Crowdsourcing Critical Success Factor Model: Strategies to harness the collective intelligence of the crowd. London School of Economics (LSE), London.

119. Sison, R., & Shimura, M. (1998). Student modeling and machine learning. *International Journal of Artificial Intelligence in Education* (IJAIED), 9, 128-158.

120. Sleeman, D. (1984). Mis-generalization: An Explanation of Observed Mal-rules.

121. Sleeman, D. (1985). Basic algebra revisited: a study with 14-year-olds.*International Journal of Man-Machine Studies*, *22*(2), 127-149.

122. Smeets, B., Boness, U., & Bankras, R. (2008). *Beginning Google Web Toolkit*. Apress.

123. Stamper, J., Barnes, T., Lehmann, L., & Croy, M. (2008). The hint factory: Automatic generation of contextualized help for existing computer aided instruction. *In Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young Researchers Track* (pp. 71-78).

124. Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning*. MIT Press.

125. Tatsuoka, K. K. (1983), Rule space: an approach for dealing with misconceptions based on item response theory. *Journal of Educational Measurement*, 20: 345–354.

126. Tatsuoka, K. K. (1995). Architecture of knowledge structures and cognitive diagnosis: A statistical pattern recognition and classification approach. *Cognitively diagnostic assessment*, 327-359.

127. Teytaud, O., Gelly, S., & Sebag, M. (2007). Anytime many-armed bandits. In*CAP07*.

128. Thissen-Roe, A., Hunt, E., & Minstrell, J. (2004). The DIAGNOSER project: Combining assessment and learning. *Behavior research methods, instruments, & computers*, *36*(2), 234-240.

129. Topping, K. J., Peter, C., Stephen, P., & Whale, M. (2004). Cross-age peer tutoring of science in the primary school: Influence on scientific language and thinking. *Educational Psychology*, *24*(1), 57-75.

130. Van de Walle, J. A., Bay-Williams, J. M., Karp, K. S., Lovin, L. H. (2014). Teaching Student-Centered Mathematics: Developmentally Appropriate Instruction for Grades 6-8 (2nd ed.). Upper Saddle River, NJ: Pearson Inc.

131. Van der Linden, W. and Hambleton, R. (1997) (Eds.). *Handbook of Modern Item Response Theory*. New York, NY: Springer-Verlag.

132. VanInwegen, E. G., Adjei, S. A., Wang, Y., & Heffernan, N. T. (2015). Using Partial Credit and Response History to Model User Knowledge. *International Educational Data Mining Society*.

133. VanLehn, K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. *The Journal of Mathematical Behavior*.

134. VanLehn, K., Siler, S., Murray, C., & Baggett, W. (1998). What makes a tutorial event effective. In *Proceedings of the Twenty-first Annual Conference of the Cognitive Science Society* (pp. 1084-1089).

135. VanLehn, K., Siler, S., Murray, C., Yamauchi, T., & Baggett, W. B. (2003). Why do only some events cause learning during human tutoring?. *Cognition and Instruction*, *21*(3), 209-249.

136. Wang, Q., Kehrer, P., Pardos, Z., & Heffernan, N. (2011). Response tabling-a simple and practical complement to knowledge tracing. In *KDD workshop*.

137. Wang, Y., Audibert, J. Y., & Munos, R. (2009). Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems* (pp. 1729-1736).

138. Wang, Y., & Heffernan, N. (2013, July). Extending knowledge tracing to allow partial credit: Using continuous versus binary nodes. In *International Conference on Artificial Intelligence in Education* (pp. 181-188). Springer Berlin Heidelberg.

139. Wang, Y., Heffernan, N. T., & Beck, J. E. (2010). Representing student performance with partial credit. In *Educational Data Mining 2010*.

140. Webb, N. M., & Mastergeorge, A. (2003). Promoting effective helping behavior in peer-directed groups. *International Journal of Educational Research*, *39*(1), 73-97.

141. Webb, N. M., & Palincsar, A. S. (1996). *Group processes in the classroom*. Prentice Hall International.

142. Wegscheider, K. (1998). Advantages and disadvantages of sequential designs.*Herzschrittmachertherapie und Elektrophysiologie*, *9*(4), 248-254.

143. Weinberger, A., Fischer, F., & Mandl, H. (2004). Knowledge convergence in computer-mediated learning environments. Effects of collaboration scripts. In*Annual Meeting of the American Educational Research Association (AERA 2004)* (p. 7).

144. Whitehill, J., & Seltzer, M. (2017). A Crowdsourcing Approach to Collecting Tutorial Videos--Toward Personalized Learning-at-Scale. *In Proceedings of the Fourth ACM Conference on Learning@ Scale* (pp. 157-160). ACM.

145. Williams, J. J., Kim, J., Rafferty, A., Maldonado, S., Gajos, K. Z., Lasecki, W. S., & Heffernan, N. (2016). Axis: Generating explanations at scale with learnersourcing and machine learning. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale* (pp. 379-388). ACM.

146. Wu, C. J. (1983). On the convergence properties of the EM algorithm. *The Annals of Statistics*, 95-103.

147. Xiong, X., Li, S., & Beck, J. E. (2013). Will You Get It Right Next Week: Predict Delayed Performance in Enhanced ITS Mastery Cycle. *In FLAIRS Conference*.

148. Xu, Y., & Mostow, J. (2011). Logistic Regression in a Dynamic Bayes Net Models Multiple Subskills Better! In *Educational Data Mining 2011*.

149. Yahyaa, S. Q., Drugan, M. M., & Manderick, B. (2014). Knowledge Gradient for Multi-objective Multi-armed Bandit Algorithms. In *ICAART (1)* (pp. 74-83).

150. Yahyaa, S., & Manderick, B. (2015). Thompson sampling for multi-objective multi-armed bandits problem. In *Proceedings* (p. 47). Presses universitaires de Louvain.

Appendix A: SQL Queries for Peer Work Data

Main Query

```
SELECT
pe.*,
pl.id AS problem_log_id,
pl.problem_id,
next_question_stats.distributed_count,
student_votes.student_up_votes,
student_votes.student_down_votes,
student_votes.student_cleared_votes,
teacher_votes.teacher_up_votes,
teacher_votes.teacher_down_votes,
teacher_votes.teacher_cleared_votes,
--next_question_stats.peer_explanation_id,
next_question_stats.next_question_correct_count,
next_question_stats.next_question_incorrect_count,
next_question_stats.next_question_count,
next_question_stats.finished_count
FROM peer_explanations pe
--join with the problem log where the explanation was created
INNER JOIN problem_logs_to_explanations_associations pltea ON pe.id = pltea.peer_explanation_id
INNER JOIN problem_logs pl ON pl.id = pltea.problem_log_id

--student votes
LEFT OUTER JOIN
(
	SELECT
	object_id,
	SUM(CASE WHEN rating = 1 THEN 1 ELSE 0 END) AS student_up_votes,
	SUM(CASE WHEN rating = -1 THEN 1 ELSE 0 END) AS student_down_votes,
	SUM(CASE WHEN rating = 0 THEN 1 ELSE 0 END) AS student_cleared_votes
	FROM ratings
	WHERE object_id IN
	(
		SELECT DISTINCT(id)
		FROM peer_explanations
	)
	--AND rating = 1
	--2 = explanation
	AND rating_object_type_id = 2

	--student
	AND user_id NOT IN
	(
		SELECT user_roles.user_id
		FROM user_roles
		GROUP BY user_id
		HAVING COUNT(user_id) > 1
	)

	--only for data from the same policy
	AND
	(
		object_id IN
		(
			SELECT rating_id
			FROM rating_to_policy_instance_associations
			WHERE policy_instance_id IN
			(
				:piece1
			)
		)

		OR object_id NOT IN
```

```
                (
                        SELECT rating_id
                        FROM rating_to_policy_instance_associations
                        WHERE policy_instance_id IN
                        (
                                :piece2
                        )
                )
        )

        GROUP BY object_id
) AS student_votes
ON student_votes.object_id = pe.id

--teacher votes
LEFT OUTER JOIN
(
        SELECT
        object_id,
        SUM(CASE WHEN rating = 1 THEN 1 ELSE 0 END) AS teacher_up_votes,
        SUM(CASE WHEN rating = -1 THEN 1 ELSE 0 END) AS teacher_down_votes,
        SUM(CASE WHEN rating = 0 THEN 1 ELSE 0 END) AS teacher_cleared_votes
        FROM ratings
        WHERE object_id IN
        (
                SELECT DISTINCT(id)
                FROM peer_explanations
        )
        --AND rating = 1
        --2 = explanation
        AND rating_object_type_id = 2

        --teacher
        AND user_id IN
        (
                SELECT user_roles.user_id
                FROM user_roles
                GROUP BY user_id
                HAVING COUNT(user_id) > 1
        )

        --only for data from the same policy
        AND
        (
                object_id IN
                (
                        SELECT rating_id
                        FROM rating_to_policy_instance_associations
                        WHERE policy_instance_id IN
                        (
                                :piece1
                        )
                )

                OR object_id NOT IN
                (
                        SELECT rating_id
                        FROM rating_to_policy_instance_associations
                        WHERE policy_instance_id IN
                        (
                                :piece2
                        )
                )
        )

        GROUP BY object_id
) AS teacher_votes
ON teacher_votes.object_id = pe.id

--next question stats
```

```
LEFT OUTER JOIN
(
	--main join can have blank rows indicating no start time
	SELECT de.peer_explanation_id AS peer_explanation_id,
	COUNT(de.peer_explanation_id) AS distributed_count,
	SUM(CASE WHEN (next_question.correct >= 1 AND next_question.end_time IS NOT NULL) THEN 1 ELSE 0 END) AS
next_question_correct_count,
	SUM(CASE WHEN (next_question.correct < 1 AND next_question.end_time IS NOT NULL) THEN 1 ELSE 0 END) AS
next_question_incorrect_count,
	COUNT(de.id) AS next_question_count,
	COUNT(next_question.end_time) AS finished_count
	FROM distributed_explanations de
	LEFT OUTER JOIN
	(
		SELECT p1.id AS current_log_id, p2.id AS next_log_id, p2.correct, p2.end_time
		FROM problem_logs p1
		LEFT OUTER JOIN problem_logs p2
		ON p1.user_id = p2.user_id AND p1.assignment_id = p2.assignment_id
		WHERE p1.problem_id = :problemId
		AND p2.id IN
		(
			SELECT MIN(id)
			FROM problem_logs
			WHERE id > p1.id
			AND user_id = p2.user_id
			AND assignment_id = p2.assignment_id
		)
	) AS next_question
	ON de.problem_log_id = next_question.current_log_id

	--only for data from the same policy
	WHERE de.policy_instance_id IN
	(
		:piece1
	)

	GROUP BY de.peer_explanation_id
) AS next_question_stats
ON next_question_stats.peer_explanation_id = pe.id

WHERE sharable = TRUE
--do not give student's their own explanations
AND pl.user_id <> :userId
AND pl.problem_id = :problemId

AND pe.id NOT IN
(
	SELECT peer_explanation_id
	FROM explanation_flaggings
	WHERE peer_explanation_id NOT IN
	(
		SELECT peer_explanation_id
		FROM explanation_unflaggings
		WHERE active = 'true'
	)
)

--optional
--only explanations from same teacher as user (or are in the same class and have also created an explanation on the same problem in another
class...)
AND pe.user_id IN
(
	SELECT user_id
	FROM user_roles
	WHERE id IN
	(
		SELECT student_id
		FROM enrollments
		WHERE student_class_id IN
		(
```

```
                                SELECT student_class_id
                                FROM teacher_classes
                                WHERE teacher_id IN
                                (
                                        SELECT teacher_id
                                        FROM teacher_classes
                                        WHERE student_class_id = :classId
                                )
                        )
                )
                AND role_id = 4
)

--only the student's last explanation for a given problem log
AND pe.id IN
(
        SELECT MAX(peer_explanation_id)
        FROM peer_explanations pe
        INNER JOIN problem_logs_to_explanations_associations pltea ON pe.id = pltea.peer_explanation_id
        WHERE sharable = TRUE
        AND pltea.problem_log_id IN
        (
                SELECT id FROM problem_logs WHERE problem_id = :problemId AND end_time IS NOT NULL AND correct = 1
        )
        GROUP BY pe.user_id, pltea.problem_log_id
)

ORDER BY pe.id DESC
```

Piece 1 Query

```
SELECT id
FROM peer_work_policy_instances
WHERE policy_id IN
(
        SELECT policy_id
        FROM peer_work_policy_instances
        WHERE id = :policyInstanceId
)
```

Piece 2 Query

```
SELECT id
FROM peer_work_policy_instances
WHERE policy_id IN
(
        SELECT policy_id
        FROM peer_work_policy_instances
)
```

Appendix B: Questions from Chapter 2

1. Which statement about 3+ (-2) and 3 2 best describes the meaning of the symbols "+" and " "?

A) 3 + (-2) and 3 2 both mean subtracting 2 from 3

B) 3 + (-2) and 3 2 both mean adding 3 to the opposite of 2

C) 3 + (-2) means subtracting 2 from 3, and 3 2 means adding 3 to the opposite of 2

D) X 3 + (-2) means adding 3 to the opposite of 2, and 3 2 means subtracting 2 from 3

Question Id: 50049, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

2. Which statement about 6 (-5) and 6 + 5 best describes the meaning of the symbols "+" and " "?

A) 6 (-5) and 6 + 5 both mean adding 6 and 5

B) 6 (-5) and 6 + 5 both mean subtracting the opposite of 5 from 6

C) X 6 (-5) means subtracting the opposite of 5 from 6, and 6 + 5 means adding 6 and 5

D) 6 (-5) means adding 6 and 5, and 6 + 5 means subtracting the opposite of 5 from 6

Question Id: 50050, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

3. Which statement about integers is true?

A) X -3 is an integer because 3 is an integer

B) is an integer because both 1 and 2 are integers

C) 0.75 is an integer because 75 is an integer

D) -1.26 is an integer because both -1 and 26 are integers

Question Id: 50051, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

4. Which is always an integer?

A) positive fraction

B) negative decimal

C) improper fraction

D) X zero

Question Id: 50052, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

5. Which correctly explains why -5 is the opposite of 5?

A) X -5 added to 5 equals 0

B) 0 is to the left of 5 on a number line

C) -5 is to the left of 5 on a number line

D) they are located on different sides of 0

Question Id: 50053, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

6. Which choice contains a pair of opposite numbers?

A) -5,

B) -5,

C) -5, -5

D) X -5, 5

Question Id: 50054, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

7. Which statement comparing two integers is correct?

A) -1,537 > 1,576

B) -1,537 < -1,576

C) 1,537 < -1,576

D) X -1,537 > -1,576

Question Id: 50055, Standard 6 "NS", Benchmark 7 "7", Indicator "a", Sub Indicator ""

8. Which expression comparing two integers is correct?

A) -2,789 > 2,980

B) X -2,789 > -2,980

C) 2,789 < -2,980

D) -2,789 < -2,980

Question Id: 50056, Standard 6 "NS", Benchmark 7 "7", Indicator "a", Sub Indicator ""

9. Which list of temperatures is in order from **highest** to **lowest**?

A) X 45 degrees F, 2 degrees F, -32 degrees F, -38 degrees F, -40 degrees F

B) 45 degrees F, -40 degrees F, -38 degrees F, -32 degrees F, 2 degrees F

C) 2 degrees F, -32 degrees F, -38 degrees F, -40 degrees F, 45 degrees F

D) -40 degrees F, -38 degrees F, -32 degrees F, 2 degrees F, 45 degrees F

Question Id: 50057, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

10. A business's net daily income for five days is shown in the table below.

Net Daily Income

	Monday	Tuesday	Wednesday	Thursday	Friday
Net Income	-$220	$170	$125	-$140	$185

Which list of incomes is in order from **least** to **greatest**?

A) -$220, $185, $170, -$140, $125

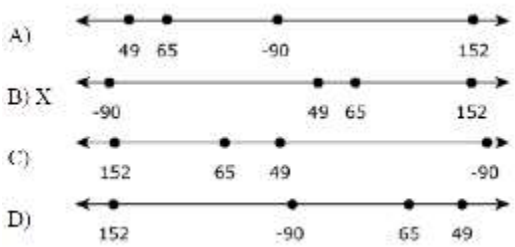
B) $125, -$140, $170, $185, -$220

C) X -$220, -$140, $125, $170, $185

D) $185, $170, $125, -$140, -$220

Question Id: 50058, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

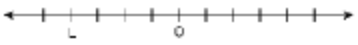
11. The school's band members are selling cookies to raise money. The band members need to pay for unsold boxes. The table below shows the total sales, including fees for unsold boxes of each member.



Question Id: 50059, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

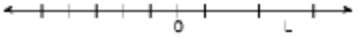
12. Each graph displays points L and 0 on a number line. On which number line is point L positioned at the value of

A)

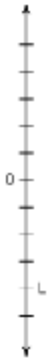


B)

C) X



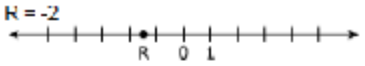
D)



Question Id: 50060, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

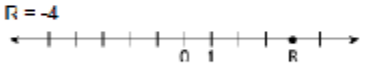
13. Which graph correctly represents the integer shown?

A)

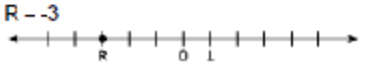


B)

C)

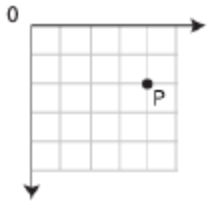


D) X



Question Id: 50061, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

14. Point P is plotted in the graph shown below. Which coordinate pair lists the correct values for P?



A) (-2, 4)

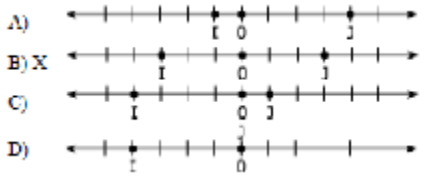
B) (-5, 4)

C) X (4, -2)

D) (4, -5)

Question Id: 50062, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

15. Which number line shows that I is the opposite of J?



Question Id: 50063, Standard 6 "NS", Benchmark 6 "6", Indicator "a", Sub Indicator ""

16. Which procedure describes correctly how to graph the ordered pair (-2, 3) on the coordinate plane?

A) X Starting from the origin, move 2 units to the left, and move 3 units up.

B) Starting from the origin, move 2 units to the right, and move 3 units up.

C) Starting from the origin, move 2 units to the left, and move 3 units down.

D) Starting from the origin, move 2 units to the right, and move 3 units down.

Question Id: 50064, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

17. The freezing point of water is 0 degrees C. Which temperature is 10 degrees below the freezing point of water?

A) 10 degrees C

B) 100 degrees C

C) X -10 degrees C

D) -100 degrees C

Question Id: 50065, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

18. Antonio's account had a $5 balance before he went to the bank to make a deposit. What is his balance after putting $25 into this account?

A) $0

B) -$20

C) $25

D) X $30

Question Id: 50066, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

19. Points R and T are plotted on the number line shown below.

Which sentence correctly explains the relationship between R and T?

A) X R is less than T, because R is to the left of T on the number line

B) T is less than R, because T is to the right of R on the number line

C) R is greater than T, because the number line extends to the left and right of R

D) T is greater than R, because the number line extends to the left and right of T

Question Id: 50067, Standard 6 "NS", Benchmark 7 "7", Indicator "a", Sub Indicator ""

20. Bob owed $50 to a business. Then Bob paid $50 to the business. Which statement is true?

A) X Bob now owes the business $0.

B) Bob now owes the business $100.

C) The business now owes Bob $50.

D) The business now owes Bob $100.

Question Id: 50068, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

21. The elevation of Village A is 5 meters below sea level. Village B is 4 meters above Village A. What is Village B's elevation, in relation to sea level?

A) 1 meter above sea level

B) X 1 meter below sea level

C) 9 meters above sea level

D) 9 meters below sea level

Question Id: 50069, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

22. Which sentence correctly explains why a temperature of 3 degrees below 0 is warmer than a temperature of 7 degrees below 0?

A) X -3 is greater than -7

B) -3 is not equal to -7

C) -3 is equal to -7

D) -3 is less than -7

Question Id: 50070, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

23. Which sentence below correctly explains why 6 meters below sea level is lower than 2 meters below sea level?

A) -6 is greater than -2

B) -6 is not equal to -2

C) X -6 is less than -2

D) -6 is equal to -2

Question Id: 50071, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

24. Family J spent $3\ ^{1}/_{2}$ hours traveling to a campsite. Departing from the same place, Family K spent $3\ ^{5}/_{6}$ hours traveling to the same campsite. Which statement correctly compares the two families' traveling time?

A) Family J spent more time traveling because $3\ ^{1}/_{2} > 3\ ^{5}/_{6}$.

B) X Family J spent less time traveling because $3\ ^{1}/_{2} < 3\ ^{5}/_{6}$.

C) Family K spent more time traveling because $3\ ^{1}/_{2} > 3\ ^{5}/_{6}$.

D) Family K spent less time traveling because $3\ ^{1}/_{2} < 3\ ^{5}/_{6}$.

Question Id: 50072, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

25. Two gas pipelines are being placed in a town. Pipeline A is placed at a depth of $2\ ^{3}/_{7}$ meters below the ground and pipeline B is placed at a depth of $3\ ^{1}/_{4}$ meters below the ground. Which statement is correct?

A) Pipeline A is deeper than pipeline B because $-2\ ^{3}/_{7} < -3\ ^{1}/_{4}$.

B) X Pipeline B is deeper than pipeline A because $-2\ ^{3}/_{7} > -3\ ^{1}/_{4}$.

C) Pipeline A is deeper than pipeline B because $-2\ ^{3}/_{7} > -3\ ^{1}/_{4}$.

D) Pipeline B is deeper than pipeline A because $-2\ ^{3}/_{7} < -3\ ^{1}/_{4}$.

Question Id: 50073, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""