

Randomized, Structured, Auto-graded Homework: Design Philosophy and Engineering Examples

Ms. Efthymia Kazakou, zyBooks, A Wiley Brand

Efthymia Kazakou is Sr. Assessments manager at zyBooks, a startup spun-off from UC Riverside and acquired by Wiley. zyBooks develops interactive, web-native learning materials for STEM courses. Efthymia oversees the development and maintenance of all zyBooks content resources used for assessment purposes.

Dr. Alex Daniel Edgcomb, zyBooks, A Wiley Brand

Alex Edgcomb is Sr. Software Engineer at zyBooks, a startup spun-off from UC Riverside and acquired by Wiley. zyBooks develops interactive, web-native learning materials for STEM courses. Alex actively studies and publishes the efficacy of web-native learning materials on student outcomes.

Dr. Yamuna Rajasekhar, zyBooks, A Wiley Brand

Yamuna Rajasekhar received her Ph.D. in Electrical Engineering from the UNC Charlotte. She served as a faculty member at Miami University where her research was focused on assistive technology, embedded systems, and engineering education. She is currently a Senior Content Developer at zyBooks, a startup that develops highly-interactive, web-native textbooks for a variety of STEM disciplines.

Prof. Roman Lysecky, University of Arizona; zyBooks, A Wiley Brand

Roman Lysecky is VP of Content at zyBooks, A Wiley Brand and a Professor of Electrical and Computer Engineering at the University of Arizona. He received his Ph.D. in Computer Science from the University of California, Riverside in 2005. His research focuses on embedded systems, cybersecurity, and STEM education. He has authored more than 100 research publications, received nine Best Paper Awards, is an inventor on multiple patents, and received multiple awards for Excellence at the Student Interface.

Prof. Frank Vahid, University of California, Riverside

Frank Vahid is a Professor of Computer Science and Engineering at the Univ. of California, Riverside. His research interests include embedded systems design, and engineering education. He is a co-founder of zyBooks.com.

Randomized, Structured, Auto-graded Homeworks: Design Philosophy and Engineering Examples

Abstract

Engineering homeworks encourage students to practice skills and apply concepts. Such homeworks are critical to a student's learning of course content and performance on high-stakes exams. Research has examined approaches to improve effectiveness of homeworks, including auto-grading for faster feedback and adaptivity to personalize a student's learning. Over the last 8 years, we have developed a homework activity framework that has been applied to multiple engineering and math disciplines with wide-spread adoptions: 600,000 students across 800 universities have submitted 90 million times. Our homework activities are integrated into web-based interactive textbooks. Such a homework activity is a sequence of progressively more difficult levels. A student must complete the first level's question to move on to the second level's question, and so on. Each level contains numerous same-difficulty questions, one of which is randomly selected when the student arrives at a level. A student's submission is auto-graded, and the student receives specific and immediate feedback to the given question and their submission. If the student answered incorrectly, then the student can try again on a new randomly-generated question of the same difficulty. Our homework activity philosophy is: (1) randomized -- each question is randomly generated to enable students plenty of practice and enable instructors to reuse the activity for an exam, (2) structured -- an activity is a sequence of incrementally harder questions so a student can demonstrate mastery, (3) auto-graded -- a student's submission is immediately assessed and the student is provided relevant feedback. This paper describes our homework activity philosophy, including pedagogical considerations made in designing such activities, many examples across different subjects, and reasons for implementing such a homework activity. Student submission data shows that on average across all the subjects discussed in this paper, an average of 98.4% of students were able to successfully complete an attempted level.

1. Introduction

Homework is a crucial aspect of an engineering course. Homework not only provides the students with the ability to practice, but also serves as a tool for instructors to assess their students' understanding of concepts taught in class. Traditionally, engineering homework is assigned by the instructor, and students complete the work, on paper or in a word processor, that the instructor then grades manually. Manual grading is time consuming for the instructor, can lead to human errors in the grading, adds delay until students receive feedback, and is one-direction communication. Also, the number of questions in the homework tends to be limited, so a student has a limited amount of practice. Further, the limited number of questions may increase the scope for academic dishonesty.

Various approaches have attempted to address the issues with traditional engineering homework. One approach is to auto-grade homework, of which numerous commercial and university solutions exist [1] [2]. Another approach adds auto-grading and randomly-generating homework questions [3]. Such approaches tend to be limited to numerical questions and do not support adaptivity. A third approach is for the homework to adapt to each student's ability and provide additional questions to help the student approach mastery [4] [5].

This paper presents an online homework activity framework that provides immediate, meaningful feedback to the students. The feedback describes what is wrong with the students' answers, and offers a step-by-step solution to the question presented. Each homework consists of a sequence of levels that have questions of increasing difficulty. Within each level, the homework activity uses randomization to enable students to be presented with a new question upon entering the wrong answer. In addition to this, the randomizations provide the students with the capability to practice more questions independently, especially in preparation for midterms or final exams.

This style of homework activity was developed at zyBooks in their online web-based interactive textbooks, which mainly target lower-division engineering and computer science courses [6]. In earlier papers, student usage of this homework activity framework was analyzed for digital design [7] and circuits [8] courses. This paper defines the homework activity philosophy and provides numerous examples across engineering.

2. Definition and philosophy

This section describes the common features of randomized, structured, auto-graded homework activities. This section includes an overview and philosophy description. The subsequent section gives examples from many engineering disciplines.

2.1 Overview

Our style of a homework activity consists of a series of auto-generated, randomized questions, each progressively more difficult. Students must correctly answer a question at each level before proceeding to the next higher level. For students, clicking on a level beyond the next uncompleted level yields an error message (instructors may jump to any level, however). Each level typically builds on earlier levels, so earlier levels must be completed first.

As shown in Figure 1, a homework activity contains:

- (a). A title describing the activity at a high level.

- (b). An area displaying the questions of the current level and fields for the student to answer.
- (c). A bar showing each level of the activity. Blue, filled-in levels are the completed levels, and the grayed out levels are the incomplete levels.
- (d). A “Check” button validating students' answers when pressed.
- (e). A “Next” button proceeding to the next higher level once the current level is successfully completed. If the answer is incorrect, clicking “Next” provides a new question of similar difficulty for the current level.
- (f). An explanation for the given answer. The green checkmark indicates the answer is correct.
- (g). Completion of the activity, viewed by filled-in icons next to the activity.

Figure 1. The components of a homework activity are numbered as follows: (a) title, (b) question area, (c) levels, (d) "Check" button, (e) “Next” level button, (f) explanation [in this case, for a correct answer], (g) progression bar.

The screenshot shows a homework activity interface. At the top, a header bar contains the text "HOMEWORK ACTIVITY" and "1.7.1: Adding and multiplying fractions." (a). Below the header, a question area (b) contains the instruction "Type answer in simplest (reduced) form." and a "Jump to level 1" link. The math problem $4/16 + 3/16 = 7/16$ is shown with the answer $7/16$ entered in a text box. Below the question area, a progression bar (c) shows four levels: level 1 is filled blue, while levels 2, 3, and 4 are grayed out. To the right of the question area is a vertical progression bar (g) with four icons: the first is a blue checkmark, and the others are empty boxes. Below the progression bar are two buttons: "Check" (d) and "Next" (e). At the bottom, an explanation box (f) shows a green checkmark, the text "Expected: 7/16", and the explanation "The bottoms are the same, so just add the tops: $4 + 3 = 7$, so $4/16 + 3/16 = 7/16$."

As shown in Figure 2, when an answer is incorrect, another two views appear:

- (a). A red box indicating a wrong answer.
- (b). A red X followed by an explanation when the answer is incorrect.

Figure 2. Homework activity when an answer is incorrect: (a) wrong answer, (b) explanation of wrong answer.

Type answer in simplest (reduced) form.

Jump to level 1

$3/8 + 1/4 =$ (a)

1 2 3 4

Check Next

(b) ✘ Expected: 5/8
 Expand 1/4 first: $1/4 \times 2/2 = 2/8$. Now that bottoms are same, tops can be added: $3/8 + 2/8 = 5/8$.

The screenshot shows a digital homework interface. At the top, it says 'Type answer in simplest (reduced) form.' Below that is a 'Jump to level 1' button. The main part of the screen displays the math problem $3/8 + 1/4 =$ followed by a text input field containing the incorrect answer '4/8', which is highlighted with a red border and labeled '(a)'. Below the input field is a progress bar with four segments labeled 1, 2, 3, and 4. Segment 1 is blue, segment 2 is grey with a blue border, and segments 3 and 4 are light grey. Below the progress bar are two buttons: 'Check' (orange) and 'Next' (orange with a blue border). Below the 'Check' button is a feedback box labeled '(b)' containing a red 'X' icon and the text: 'Expected: 5/8. Expand 1/4 first: $1/4 \times 2/2 = 2/8$. Now that bottoms are same, tops can be added: $3/8 + 2/8 = 5/8$.' On the right side of the interface, there is a vertical sidebar with a blue checkmark icon at the top, followed by four empty square boxes labeled 1, 2, 3, and 4.

2.2 Philosophy

This section describes the core homework activity philosophy and the enhancements such activities bring in modern, interactive learning.

One common feature in modern educational material is adaptivity. Adaptive learning provides personalized learning by delivering customized learning paths to keep students engaged and learning. The pace of learning is adjusted to each individual student using a variety of customized resources and activities to address the student's needs. Our homework activities are designed in such a way that each level covers a different subset of concepts and each successive level's subset is harder than the previous level. A student can continue retrying a level until a correct answer is reached (including generating new questions). Such an incremental approach represents "structured adaptivity", teaching specific concepts in an incremental manner to help students progress, while still enabling a less-prepared student many opportunities to practice. An explanation provides the student feedback, guides the student through the level, and adapts to the given question and answer provided.

As more classes become virtual and instructors need to cope with larger groups of learners more efficiently, auto-grading and self-assessment as a result, have become very important. Self-assessment promotes students' skills of reflective practice and self-monitoring, and increases students' motivation and confidence. This homework activity style encourages self-assessment especially with the use of activities where students receive immediate feedback on each question. Even when a student completes a question correctly, the student still benefits

from receiving feedback as the student's approach to solving the question may have been different, or the student may have been uncertain about her/his answer.

The effort has been to design homework activities with as many randomized questions as possible at each level to provide students with many examples to practice and learn from. And, of course, a side-benefit may be that students have a harder time sharing answers. Our homework activities support three different forms of randomization:

- **Meaningful:** Each meaningful permutation is a substantially different question and requires a significantly different solution. A meaningful randomization is introduced to avoid giving the same student the same meaningful question twice in a row. This also makes the activities highly-useful for exam and homework purposes.
- **Cosmetic:** Extends a meaningful permutation to have more cosmetic variations to mitigate quick cheating via look-up. An example from circuits is to randomize the amount of ohms of a resistor, whereas a meaningful randomization might be to change the placement of the resistor in the circuit.
- **Test-case:** Some disciplines' assessment is more open-ended, so a student's answer may be compared against a set of test cases or properties. Such an assessment has a unique problem: A student solving to satisfy the test cases, rather than the question statement. So, along with a core set of test cases, a randomly-generated additional set of test cases may be included. That random generation is done for each submission to an activity to reduce the effectiveness of just solving for the test cases.

Examples of different activities with all three types of randomization are shown in the next section.

2.3 Uses of randomized, structured, and auto-graded homework activities

The most common use of our homework activities is the same as for traditional homework activities: As an assignment. Our homework activities enable students to practice concepts and skills via a sequence of levels. Each level covers a different set of concepts, thus, completing an entire homework activity demonstrates a student's mastery.

When our homework activities are assigned by instructors, students must eventually successfully answer each level in order to receive points, and thus such activities serve as a lightweight assessment for instructors as well.

Another common use case is a student re-working through such activities while preparing for an exam. Since the activities are randomized, the student has ample opportunity to hone skills and reinforce understanding.

Another use case is an instructor assigning (or even re-assigning) a set of homework activities for an exam. This use case has interesting motivational implications for students, especially when students are informed that the activities will be the exam. Such an approach is analogous to making all past exams available to students, then re-writing a new exam for the current course. This approach works well due to the high number of question permutations in each activity.

3. Examples

This section includes detailed examples of the homework activity philosophy described in the previous section. The examples are from a variety of engineering and engineering-adjacent disciplines.

3.1 Programming in C++

Programming in C++ includes programming foundations and C++ basics, including branches, loops, arrays and vectors, pointers, functions, classes, inheritance, and exceptions. We define two types of homework activities: Code analysis and code writing. Both types assess students' mastery in programming by introducing code snippets that students have to either read and identify the code's output, or write to complete a given code snippet.

3.1.1 Code analysis

In code analysis activities, students are asked to read one code snippet per level, an optional input provided in a separate input text box (that can not be modified), and type the program's output, as shown in Figure 3. Students have to read through the code, follow the correct branches and calculate the correct answer.

The student's answer can be correct, incorrect, or nearly correct when the answer differs with the expected only in whitespace. Certain whitespace characters, such as a newline or tab, that are in the student's output but are not in the expected output, will be shown using special arrow symbols. In the later case, the tool allows the student to edit their code in order to modify the whitespace. On the one hand, whitespaces can be easily forgotten or accidentally added, so students should not fail an entire level when their answer is correct. Some of the questions are tricky enough so students showing mastery in reading a complex piece of code should be rewarded. On the other hand, their answer should not be immediately accepted as they may make the same mistake again if they don't correct it themselves, such as shown in Figure 3.

Figure 3: Code analysis question with a for loop and a break statement where the student's answer is nearly correct. Student asked to make a small update to pass the level.

Type the program's output

```
#include <iostream>
using namespace std;

int main() {
    int stop;
    int result;
    int n;

    cin >> stop;
    result = 0;

    for (n = 0; n < 10; ++n) {
        result += n * 2;
        if (result > stop) {
            cout << "n=" << n;
            cout << endl;
            break;
        }
        cout << result << endl;
    }

    return 0;
}
```

Input
9

Output
0
2
6
n=3

1 2 3 4 5

Check Next

✘ Output is nearly correct. Whitespace differs.

Yours
0
2
6
n=3

Expected
0
2
6
n=3

Create your missing newline by pressing Enter on your keyboard.

If the answer is incorrect, an explanation is shown that traces the code and indicates the correct answer. As in other activities, clicking "Next" generates a new question of the same difficulty for that level. If the answer is correct, a green checkmark appears, as shown in Figure 4.

Figure 4: Same code analysis question as Figure 3, except the student's answer is correct.

Type the program's output

```
#include <iostream>
using namespace std;

int main() {
    int stop;
    int result;
    int n;

    cin >> stop;
    result = 0;

    for (n = 0; n < 10; ++n) {
        result += n * 2;
        if (result > stop) {
            cout << "n=" << n;
            cout << endl;
            break;
        }
        cout << result << endl;
    }

    return 0;
}
```

Input
9

Output
0
2
6
n=3

1 2 3 4 5

Check Next

✓ When result > 9, a break statement executes. The rest of the loop body is skipped, and the loop exits.

Yours
0
2
6
n=3

Expected
0
2
6
n=3

3.1.2 Code writing

In code writing activities, students are shown an incomplete code snippet and are asked to type in their own code as the instructions indicate, as shown in Figure 5. Students should replace `/* Your solution goes here */` (or `""Your solution goes here""`) with their code while the surrounding sample program's code is NOT editable. The instructions always show one or more examples of the expected output/result depending on the different cases examined so that the question is never ambiguous to the student. As with code analysis activities, when some concepts are not directly covered in that section, hints are provided to help the student focus on the core concepts.

Figure 5: Outline of a code writing activity asking students to complete the code given in order to make the program return a specific result.

Define a function `FilterStr()` that takes a string parameter and returns "Yes" if the character at index 3 in the string parameter is lowercase. Otherwise, the function returns "No".

Ex: `FilterStr("SANDWICH")` returns

No

Recall `islower()` checks if the character passed is lowercase. Ex: `islower('a')` returns a non-zero value. `islower('A')` returns 0.

`string's at()` returns a character at the specified position in the string. Ex: `myString.at(3)`

```
1 #include <iostream>
2 #include <string>
3 #include <cctype>
4 using namespace std;
5
6 /* Your code goes here */
7
8 int main() {
9     string input;
10    string output;
11
12    getline(cin, input);
13    output = FilterStr(input);
14    cout << output << endl;
15
16    return 0;
17 }
```



Once an answer is submitted, the tool compiles the code. If a compilation error occurs, a descriptive message indicating the error is shown, as in Figure 6.

Figure 6: Compilation error of student's code in a code writing activity.

```
✖ Failed to compile
main.cpp: In function 'std::string FilterStr(std::string)':
main.cpp:14:12: error: a function-definition is not allowed here before '{' token
 14 | int main() {
    |           ^
main.cpp:23:1: error: expected '}' at end of input
 23 | }
    | ^
main.cpp:6:28: note: to match this '{'
   6 | string FilterStr(string s) {
     |                             ^
```

Note: Although the reported line number is in the uneditable part of the code, the error actually exists in your code. Tools often don't recognize the problem until reaching a later line.

If the code is successfully compiled, a set of randomized test cases are run against the student's code. As shown in Figure 7, if all tests pass, a green checkmark appears with the message "All tests passed", and the student can proceed to the next level. If any of the test cases did not pass, the expected output (or returned value) is shown along with the actual output produced by the student's code.

Figure 7: Example of a correctly-answered code writing activity, where all the test cases pass.

✓ 1: Compare output ^

Input

Your output

✓ 2: Compare output ^

Input

Your output

✓ 3: Compare output ^

Input

Your output

✓ 4: Compare output ^

Input

Your output

✓ 5: Compare output ^

Input

Your output

3.1.3 Common patterns between code analysis and code writing activities

Overall, a few common patterns are followed in the design of both code analysis and code writing activities:

- Randomization is rich, so that each permutation requires the student to read, interpret, or think about the part of the level's code that covers the new point made in the corresponding section.
- Randomization values are chosen to be distinct. As a result, the same value doesn't appear in the same question set, so in the explanations each value represents a different metric.
- Some good candidates for randomization are: different kinds of values used in the same level (e.g. a choice between a list of integers and a list of real numbers), strings of different lengths, order of print statements, comparison and arithmetic operators, function argument values and many more.
- All points made in the section should be covered across the activity's levels and each level should focus on one point.
- Functions and variables should not reveal their purpose so that students can't guess the output. For example, "result" is used rather than "total", "compute()" is used rather than "sum_of_squares()". Less committal names open up further opportunities for generalizing

the computation and data. They also facilitate randomization, so that, for example, the randomization doesn't need to change "total" to "product".

- Code comments are often used to give students a hint, particularly regarding a concept not directly covered in that section, thereby helping the student focus on the core concepts taught in that section.
- Activities are written in such a way that reduces the amount of typing students need to do. For example, instead of asking students to type "Sorry, your password was invalid", an activity asks students to type "Invalid".

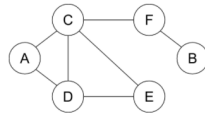
3.2 Data structures

Data structures include sorting, runtime complexity, lists, stacks, queues, hash tables, trees, graphs, and more. This section introduces two different styles to give a sense of the types of homework activities we have developed.

3.2.1 Graphs

The first style of question comes from algorithms involving graphs and trees. Such questions can be demanding as students have to traverse one or more graphs multiple times and answer a set of complex questions. If activities used that approach, getting an answer wrong would mean that the student would be shown a different graph and would have to perform the same calculations from scratch. For this reason, other question types are considered. As an example, Figure 8 shows a breadth-first search traversal question presenting a list of possible traversals and asking students to select the valid ones. This approach enables the students' critical thinking and helps them identify the key aspects of what makes a traversal valid.

Figure 8: Graphs question wherein a student applies the Breadth-first search algorithm and identifies all valid traversals.



Which of the following are valid breadth-first search traversals when C is the starting vertex?

- (1) C, E, D, F, A, B
- (2) C, D, E, F, A, B
- (3) C, B, A, D, E, F
- (4) C, A, D, E, F, B
- (5) C, F, A, E, D, B

1 2 3 4

Check Next

Expected:

- (1) C, E, D, F, A, B
- (2) C, D, E, F, A, B
- (4) C, A, D, E, F, B
- (5) C, F, A, E, D, B

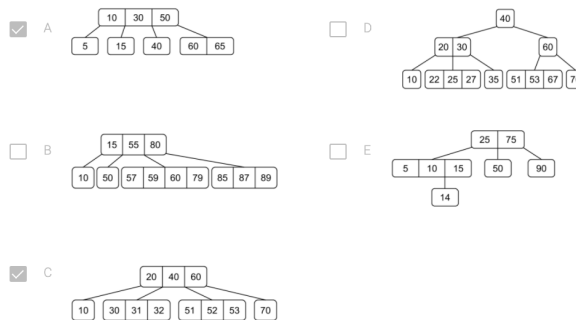
BFS first visits the starting vertex C.
 Vertices with a distance of 1 are visited next, so A, D, E, F in any order.
 Vertices with a distance of 2 are visited next, so B.

Invalid traversal:
 (3) Assuming C is visited, BFS should continue the path to vertex A, D, E, or F.

Figure 9 shows a similar example with 2-3-4 trees, where students are asked to select the valid ones. The question's explanation specifies the rules that were violated for each invalid graph. Each invalid tree violates one specific rule and thus shows the importance of each rule.

Figure 9: Trees question wherein a student identifies all valid 2-3-4 trees.

Select all valid 2-3-4 trees.



1 2 3 4

Check Next

Expected: A, C

Tree A: Valid. No 2-3-4 tree rules are violated.
 Tree B: Invalid. Violates the 2-3-4 tree requirement that all nodes must have 1, 2, or 3 keys: the root's middle child has 4 keys.
 Tree C: Valid. No 2-3-4 tree rules are violated.
 Tree D: Invalid. Violates the 2-3-4 tree ordering requirement: 67 is in the left child subtree of (60), but should be in the middle child subtree.
 Tree E: Invalid. Violates the 2-3-4 tree requirement that all leaves must be at the same level: leaf nodes (50) and (90) are on level 1, but leaf node (14) is on level 2.

Homework activity authors produce graphs used across different levels with different nodes, so that all possible cases are covered and randomizations are meaningful enough without being too repetitive.

3.2.2 Search and sort

The second style of question comes from searching and sorting algorithms. When introducing a complex algorithm, each level focuses on assessing one step of the algorithm. This way, students are shown how each step of the algorithm is executed and what it's responsible for. Figure 10 as an example, shows one step of the Quicksort algorithm where a list of numbers is to be partitioned into a low partition with values less than or equal to the pivot and a high partition with values greater than or equal to the pivot. Once all levels are completed, instructors know that the student has successfully grasped all aspects of the algorithm.

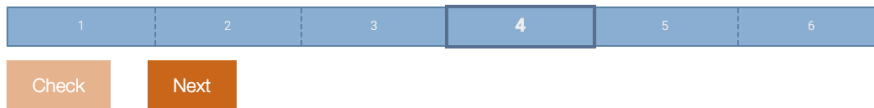
Figure 10: Example of a homework activity asking students to perform Quicksort's partitioning step given a list of numbers and a pivot.

Given numbers = (74, 91, 23, 59, 32, 67), pivot = 59

What is the low partition after the partitioning algorithm is completed?

(comma between values)

What is the high partition after the partitioning algorithm is completed?



✓ Expected:
low: 32, 59, 23
high: 91, 74, 67

Since 74 is greater than the pivot 59, and 32 is less than the pivot, the partitioning algorithm swaps 74 and 32 to place 32 in the low partition and 74 in the high partition.
numbers is now: (32, 91, 23, 59, 74, 67)

Since 91 is greater than the pivot and 59 is equal to the pivot, the partitioning algorithm swaps 91 and 59 to place 59 in the low partition and 91 in the high partition.
numbers is now: (32, 59, 23, 91, 74, 67)

Thus, the elements in the low partition are less than or equal to the pivot 59, so (32, 59, 23), and the elements in the high partition are greater than or equal to the pivot 59, so (91, 74, 67).

When less complex algorithms are introduced, such as quadratic search shown in Figure 11, each level may assess students on applying the same algorithm in progressively harder examples of the same data structure. When the data structure used consists of different states such as

Empty-since-start, Empty-after-removal, and Occupied for hash tables, great attention is given to the data structure's layout to make each state distinct.

Figure 11: Example of a homework activity with a quadratic search question for hash tables.

valsTable: 0
1
2 90
3
4
5
6
7 18
8 84
9 97
10

Empty-since-start
 Empty-after-removal
 Occupied

Hash table valsTable uses quadratic probing, a hash function of key % 11, c1 = 1, and c2 = 1.

What is the specific sequence of buckets probed by HashSearch(valsTable, 84)?

(commas between values)

1 2 3 4

Check Next

✓ Expected: 7, 9, 2, 8

$((84 \% 11) + 0 + 0 * 0) \% 11 = 7$. Bucket 7 is occupied with 18, so search continues.
 $((84 \% 11) + 1 + 1 * 1) \% 11 = 9$. Bucket 9 is occupied with 97, so search continues.
 $((84 \% 11) + 2 + 2 * 2) \% 11 = 2$. Bucket 2 is occupied with 90, so search continues.
 $((84 \% 11) + 3 + 3 * 3) \% 11 = 8$. 84 is in bucket 8, so 84 is found.

Finally, when it comes to explaining such algorithms, as shown in both Figures J and K, explanations use a step by step approach to explain each part of the algorithm, so students can identify any mistake easily or verify their end to end solution.

3.3 Discrete mathematics

Discrete mathematics includes a diverse set of math concepts, including propositional logic, set theory, boolean algebra, induction, recursion, and discrete probability. Each concept requires a unique question style to best assess a student's understanding of that concept and provide students meaningful feedback. So, auto-graded homework questions in discrete mathematics have many styles. This section includes three diverse styles to give a sense of that variety.

3.3.1 Propositional logic

The first style comes from propositional logic, specifically applying laws to transform an initial proposition into a goal proposition, as shown in Figure 10. Initially, a student is given a randomly-generated proposition and the simplified version of that proposition as the goal. The student selects a law from the laws on the right-hand side, then the student selects the respective parts of the proposition on the left-hand side on which to apply that law. If the law cannot be applied, then the tool automatically informs the student, explaining why. Otherwise, the resulting proposition is shown. If the resulting proposition is the goal proposition, then the tool

automatically marks the question as correct. Otherwise, the student can select the next law to apply and continue work. This question style is used for both reducing and expanding a proposition (and sometimes a combination of both as Figure 12 shows).

Figure 12: Propositional logic question wherein a student applies 1 law at a time to transform an initial proposition into a goal proposition.

Simplify $\neg(q \vee (w \wedge \neg q))$ to $\neg q \wedge \neg w$

See feedback below.

$\neg(q \vee (w \wedge \neg q))$
 $\neg q \wedge \neg(w \wedge \neg q)$
 $\neg q \wedge (\neg w \vee \neg \neg q)$
 $\neg q \wedge (\neg w \vee q)$
 $(\neg q \wedge \neg w) \vee (\neg q \wedge q)$
 $(\neg q \wedge \neg w) \vee (q \wedge \neg q)$
 $(\neg q \wedge \neg w) \vee F$
 $\neg q \wedge \neg w$

Laws	
Distributive	
$(a \wedge b) \vee (a \wedge c)$	$= a \wedge (b \vee c)$
$(a \vee b) \wedge (a \vee c)$	$= a \vee (b \wedge c)$
Commutative	
$a \vee b$	$= b \vee a$
$a \wedge b$	$= b \wedge a$
De Morgan's	
$\neg(a \wedge b)$	$= \neg a \vee \neg b$
$\neg(a \vee b)$	$= \neg a \wedge \neg b$
Conditional	
$a \rightarrow b$	$= \neg a \vee b$
$a \leftrightarrow b$	$= (a \rightarrow b) \wedge (b \rightarrow a)$
Complement	
$a \vee \neg a$	$= T$
$a \wedge \neg a$	$= F$
$\neg T$	$= F$
$\neg F$	$= T$
Identity	
$a \wedge T$	$= a$
$a \vee F$	$= a$
Double negation	
$\neg \neg a$	$= a$



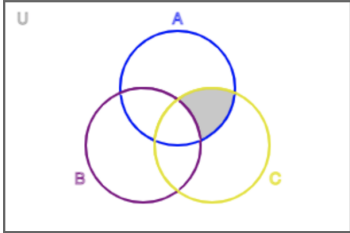
✔ Correct

3.3.2 Set theory

A second style of question comes from set theory, specifically the assessment of set operations and notation, as shown in Figure 13. A student is asked to select the region(s) of a Venn diagram for a randomly-generated combination of set operations. Each level has specific combinations of operations that are assessed. The student selects the regions then clicks Check. If correct, then the student is told so. Otherwise, another Venn diagram with the correct answer is shown to the right of the student's Venn diagram, and then the student is given another randomly-generated question.

Figure 13: Set theory question wherein a student selects the region(s) of a Venn diagram defined by a given combination of set operations.

Select the region(s) of:
 $(A - B) - \bar{C}$



U

1 2 3 4 5

Check Next Done. Click any level to practice more. Completion is preserved.

✓ Correct.

3.3.3 Recurrence relations

The third style of question comes from recurrence relations, specifically the computation of a term in a sequence. The student is given a randomly-generated definition of a sequence and asked to compute the value of a randomly-generated term, as shown in Figure 14. Each level has particular variations of a sequence definition from which a question is generated. A student enters the value in an input field that only accepts numbers, and then the student clicks Check. Regardless of correctness, the student is given an explanation showing how to compute each term until the goal term.

Figure 14: Recurrence relation question wherein a student computes a term's value from a given definition of a sequence. The explanation steps through each term in the sequence until the goal term's value is found.

Given the definition of a sequence f_n whose domain is the set of all non-negative integers:

$$f_0 = 5$$

$$f_n = n^2 \cdot f_{n-1} \quad \text{for } n \geq 1$$

$f_3 =$

1 2 3 4

Check Next

✓ Expected: 180

$$f_0 = 5$$

$$f_1 = 1^2 \cdot f_0 = 5$$

$$f_2 = 2^2 \cdot f_1 = 20$$

$$f_3 = 3^2 \cdot f_2 = 180$$

3.4 Digital design

Digital design introduces the basic concepts required to design and implement RTL logic. The auto-graded homework activities cover a variety of concepts including implementing circuits with gates, boolean algebra concepts, K-maps, Finite State Machines, datapath components, and more. The style of each homework activity depends on the topic, and the randomizations are tailored to best teach each concept effectively.

3.4.1 K-maps

The first style of question presented assesses K-map simplification as shown in Figure 15. Initially, a student is given a randomly generated 3-variable K-map to simplify. The student is asked to add circles to cover all the 1s in the K-map. To achieve that, the student clicks to select one or two 1s and then clicks the “Add circle” button. When the student is done adding circles, the student clicks the Check button to verify the answer. If the student tries to add an invalid circle (Ex: diagonal 1s), the activity offers a hint that states "Invalid circle. Valid circles have adjacent cells". If the student's answer contains the largest and fewest circles, the student can proceed to the next level. Once the student completes all the levels in an activity, the student can click any level to practice more.

Figure 15: Example of a 3-variable K-map simplification when the student is adding circles.

Add fewest and largest circles to cover all the 1s.

Jump to level 1

bc	00	01	11	10
a				
0	0	1	1	1
1	0	0	0	0

Add circle Undo

1 2 3 4 5 6 7

Check Next

Done. Click any level to practice more. Completion is preserved.

✔ Correct.

If the student's answer is wrong and does not contain the correct number of circles, the student is offered an explanation of what was wrong with the answer, as shown in Figure 16. The incorrect circles are shown in red, and the missing circles are shown in green. If the student is ready to try the level again, the student will be presented with a different question of similar difficulty, that is randomly generated.

Figure 16: Example of a 3-variable K-map simplification when student's answer is incorrect.

Add fewest and largest circles to cover all the 1s.

Jump to level 1

	bc	00	01	11	10
a	0	0	0	1	0
	1	0	0	1	1

✘ Use the fewest and largest possible circle(s) that contains only 1s.
 Missing circle(s) shown in light green.
 Incorrect circle(s) shown in red.

This particular question presents only one type of K-map homework activity. Other questions include 2-variable K-map simplification, writing the simplified terms for 3-variable K-maps, converting truth tables and equations to K-maps, as well as “don't cares”. Each activity has different questions, and each question has many meaningful randomizations, which is useful for a student to practice and build skill.

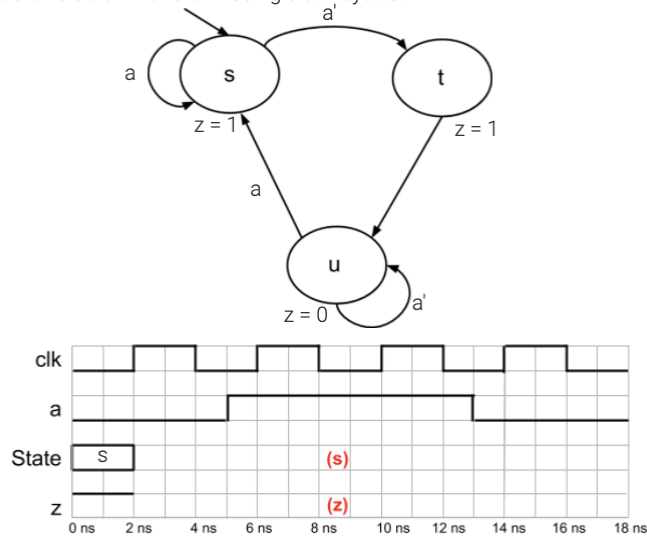
3.4.2 Finite state machines

Another major concept covered in any digital design course is Finite State Machines (FSMs). We have a variety of homework activities to test the different concepts of FSMs. The first question, shown in Figure 17, tests the student's ability to read an FSM and walk through the different states and transitions. The student is given an FSM and has to indicate the values of the output and the state of the FSM for the given input and clock cycle combinations. The question provides

a detailed explanation of the input value and the transitions taken at each clock cycle to arrive at the expected values of the state at each clock cycle. Using the value of the state, the output values are determined. This activity uses two types of randomizations: one on the FSM itself by changing the transitions to different states and the state outputs, and another one on the timing diagram by changing the value of the input itself.

Figure 17: Example of question capturing behavior as an FSM.

Indicate the value of z for the missing clock cycles.



(s) =

(z) =

1 | 2 | 3

Done. Click any level to practice more. Completion is preserved.

✓ Expected: (s) = tust, (z) = 1011

State is updated each clock rise:

Clock edge at 2 ns: a is 0, so the FSM transitions to state t.

Clock edge at 6 ns: The transition leaving t has an implicit condition of true. So the FSM transitions to state u, regardless of a's value.

Clock edge at 10 ns: a is 1, so the FSM transitions to state s.

Clock edge at 14 ns: a is 0, so the FSM transitions to state t.

Thus, (s) = tust.

z is 1 in state s, 1 in state t, and 0 in state u. So, (z) = 1011.

Another question that assesses the FSM concepts is one with an FSM simulator that enables the students to build an FSM as shown in Figure 18. The student is given a simple sequence generator FSM question to start with. Questions 2 and 4 ask questions with one and two outputs, respectively. Question 3 asks the student to maintain a one cycle pulse on the output given an input. Here, the student captures the behavior of the question by building an FSM. The student

can add states with actions for state outputs, and transitions with conditions for each transition. The student has the ability to simulate the FSM to verify the behavior before checking the result. The randomizations are on the pattern to be generated.

Figure 18: Example of building an FSM.

Generate the repeated pattern 0 1 1 on z

State/transition info

Initial state

Actions

Condition

1 2 3 4

Check Next

✓ Correct

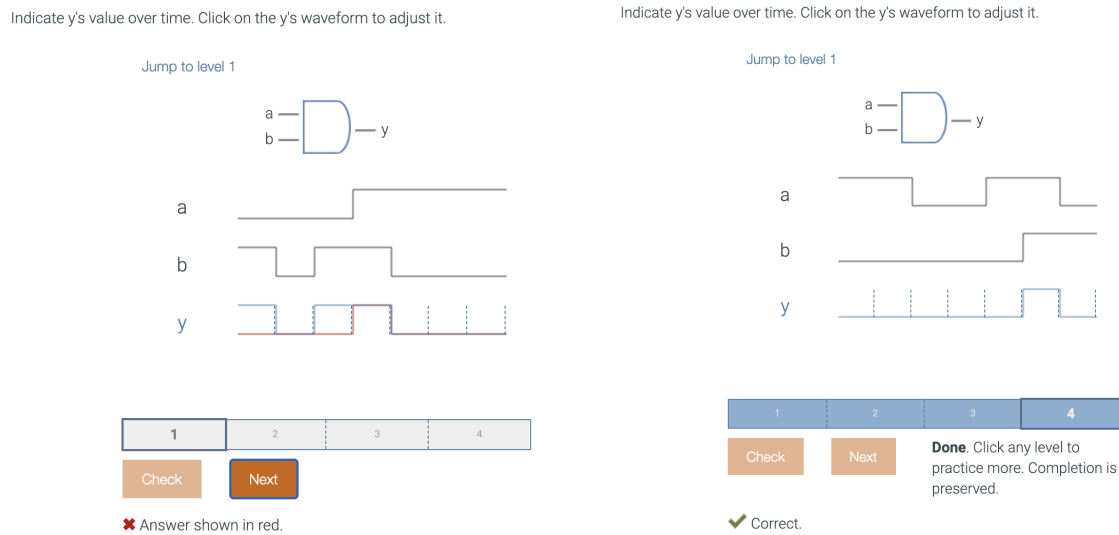
Expected z	0	1	1	0	1	1	0	1	1
Your z	0	1	1	0	1	1	0	1	1

Other homeworks include converting FSMs to truth tables, Mealy FSMs, controller clock frequency, and converting circuits to FSMs.

3.4.3 Timing diagram

The understanding of timing diagrams is crucial to building hardware circuits. While there are a few homework activities that teach and assess the understanding of this concept, Figure 19 shows one such timing diagram activity for gate output. The student is given a gate and the values of the gate inputs are presented as a timing diagram. This activity has four questions: the first two questions ask for the output of an AND gate, and the next two questions ask for the output of an OR gate. The student clicks on the output's given dotted lines to change the value of y from 0 to 1. If the entered answer is incorrect, the student is shown the correct answer, in comparison with the student's wrong answer. The student is then presented with another question in the same level but with different input values. Like all other homework activities presented in this paper, if the student enters the correct answer, the student moves on to the next level.

Figure 19: Example of a timing diagram of gate output.



Other homeworks include this timing diagram tool to teach different concepts. For example, the working of the SR latch, D flip-flops, and load registers.

3.5 Circuits

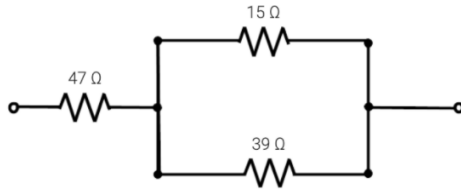
Circuits is a required course for all electrical and computer engineering students. Circuits homework activities cover basic electricity concepts like resistors, capacitors, and resistor networks using different laws. Then, slightly more advanced topics like network and time-domain analysis, op-amps, and frequency-domain analysis are also covered. For circuits, some activities focus more on randomizing the values of the circuit components to help students practice with the formulae and calculations.

3.5.1 Equivalent resistances for combinations of series and parallel resistors

This homework activity presents students with different circuits and asks the student to calculate the equivalent resistance. As shown in Figure 20, question 1 presents the student with a simple circuit that consists of three resistors, and question 5 presents the student with a complex circuit with 7 resistors in a series/parallel combination. Each question adds resistors to present increasingly complex circuits. If the student gets the answer incorrect, the student is presented with a detailed explanation of the formula and how the values are applied to arrive at the correct answer. This explanation is presented to the student even if the student gets the answer correct to reinforce the concepts. Each question consists of several different questions with the values of the resistors changed, to enable the student to do more practice.

Figure 20: Example of a circuits question calculating the equivalent resistance.

Calculate the equivalent resistance for the following circuit.

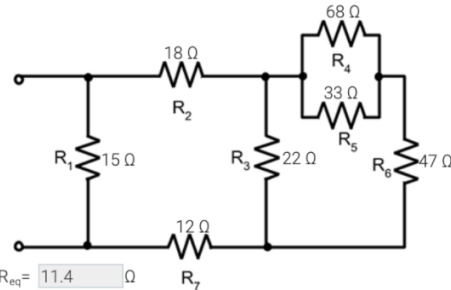


$R_{eq} =$ Ω
Round to three significant figures.



✓ Expected: $R_{eq} = 57.8\Omega$.
For this circuit, the 15Ω and 39Ω resistors are in parallel, and the combination is in series with the 47Ω resistor.
$$R_{eq} = 47 + \frac{15 \cdot 39}{15 + 39} = 57.8\Omega$$

Calculate the equivalent resistance for the following circuit.



$R_{eq} =$ Ω
Round to three significant figures.



Done. Click any level to practice more. Completion is preserved.

✓ Expected: $R_{eq} = 11.4\Omega$.
The 68Ω and 33Ω resistors are in parallel, and the combination is in series with the 47Ω resistor.
$$\text{combination1} = 47 + (68||33) = 47\Omega + \frac{68\Omega \cdot 33\Omega}{68\Omega + 33\Omega} = 69.2\Omega$$

The combination of resistors is in parallel to the 22Ω resistor.
$$\text{combination2} = 69.2||22 = \frac{69.2\Omega \cdot 22\Omega}{69.2\Omega + 22\Omega} = 16.7\Omega$$

The calculated value (16.7Ω) is in series with the 18Ω and 12Ω resistors.
$$\text{combination3} = 18\Omega + 16.7\Omega + 12\Omega = 46.7\Omega$$

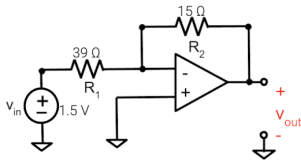
Finally, combination3 (46.7Ω) is in parallel with the 15Ω resistor.
$$R_{eq} = 46.7\Omega||15\Omega = \frac{46.7\Omega \cdot 15\Omega}{46.7\Omega + 15\Omega} = 11.4\Omega$$

3.5.2 Op-amps

This homework activity requires students to find the voltages for inverting op-amps. The activity has five questions, and each question covers a different concept. The first question, as shown in Figure 21, asks students to calculate V_{out} , the second question asks for V_{in} , and the third question asks for $-v_{in}$, the question level asks for R_2 , and the fifth question presents a conceptual question. The randomizations within each level are generated by changing the values of the resistors and voltages.

Figure 21: Example of circuits question finding voltages for inverting op-amps.

Calculate v_{out} for the following ideal inverting op-amp.



$v_{out} =$ V
Round to three significant figures.



✓ Expected: $v_{out} = -0.577V$.

For ideal op-amps, both the currents entering the inverting and non-inverting inputs of the op-amp are zero, and the voltage difference between the inverting and non-inverting inputs is zero. Thus,

$$v_{out} = -\frac{R_2}{R_1} \cdot v_{in} = -\frac{15}{39} \cdot 1.5 = -0.577V$$

A summing amplifier has a voltage output of $-2v_1 - 4v_2$. What would be the simplest way to invert the signs of each term?

- ✓ Choose the best answer
- Add an inverting amplifier between v_1 and the inverting terminal of the op-amp.
 - Add an inverting amplifier to the summing amplifier's output.
 - Add an inverting amplifier to the summing amplifier's input.



4. Data on student submissions to homework activities

This section presents student submission data on the randomized, structured, and auto-graded homeworks described in the previous section and gives a sense for whether students were successful at solving the levels. Some past research has deeper analysis on specific subjects, including digital design [7] and circuits [8].

The data presented is from the latest release and includes data of students who were awarded points for completing the levels and students who were not.

The metrics included per subject are:

- Number of homework activities: A count of the total number of homework activities in a subject.
- Number of levels: The total number of levels across all the homework activities in a subject.
- Number of submissions: The total number of student submissions for any homework activity level.
- Completion rate: A percentage approximating persistence, that shows how many students complete a level after attempting. This is computed as the (number of completed levels) / (number of attempted levels), wherein a completed level for a student has at least 1 correct submission and an attempted level for a student has at least 1 submission.

Table 1: Student submission data to homework activities across multiple engineering subjects.

Subject	Number of homework activities	Number of Levels	Number of submissions	Completion rate
Circuits	50	205	412,317	98.1%
Data Structures	44	169	2,636,916	97.9%
Digital Design	55	246	1,517,738	98.3%
Discrete Math	66	297	4,419,262	98.9%
Programming in C++	210	541	9,290,225	98.9%

As shown in Table 1, each subject has homework activities that range from a few tens of activities to over 200 activities for Programming in C++. Each homework activity has several levels and each of those levels have many submissions. Most students completed the level that they had attempted. Across all the subjects, an average of 98.4% of students that attempted a particular level ended up completing that level. This indicates that most students were able to successfully solve the level that was attempted. Each subject had hundreds of thousands to millions of submissions, so these activities have had a lot of use.

5. Authoring considerations

A traditional homework presents the same question to each student. The author of a traditional homework can quickly proofread and revise the question, while getting help from auto-spelling and auto-grammar tools. And then the author can simply email a colleague or assistant to quickly review. A highly-randomized homework activity adds complexity at each step. This section describes different techniques to handle each complexity, as well as present trade-offs between competing techniques.

One complexity is how to define questions. One technique is to explicitly author a number of questions then just randomly pick one question. While this technique does not scale well with the number of questions, some concepts, especially the more conceptual ones, do not offer many permutations. Another technique is to use structured randomization, such as randomizing an integer in the question to be picked from a list of integers, or similarly randomizing a word to be picked from a list of words. The list is the structured randomization. Structured randomization enables many questions to be concisely defined. Adding another permutation may be as simple as adding another item to the list. An activity may have multiple lists, such as one list for one number and another list for a word. The number of questions then becomes a product of the lengths of each list (e.g., 3 lists with 10 items each is $10 \times 10 \times 10 = 1,000$ questions). One challenge with structured randomization is when interdependencies between lists are introduced, such as

avoiding picking the same number from two lists like $a=[1, 2, 3]$ and $b=[2, 3, 4]$. A possible solution is to contrive each list to have elements not found in the other list, but that reduces the number of possible questions. Another solution is supporting another structure, called a collection. A first collection has $a=[1]$ and $b=[2, 3, 4]$, a second has $a=[2]$ and $b=[3, 4]$, and a third has $a=[3]$ and $b=[2, 4]$. The collection is in a list itself, so that one of the collections is randomly-selected, then one element from each list is selected. That avoids the loss of potential questions, but certainly adds complexity. A third technique to define questions is by the author writing code, such as Python or MATLAB, that generates a question. This technique is the most flexible, but requires code writing experience, introduces the possibility of crashing the randomization process, and makes it challenging to compute the number of possible questions.

Another complexity is how to review a randomized homework activity. One technique is to generate one question, proofread, and repeat. This manual technique does not scale, so a reasonable upper-bound on the number of questions to proofread needs to be set, such as a constant number (like 5 questions), a constant proportion (like 20% of all questions), or a likelihood that no questions have errors (like enough questions are proofread to be 90% confident that no other question has an error). Another technique is to include automation where possible, such as automatically generating many questions, running each question through a spell-checker, then report to the author if an error was found. A similar technique can be used for grammar checking and other forms of automatable error checking. Such error checking may again not be feasible when the number of question permutations is sufficiently large.

Another complexity introduced by highly-randomized homework activities is handling reports of a mistake in a question; namely, identifying how to reproduce the question and verify that the question was fixed. One technique is to log each question that was generated for each user, including how to generate the question, if possible.

6. Conclusions

This paper presented numerous examples of randomized, structured, and auto-graded homework activities across several engineering disciplines. 600,000 students across 800 universities have submitted 90 million times to these homework activities. The homework activities are structured into a sequence of incrementally-harder questions. Each question is randomly-generated from among other questions of a similar difficulty. Each question is automatically graded, and often a question-specific (and sometimes answer-specific) explanation is provided. Such homework activities share much in common with traditional homework approaches in terms of benefits, yet provide substantial advantages to students and instructors at the cost of being complex to author. This paper discussed the challenges of those complexities, as well as the trade-offs between techniques to mitigate each challenge. For instructors and authors, a key consideration is the balance between providing practice for students versus fatiguing the students. These homework

activities continue to evolve to support additional disciplines and to provide additional advantages to students, instructors, and authors.

References

- [1] Koller, D., & Ng, A. (2013, January). The online revolution: Education for everyone. In *Seminar Presentation at the Said Business School, Oxford University*. Retrieved from <http://www.youtube.com/watch>.
- [2] Mohammed, M. K. O. (2020, February). Teaching Formal Languages through Visualizations, Simulators, Auto-graded Exercises, and Programmed Instruction. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 1429-1429).
- [3] Basitere, M., & Ivala, E. N. (2017). An Evaluation of the Effectiveness of the use of Multimedia and Wiley Plus Web-Based Homework System in Enhancing Learning in The Chemical Engineering Extended Curriculum Program Physics Course. *Electronic Journal of e-Learning*, 15(2), pp156-173.
- [4] Hagerty, G., & Smith, S. (2005). Using the web-based interactive software ALEKS to enhance college algebra. *Mathematics & Computer Education*, 39(3).
- [5] Knewton. <https://www.knewton.com>. Visited: March 2021.
- [6] zyBooks. <https://www.zybooks.com/>. Accessed May. 2021.
- [7] Rajasekhar, Y., Edgcomb, A., Vahid, F. (2019, June). Student Usage of Digital Design Interactive Learning Tools in an Online Textbook. In ASEE Annual Conference and Exposition, Conference Proceedings, June, 2019.
- [8] Sambamurthy, N., Edgcomb, A., & Rajasekhar, Y. (2019, October). Student Usage of Interactive Learning Tools in an Online Linear Circuit Analysis Textbook. In 2019 IEEE Frontiers in Education Conference (FIE) (pp. 1-6). IEEE.