Analyzing Student Session Data in an eTextbook

Samnyeong Heo

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Application

Clifford A. Shaffer, Chair
Mohammed F. Farghally
Margaret Ellis

June 8, 2022
Blacksburg, Virginia

Keywords: Clickstream data, OpenDSA, LMS, Electronic Textbook

Analyzing Student Session Data in an eTextbook

Samnyeong Heo

ABSTRACT

As more students interact with online learning platforms and eTextbooks, they generate massive amounts of data. For example, the OpenDSA eTextbook system collects clickstream data as users interact with prose, visualizations, and interactive auto-graded exercises. Ideally, instructors and system developers can harness this information to create better instructional experiences. But in its raw event-level form, it is difficult for developers or instructors to understand student behaviors, or to make testable hypotheses about relationships between behavior and performance.

In this study, we describe our efforts to break raw event-level data first into sessions (a continuous series of work by a student) and then to meaningfully abstract the events into higher-level descriptions of that session. The goal of this abstraction is to help instructors and researchers gain insights into the students' learning behaviors. For example, we can distinguish when students read material and then attempt the associated exercise, versus going straight to the exercise and then hunting for the answers in the associated material. We first bundle events into related activities, such as the events associated with stepping through a given visualization, or with working a given exercise. Each such group of events defines a state. A state is a basic unit that characterizes the interaction log data, and there are multiple state types including reading prose, interacting with visual contents, and solving exercises. We harnessed the abstracted data to analyze studying behavior and compared it with course performance based on GPA. We analyzed data from the Fall 2020 and Spring 2021 sections of a senior-level Formal Languages course, and also from the Fall 2020 and

Spring 2021 sections of a data structures course.

Analyzing Student Session Data in an eTextbook

Samnyeong Heo

GENERAL AUDIENCE ABSTRACT

OpenDSA is an online learning platform used in multiple academic institutions including Virginia Tech's Computer Science courses. They use OpenDSA as the main instructional method and students in these courses generate massive amounts of clickstream data while interacting with the OpenDSA content. The system collects various events logs such as when students opened/closed a certain page, how long they stayed on the page, and how many times they clicked an interface element for visualizations and exercises. However, in its raw event-level form, it is difficult for instructors or developers to understand student behaviors, or to make testable hypotheses about relationships between behavior and performance. We describe our efforts to break raw event-level clickstreams into a session (continuous series of work by a student) and then to abstract the events into meaningful higher-level descriptions of students' behavior. We grouped raw events into related activities, such as the events associated with stepping through a given visualization, or working with a given exercise. We defined such a group of activities as a state, which is a basic unit that can characterize the interaction log data such as reading, slideshows, and exercises state. We harnessed the abstracted data to analyze students' studying behavior and compared it with their course performance based on their GPA. We analyzed data from two offerings of two CS courses at Virginia Tech to gain insights into students' learning behaviors.

Dedicated to Virginia Tech.

Acknowledgments

My sincere thanks to everyone who made this project to be possible. This work would not have been possible without the constant support, guidance, and assistance of many people.

I would first like to thank Professor Clifford A. Shaffer for his inspiration and guidance. The door to his office was always open whenever I ran into a trouble or had a question about my research or writing.

I would also like to thank Dr. Mohammed Farghally for his encouragement and steering me in the right direction whenever I faced a wall. I would like to thank Margaret Ellis for serving on my thesis committee and Dr. Mostafa Mohammed, who has supported me since I joined the OpenDSA research group. I am grateful to Jieun Chon, for the extraordinary guidance and support as a mentor who kept me motivated and confident.

Thanks to my friends and family for their unconditional love and support throughout the entire thesis process and every day.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Online educational resources are becoming increasingly sophisticated. In Computer Science in particular, there is widespread use of programming environments both for large and small programs (projects vs. exercises), interactive exercises, and eTextbooks. Such environments include automatic assessment of programming exercises [1], Web-CAT [2] for testing and grading for student projects, and teaching environments like Alice [3] and GreenFoot [4] The most important pedagogical impact of these systems is that they increase the level of student interaction over paper textbooks, lectures (live or video), and paper homework that have slow feedback cycles [5, 6]. Unlike traditional textbooks with static images, electronic textbooks could include videos, and instead of plain text, eTextbooks could include dynamic texts that can easily be updated to provide better presentations. Another important aspect is their ability to reduce the grading burden for instructors, as online activities are typically graded automatically. All of the students' progress and exercise results are automatically recorded and assessed through the platform.

1.1 Problem Statement

To the extent that the primary purpose of these online systems is to increase the level of student engagement with the material, the actual amount of engagement and the details of the type of engagement are likely to be important.

Fortunately, these systems by their very nature make it easy to collect detailed event-level data about how students interact with the system, including the time distribution of the interactions (such as a little, a lot, many small sessions, a few large sessions, etc.), and the behavior. (For example, do students read the material and then work the assessment, or do they go to the assessment and only look at the material as necessary to get credit?)

On the other hand, the basic unit of data collection is an event such as a button click, typing a character or unit of text, opening or leaving a page. A large class with a lot of system usage could easily generate in excess of a million events. How can instructors or researchers hope to make practical use of this firehose of data?

In this study, we describe our efforts to organize the event clickstream from the use of an eTextbook system, first into individual "sessions" of use, and then into higher levels of abstraction for the activity in the session.

1.2 Objectives

The primary aim of this thesis is to further investigate how students are interacting with the OpenDSA by exploring the continuous stream of session data. We attempt to achieve this goal by abstracting raw event stream data into a study session, and then trying to classify study behaviors that students generally follow while in a module.

The second aim of this study is to observe the behaviors we found from abstracted session data and see if there is any correlation between basic module use behavior patterns and students' overall performance in the class (e.g., their overall GPA). We attempt to achieve this goal by splitting students into two groups based on their overall GPA score and discovering patterns that are likely to happen in one of the groups. Such patterns that can be discovered

include students opening a module and spending some time to read prose before attempting to solve exercises associated with the reading materials, or skipping directly to the exercises in order to receive credit by completing an assignment regardless of whether they learned the material. This second approach we refer to as a credit-seeking behavior.

There could be multiple behaviors we can discover and details of these behaviors will be discussed in a later section of this thesis. However, the purpose of finding the correlation between behaviors and performance is to provide future guidance to instructors and developers on ways to make OpenDSA be a better educational tool.

Chapter 2

Background

This chapter introduces some background about the architecture of OpenDSA, the eTextbook that we used in our study, and how it stores information in the database, retrieves data needed for the study, and related work done previously on students log data. Understanding how OpenDSA is constructed is crucial before diving into data analysis. In addition, how OpenDSA gathers information from users to record activities and interactions within a module is a core part of understanding users' behavior.

2.1 OpenDSA

OpenDSA [7, 8] is an open-source electronic textbook system that is used in several courses in the Computer Science department at Virginia Tech and other institutions around the globe. It provides content on various computer science-related topics including Data Structures and Algorithms, Formal Languages, Finite Automata, and Programming Languages. It includes algorithm visualizations, interactive exercises such as small code writing problems, and proficiency exercises designed with goals such as leveraging technology to make the learning process easier, motivating, and entertaining, and to evaluate how well students learn [7]. OpenDSA textbooks are custom configurable, allowing instructors to easily customize course materials to fit the needs of various courses. It provides an environment where both learners and instructors can overcome obstacles such as visualization technology may

not be beneficial to learners and instructors may have difficulty finding good materials [9, 10]. OpenDSA helps learners practice skills and develop knowledge by providing visual aids and various exercises. Instructors can create custom electronic textbooks, selecting from comprehensive materials and exercises.

2.1.1 Structure of OpenDSA

Module

A fundamental unit of OpenDSA is the module. A module organizes content into a single web browser page that might contain prose, visualizations, graphics, code snippets, and various exercises. Each module resembles a specific topic or a section in a traditional textbook, and groups of modules form chapters as depicted in Figure 2.1.

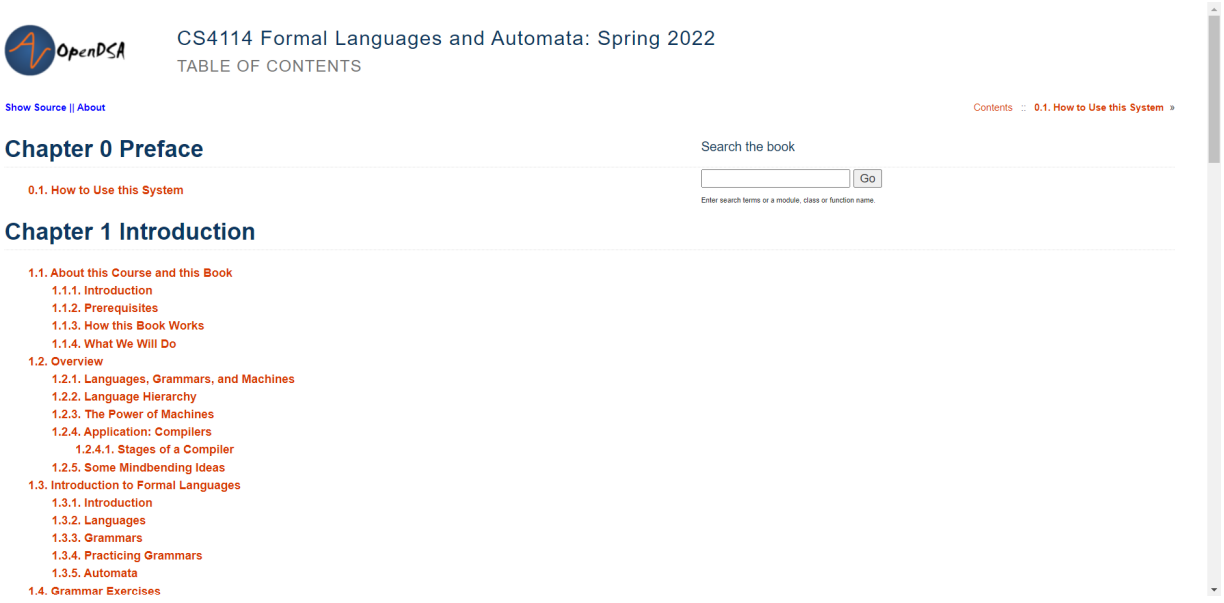

Figure 2.1: OpenDSA Table of Contents Page for a Formal Languages and Automata course

Proficiency Exercises

The courses we investigated, Formal Languages and Automata (FLA) and Data Structures and Algorithms (DSA), have many visualizations and exercises to promote increased engagement. The presentation of materials attempts to reduce the amount of text to cover a given topic since previous studies [11] found that the students tend to skip reading prose in eTextbooks, especially if it involves complex mathematical content. One such exercise type used in the FLA course is called a Proficiency Exercise (PE). These types of exercises were inspired by the TRAKLA2 system developed by Malmi et al. [12]. Students must devise a state machine, represented by a graph, that can correctly determine whether a given string is in a specified language. Student solutions are tested by applying multiple sets of test cases like unit tests for checking a program. Figure 2.2 shows an example of a proficiency exercise to create a Deterministic Finite Automaton (DFA). In the exercise, students are asked to develop a DFA for the problem statement with nodes and edges shown at the top of the figure, and their DFA is graded against the sample test cases as demonstrated at the bottom of the figure.

PIFrame

The customized slideshows used in the FLA course are called Programmed Instruction Framesets (PIFrame). Their goal is to enhance students' interaction and engagement (an example of a PIFrame shown in Figure 2.3). A PIFrame is a series of slides to help students see how different models and their respective algorithms work. Students must answer questions on most slides in order to advance to the next slide, ultimately to receive a credit for their grade by completing a frameset [11, 13].

Figure 2.3 shows a slide from a demonstration of how to use the Pumping Lemma to prove

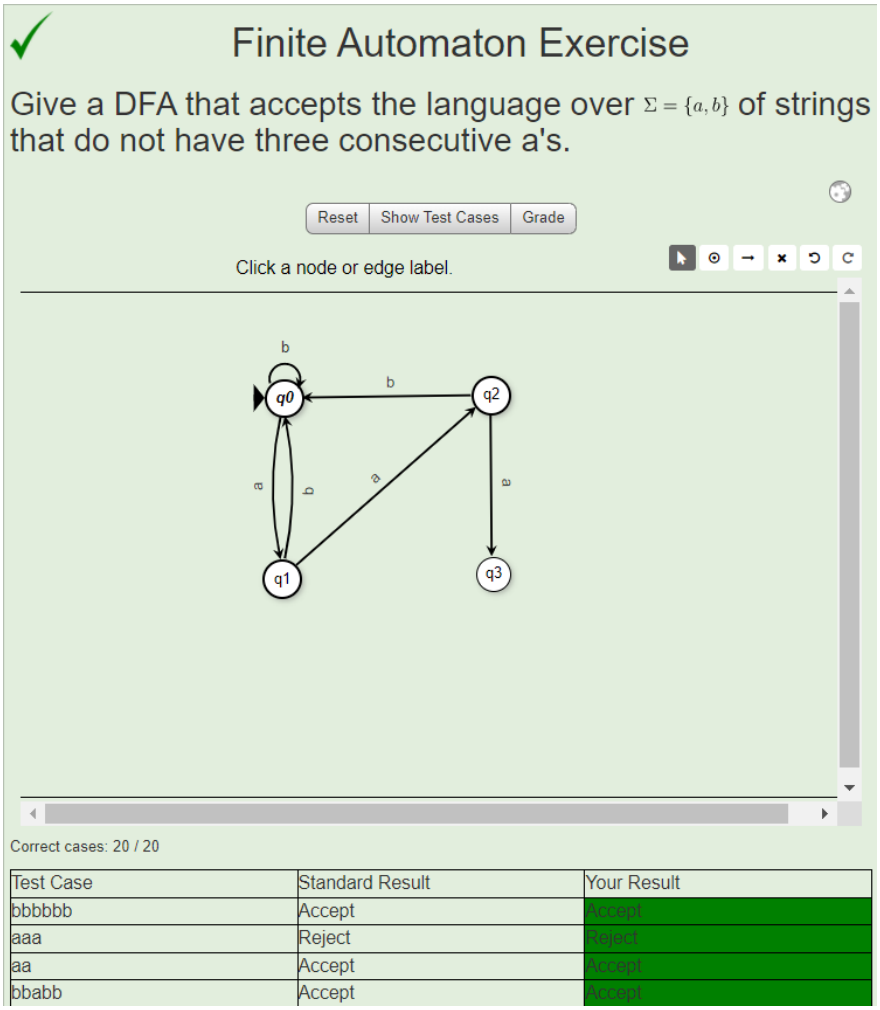

Figure 2.2: DFA auto-graded exercise

that a language is not regular. PI frame slides consist of two parts: some instructional text on the left, and a question on the right. The slideshow controller part on top of the slide lets the user move back and forth through the slides, and the canvas section below controls where the main contents are displayed. To develop PI frames, instructors can easily modify a JSON file (shown in Figure 2.4) to have customized visualizations. Students are required to corrently answer questions proposed in a slide before they can advance to the next slide; until then, the forward controller remains locked. Students get unlimited attempts until they find a correct solution.

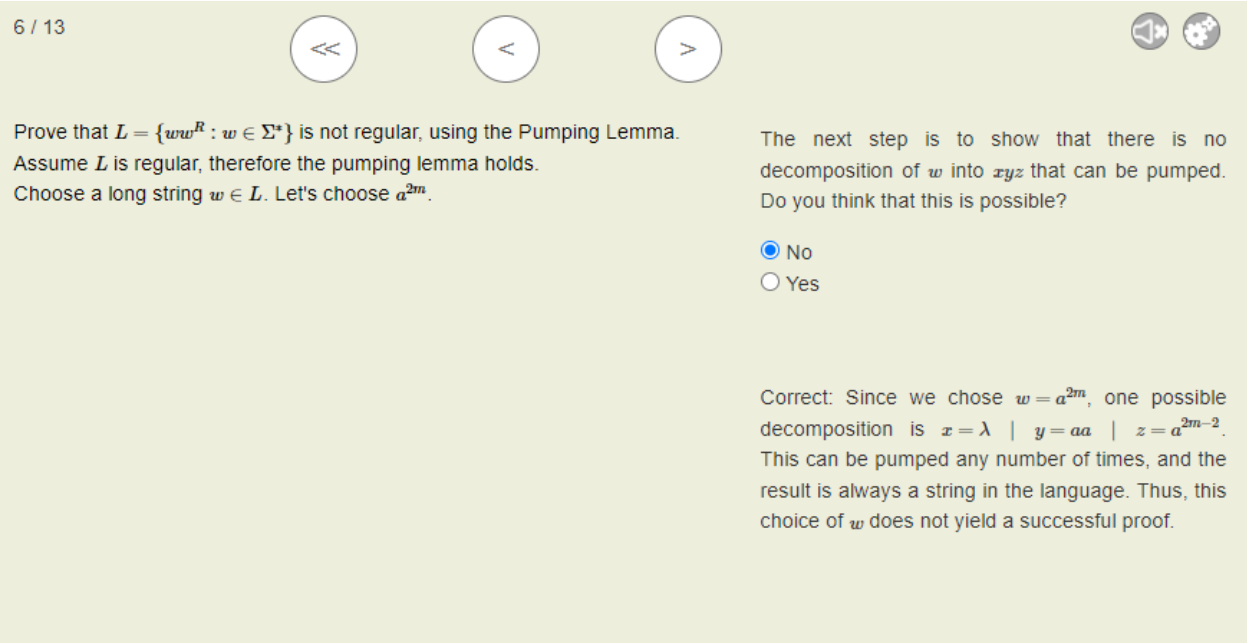

Figure 2.3: A PI Frame slide showing part of a Pumping Lemma example

```
"aswrong": {
  "type": "select",
  "description": "Prove that $L = \\{ww^R : w \\in \\Sigma^*\\}$ is not regular, using the Pumping Lemma.<br/>Assume $L$ is regular,therefore the pumping lemma holds.<br
  "question": "The next step is to show that there is no decomposition of $w$ into $xyz$ that can be pumped. Do you think that this is possible?",
  "answer": "No",
  "choices": ["Yes", "No"],
  "correctFeedback": ["Since we chose $w = a^{2m}$, one possible decomposition is $x = \\lambda\\quad |\\quad y = aa\\quad |\\quad z = a^{2m-2}$. This can be pumped any
},
```

Figure 2.4: A sample JSON file to generate contents for a PIFrame.

Instructors or course designers have numerous choices in designing their course, whether to keep the traditional textbook-like modules or choose PIFrames and integrate reading prose inside the slideshows. They can also set part of students' overall grades from completing assignments and exercises within the OpenDSA system, making students participate and interact with OpenDSA if they want to receive a better grade. A typical example is a battery of multiple choice questions. Most interactions triggered by users are recorded, allowing us to analyze and understand how students behave when navigating through modules in OpenDSA.

Additional information on the OpenDSA architecture can be found in [7].

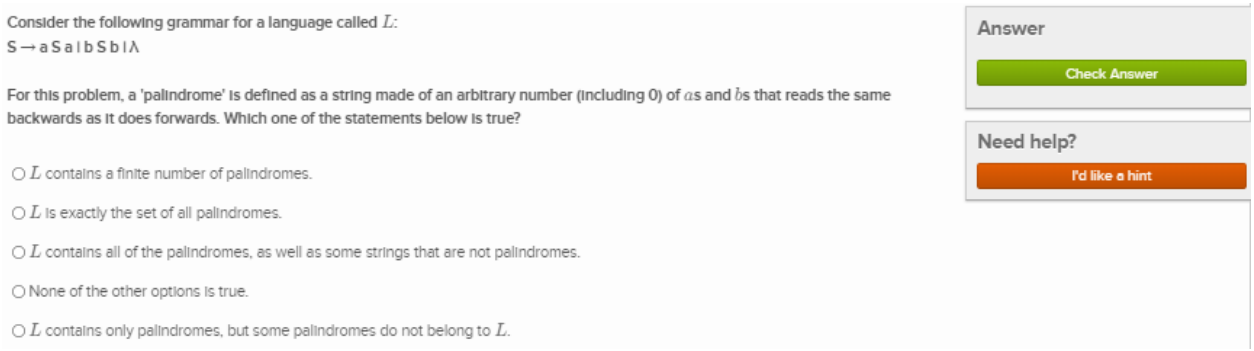

Figure 2.5: Khan academy style exercise: Characterizing Languages.

2.1.2 Prior Studies on analyzing event logs

Tax et al. propose a method called Mining Local Process Models (LPM) [14] that discovers frequent behavior patterns in event logs, typically between three and five types of events. Local process model mining focus on tracing partial behaviors focusing on frequently occurring patterns. We attempt to apply local process mining on the interaction log data using ProM [15], which is a framework that supports a wide variety of process mining techniques in the form of plug-ins, and one of the plug-ins is a LPM.

Two papers written by Carter et al., "The normalized programming state model: Predicting student performance in computing courses based on programming behavior" [16] and "Using programming process data to detect differences in students' patterns of programming" [17] proposed the Normalized Programming State Model (NPSM). The goal of those two papers is to gain a better idea of students learning process and to benefit from advanced predictions of students' course performance. The authors suggested a more holistic predictive model of student performance which characterizes student's programming activity in terms of different states, and derived a formula that can predict students' performance in a programming course. The authors investigated the relationship between the paths that students take through the programming states defined by the NPSM and their overall course achievement.

They examined the correlation between the frequency of the most common transition paths and students' overall performance in a CS2 course.

The paper successfully made a clear comparison between the two previously suggested measures and found which model is the best for explaining the variance in grades. The paper also justified why the NPSM model is a better predictive model in terms of predicting student performance in programming courses. However, further research is needed in different settings with a larger number of students, with different environments, and languages to validate the accuracy of the NPSM model. We use the method suggested in the NPSM paper to identify such learning behavior patterns within OpenDSA modules. OpenDSA's interaction data can be analyzed and divided into different states as described by Carter et al., such as reading state, exercise state, idle state, or slideshows state. We attempt to use the interaction data from FLA and DSA courses and suggest a few possible ways to analyze study behavior patterns.

Previous studies evaluating the effectiveness of visualizations of various types show the efficacy of interactive methods in delivering an engaging experience for learners. For example, a study by Farghally et al. [11] found that students tend to spend more time interacting with Algorithm Analysis Visualizations (AAVs) than with prose presentations of that same material. AAVs are enhanced versions of Algorithm Visualizations (AVs) that focus on conveying algorithm analysis concepts. Research [18] indicates that learning aids such as visualizations are better at explaining the complex and dynamic process as they help learners to track changes and observe patterns. Results indicate positive feedback from a group of students who extensively used AAVs, and final exam scores were significantly higher than another group of students who used primarily prose content. Even though the study was limited to evaluating the effectiveness of AAVs, the result is promising that interactive textbooks are more engaging and successful in learning than traditional textbooks. We need to investigate

how other interactive methods affect students' performance and carefully evaluate learning behaviors.

Fouh et al. [19] presented results from analyzing student interaction data with OpenDSA interaction log. They analyzed student interaction data within the OpenDSA system and identified basic learning behavior. The result of the research suggests that the majority of students skip directly to an exercise without reading the preceding content. While some students use the OpenDSA system as a study aid when preparing for exams, most studying patterns indicate "credit-seeking" behavior where students jump straight to exercises to get credit.

Fouh's prior work on evaluating the OpenDSA system is the the motivation for our research. His paper defines some of the students' learning behaviors and presents evidence that different behaviors can produce different outcomes. The majority of students wait until the day before the due date of an assignment to get started on the exercises, and they tend to skip directly to the exercises. The object of his research is to determine whether students are reading the text before attempting to solve the associated exercises. We want to expand the investigation further in our research. We attempt to determine how much the reading contents matter in terms of students' overall performance in a course.

Baker et al. [20] discuss prior work on analyzing so-called "game the system" behavior when students use interactive learning technologies. Examples of gaming the system include systematic guessing [21], intentional rapid mistakes, and analogous behaviors where students ask teachers and teaching assistants repeatedly for answers. Prior study [22] shows that disliking the software's subject matter and lack of self-drive are two main factors associated with a student's choice to game the system. According to Baker et al., a student who games the system typically does not learn at all when exhibiting the gaming behavior. They developed models that can infer gaming activities in log files of student interaction using

human observation and educational mining. Baker et al. discuss how gaming the system impacts learning and why students choose to game [22]. In this thesis, we also present our efforts in finding a gaming system behavior and how "game the system" behavior impacts a student's overall performance.

2.1.3 Data Collection

The OpenDSA platform collects a stream of learning process data in its database [23]. This includes event-level data such as when a user clicks a button, switches pages between modules, or gets a score on an exercise. In theory, educators could use this event stream to assess students' detailed learning processes and progress to figure out how students were interacting within the infrastructure. There have been several attempts to do this in the field of educational data mining and learning analytics [24, 25, 26]. For example, Baker et al. [27] analyzed these data to help students to achieve better performance in a class. Successful results from previous research and analysis open up new techniques and opportunities for instructors to tailor their educational tools for learners [28].

Analyzing students' learning behavior and interaction data is conducted on several computer science courses taught at Virginia Tech over different semesters. We collected data from two different courses throughout two semesters, including CS5040 Intermediate Data Structures and Algorithms (DSA) and CS4114 Formal Languages (FL) from the Fall of 2020 and Spring of 2021. Both courses typically have about 70 students each semester, but the FLA course consists of undergraduate students, while the DSA course is mainly for graduate students. FLA and DSA courses are chosen over others since their primary educational tool is OpenDSA, where portions of students' final grades are weighted by completing framesets or solving exercises that force students to use OpenDSA more than other CS courses.

The data used to investigate students' study behavior are collected from across multiple tables in the database. The odsa_user_interactions, odsa_exercise_attempts, pi_attempts, and inst_books tables are essential for gathering students' information, learning processes, and progress. Fall 2020 CS 4114 with 68 students generated 763,356 rows of raw event stream data. There were 65 students in Spring 2021 CS 4114, which generated 1,638,155 raw data. For CS5040, there were 33 and 23 students in the Fall 2020 and Spring 2021 classes, respectively. These courses generated 447,563 and 298,152 rows of raw data, respectively.

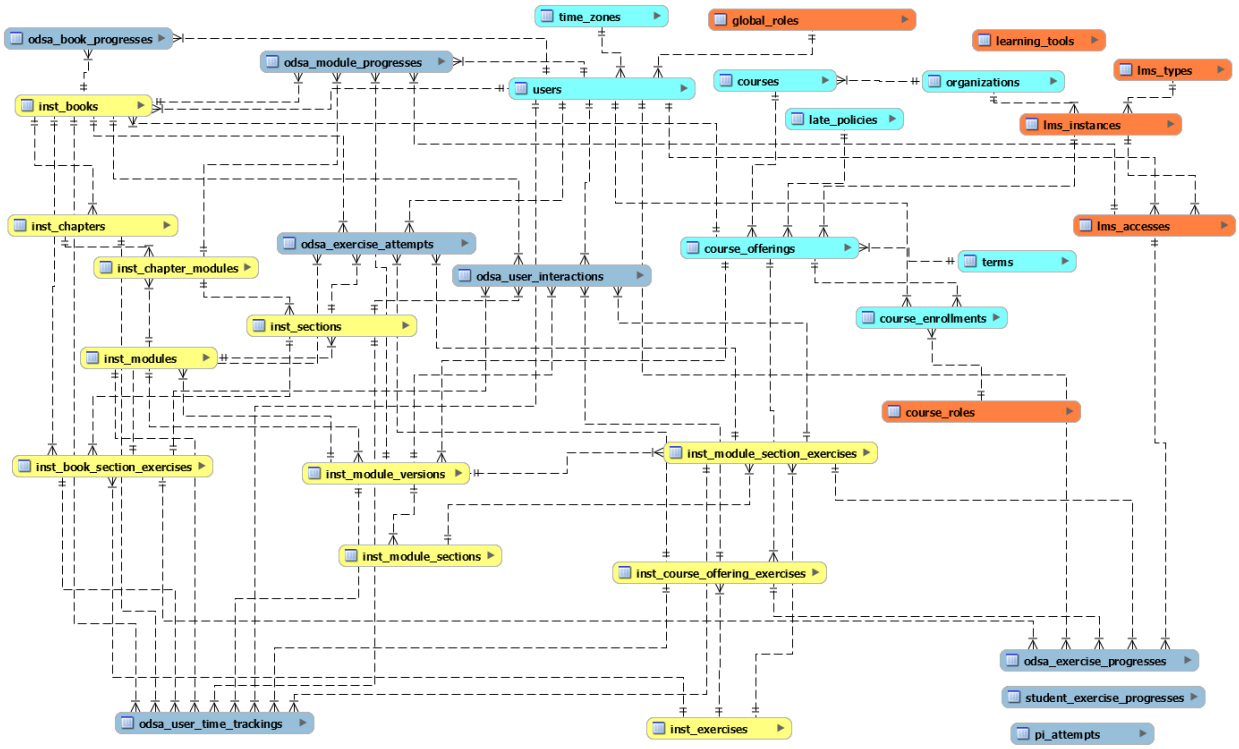

Figure 2.6: Overview of OpenDSA's database

An overview of OpenDSA's database is shown in Figure 2.6. Detailed documentation and explanation can be found at [29].

2.1.4 User Interaction Data

The core information we need to analyze students' behavior is user interaction data. The odsa_user_interactions table is used to record every user's activity from when the individual opens a module until they close it, where they were in a module in that time frame, and which slideshows or exercises they interacted with.

user_id	inst_book	name	description	action_time	inst_chapter_module_id	inst_section_id	inst_exercise_id	short_name	ex_type
58	852	document-ready	"User loaded the 01.01 Introduction to Formal Languages module"	2020-09-21 16:40:26	82434				
58	852	jsav-forward	{"ev_num":0,"currentStep":1,"totalSteps":33}	2020-09-21 16:40:50	82434	128570	1161	MajorConceptsFF	
58	852	window-focus	"User looking at 01.01 Introduction to Formal Languages module"	2020-09-21 16:40:50	82434				
58	852	jsav-forward	{"ev_num":1,"currentStep":2,"totalSteps":33}	2020-09-21 16:40:52	82434	128570	1161	MajorConceptsFF	
58	852	jsav-backward	{"ev_num":2,"currentStep":1,"totalSteps":33}	2020-09-21 16:40:54	82434	128570	1161	MajorConceptsFF	
58	852	jsav-forward	{"ev_num":3,"currentStep":2,"totalSteps":33}	2020-09-21 16:40:58	82434	128570	1161	MajorConceptsFF	
58	852	jsav-backward	{"ev_num":4,"currentStep":1,"totalSteps":33}	2020-09-21 16:41:01	82434	128570	1161	MajorConceptsFF	
58	852	jsav-forward	{"ev_num":5,"currentStep":2,"totalSteps":33}	2020-09-21 16:41:05	82434	128570	1161	MajorConceptsFF	
58	852	jsav-backward	{"ev_num":6,"currentStep":1,"totalSteps":33}	2020-09-21 16:41:08	82434	128570	1161	MajorConceptsFF	
58	852	jsav-backward	{"ev_num":7,"currentStep":0,"totalSteps":33}	2020-09-21 16:41:10	82434	128570	1161	MajorConceptsFF	
58	852	jsav-forward	{"ev_num":8,"currentStep":1,"totalSteps":33}	2020-09-21 16:41:11	82434	128570	1161	MajorConceptsFF	
58	852	jsav-forward	{"ev_num":9,"currentStep":2,"totalSteps":33}	2020-09-21 16:41:16	82434	128570	1161	MajorConceptsFF	
58	852	jsav-backward	{"ev_num":10,"currentStep":1,"totalSteps":33}	2020-09-21 16:41:20	82434	128570	1161	MajorConceptsFF	
58	852	window-focus	"User looking at 01.01 Introduction to Formal Languages module"	2020-09-21 16:41:26	82434				
58	852	window-blur	"User is no longer looking at 01.01 Introduction to Formal Languages module"	2020-09-21 16:41:26	82434				
58	852	window-blur	"User is no longer looking at 01.01 Introduction to Formal Languages module"	2020-09-21 16:41:29	82434				
58	852	window-blur	"User is no longer looking at 01.01 Introduction to Formal Languages module"	2020-09-21 16:43:09	82434				
58	852	window-focus	"User looking at 01.01 Introduction to Formal Languages module"	2020-09-21 16:43:09	82434				
58	852	window-unload	"User closed or refreshed 01.01 Introduction to Formal Languages module"	2020-09-21 16:43:15	82434				
58	852	document-ready	"User loaded the 01.02 Grammars exercises module"	2020-09-21 16:43:16	82435				
58	852	window-unload	"User closed or refreshed 01.02 Grammars exercises module"	2020-09-21 16:43:20	82435				
58	852	document-ready	"User loaded the 01.01 Introduction to Formal Languages module"	2020-09-25 00:58:54	82434				
58	852	window-focus	"User looking at 01.01 Introduction to Formal Languages module"	2020-09-25 00:58:58	82434				
58	852	jsav-forward	{"ev_num":0,"currentStep":1,"totalSteps":33}	2020-09-25 00:58:58	82434	128570	1161	MajorConceptsFF	
58	852	jsav-forward	{"ev_num":1,"currentStep":2,"totalSteps":33}	2020-09-25 00:59:07	82434	128570	1161	MajorConceptsFF	
58	852	window-blur	"User is no longer looking at 01.01 Introduction to Formal Languages module"	2020-09-25 00:59:12	82434				
58	852	window-focus	"User looking at 01.01 Introduction to Formal Languages module"	2020-09-25 00:59:16	82434				
58	852	window-blur	"User is no longer looking at 01.01 Introduction to Formal Languages module"	2020-09-25 00:59:55	82434				

Figure 2.7: Example of User Interaction Data from CS4114 course.

The structure of the user interaction table is straightforward. Figure 2.7 shows that the table contains a unique student identifier (user_id), unique book identifier (inst_book_id), name and description of the event, when did it happen (action_time), and other necessary data to help understand user interactions.

We identified 122 different event names, including many that are not useful in defining the states of the event. Thus, we grouped such events into five different states: document state, window state, PE state, FF state, and other states. The condition for splitting each event into the states is as follows:

- Document state: Events related to opening and loading a module (e.g., document-

ready)

- Window state: Events related to unloading a module (closing a browser page) or leaving a page and coming back to a module (e.g., window-focus, window-blur, window-unload)
- PE state: Events related to proficiency exercises (e.g., jsav-node-click, button-addrowbutton, submit-finish, etc.)
- FF state: Events related to slideshows and PIFrames (e.g., jsav-begin, jsav-end, jsav-forward, jsav-backward)
- Other states: Remaining events that are not part of mainstream of interaction (e.g., hyperlink, jsav-narration-on, jsav-narration-off, etc)

Chapter 3

Abstracting Session Data

We seek to support researchers and instructors who wish to answer the following types of questions about student use of eTextbooks.

- How do we define a study session?
- How do we determine semantic-level activity (e.g., reading, solving exercises state, viewing a slideshow)?
- What kinds of study patterns can we observe? Specifically, what kinds of transitions within modules can we observe?
- Which study behaviors will yield the best performance in a student's overall grade in a course?

We find that higher-level understanding is hampered as much as helped by the overwhelming mass of clickstream data to be analyzed. In order to get to the level of understanding student's semantic-level behavior, we need to organize and abstract the event-level data first. We believe that a key organizing unit of behavior is the "session". A session is a period of use of the eTextbook. But recognizing which events form a session is not trivial.

Study sessions are primarily determined by a continuous stream of interactions without a significant gap. For example, one study session can start when a student opens the module

and end when the student closes it. There are exceptional cases we need to address when considering study sessions. If a student leaves the module open for a while, meaning that we do not find a closing window interaction, do we consider that the student keeps studying, or are they away from the keyboard? If we set the time limit too short, for example, 5 minutes, the student might have been reading materials on the screen without generating recognizable events. On the other hand, if we set the gap threshold too long, multiple days of records can be considered as a single study session. We start from choosing an arbitrary time threshold to address this issue, but our analysis software can easily redefine sessions using another threshold.

Figure 3.1 shows the number of instances based on the time differences between two consecutive events of a student. The figure disregards delta times less than 60 seconds due to its massive volume, and it is convincing that a student was actively interacting if generated events are recorded with less than a 60 second gap.

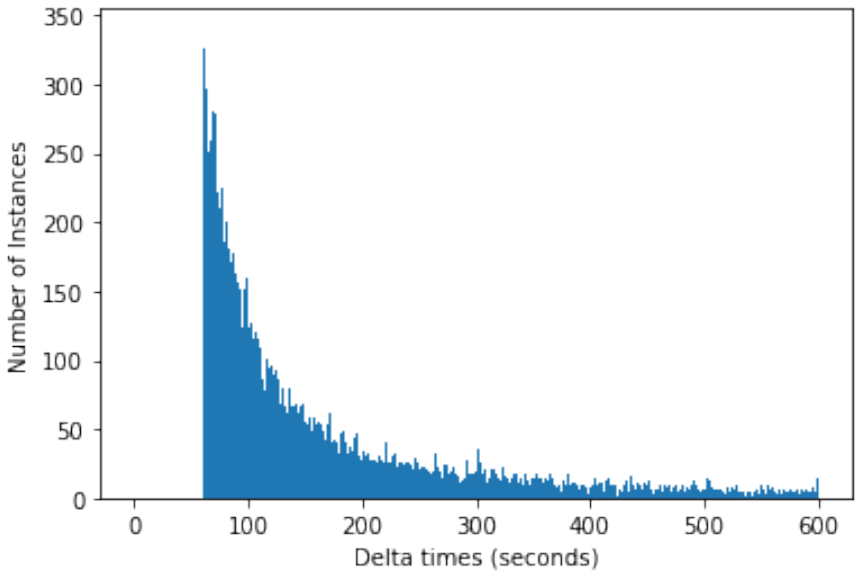

Figure 3.1: Histogram of delta time between two consecutive events of all interaction data in CS4114 Spring 2021 class. Delta times less than 60 seconds are not shown in the graph due to massive volume.

The statistic of delta time shows the 25th quartile at 88 seconds, 50th quartile at 163 seconds, and 75th quartile at 599 seconds, or about 10 minutes. Given the distribution of delta times, we abstracted sessions with thresholds of 10 minutes and one hour, and checked how sensitive the results are to these values. We find that anything in that range gives effectively the same results, so choose the threshold as 10 minutes.

Given a definition to determine events that comprise a session, we next would like to abstract these into semantic-level behavior. We need to find a way to distinguish whether a student is working on reading content, navigating through slideshows, or solving exercises. Carter et al. [16] developed the Normalized Programming State Model (NPSM), which characterizes students' programming activity in terms of the dynamically changing syntactic and semantic correctness of programs. Inspired from the way Carter's study defined programming activities as a state, we characterize a series of events on the same content element as a state, such as document state, PE state, and FF state. Each event type is classified as one of the pre-defined states described in the previous section.

Before analyzing the students' activities, we first need to pre-process the raw data extracted from the database. We mainly use database tables associated with gathering users' information, course information, and interaction data. These tables include users, odsa_ user_ interactions, odsa_ exercise_ attempts, pi_ attempts, and inst_ books tables. The osda_ user_ interactions table records all activities generated in OpenDSA from users, the inst_ books table contains information regarding the courses, and the odsa_ exercise_ attempts and pi_ attempts tables store the students' progress with associated exercises. In addition to the odsa_ user_ interactions table, we collect data from two other tables in order to collect possibly missing information in between users' interaction data. Exercise attempts and PIFrames attempts tables can provide detailed steps on how students progressed in proficiency exercises, Khan Academy style exercises, and PIFrames. The user interaction table

alone cannot tell the detailed activities in slideshows and exercises, thus the tables depicted in Figures 3.2 and 3.3 are introduced to address this shortcoming. Both tables contain user IDs, timestamp, exercise names, and other necessary information for the analysis.

user_id	inst_book_id	inst_section_id	inst_book_section_exercise_id	worth_credit	time_done	time_taken	count_hints	hint_used	points_earned	earned_proficiency	count_attempts	ip_address	question_name	request_type
7753	852	128583	217009	1	2020-08-26 20:34:16	17	0	0	0.00	0	1	66.199.90.203	SetTFequivrelp	attempt
7753	852	128583	217009	1	2020-08-26 20:34:23	6	0	0	1.00	1	1	66.199.90.203	SetTFequivrelp	attempt
7753	852	128583	217010	1	2020-08-26 20:35:15	689	0	0	0.00	0	1	66.199.90.203	SetTFpartialorderp	attempt
7753	852	128583	217010	1	2020-08-26 20:35:20	5	0	0	0.00	0	1	66.199.90.203	SetTFpartialorderp	attempt
7753	852	128583	217010	1	2020-08-26 20:35:26	5	0	0	0.00	0	1	66.199.90.203	SetTFpartialorderp	attempt
7753	852	128583	217010	0	2020-08-26 20:35:40	13	0	0	0.00	0	1	66.199.90.203	SetTFpartialorderp	attempt
7753	852	128583	217010	0	2020-08-26 20:35:43	3	0	0	0.00	0	2	66.199.90.203	SetTFpartialorderp	attempt
7753	852	128583	217010	1	2020-08-26 20:36:35	51	0	0	0.00	0	1	66.199.90.203	SetTFpartialorderp	attempt
7753	852	128583	217010	1	2020-08-26 20:36:37	2	0	0	0.00	0	1	66.199.90.203	SetTFpartialorderp	attempt
7753	852	128583	217010	1	2020-08-26 20:36:40	2	0	0	1.00	1	1	66.199.90.203	SetTFpartialorderp	attempt
6630	852	NULL	217001	1	2020-08-26 20:44:06	907	0	0	2.00	1	1598473739809	98.249.4.34	sheet1exercise6	PE
6630	852	NULL	217002	1	2020-08-26 20:54:49	287	0	0	2.00	1	1598475002015	98.249.4.34	sheet1exercise7	PE
6630	852	NULL	217005	1	2020-08-26 20:57:41	131	0	0	2.00	1	1598475330702	98.249.4.34	sheet1Practice7	PE
6630	852	NULL	217004	1	2020-08-26 20:59:25	234	0	0	2.00	1	1598475330917	98.249.4.34	sheet1Practice6	PE
6630	852	128583	217009	1	2020-08-26 21:09:46	27	0	0	0.00	0	1	98.249.4.34	SetTFequivrelp	attempt
6630	852	128583	217009	0	2020-08-26 21:10:13	26	0	0	0.00	0	1	98.249.4.34	SetTFequivrelp	attempt
6630	852	128583	217009	0	2020-08-26 21:10:14	1	0	0	0.00	0	2	98.249.4.34	SetTFequivrelp	attempt
6630	852	128583	217009	0	2020-08-26 21:10:18	4	0	0	0.00	0	3	98.249.4.34	SetTFequivrelp	attempt
6630	852	128583	217009	1	2020-08-26 21:10:20	2	0	0	0.00	0	1	98.249.4.34	SetTFequivrelp	attempt
6630	852	128583	217009	1	2020-08-26 21:10:30	10	0	0	0.00	0	1	98.249.4.34	SetTFequivrelp	attempt
6630	852	128583	217009	1	2020-08-26 21:10:38	7	0	0	0.00	0	1	98.249.4.34	SetTFequivrelp	attempt
6630	852	128583	217009	1	2020-08-26 21:10:43	5	0	0	0.00	0	1	98.249.4.34	SetTFequivrelp	attempt

Figure 3.2: Example of odsa_exercise_attempts from CS4114

After aggregating the data, we select desired course ids from the inst_books table and select all users in the course. Then we can use course id to find interaction logs within the course. Next, we need to query multiple tables beside the interaction table. The odsa_exercise_attempts and pi_attempts tables have all records of students' progress on the various exercises. We need this information in addition to the interaction data to fill in the gaps between data points from the interaction table.

Once we have all three lists of raw data, a python script merges them into a single spreadsheet sorted by user ID and then by event time. Now we have a massive sorted raw stream of data with all information about the students' interactions. The output raw data of CS 4114 Spring 2021 alone generates about 1,640,000 rows with 12 columns for 65 students.

The next step is to take the raw event stream and begin abstracting important and useful data. We first generate the sessions for individuals by finding gaps between events greater than a threshold (we currently use a time of 10 minutes).

The following step is to combine similar events within the session. We bundle the consecutive

id	user_id	frame_name	question	correct	created_at	updated_at
28540	5283	RegularExpressionsFF	0	1	2020-09-19 02:38:45	2020-09-19 02:38:45
28541	5283	RegularExpressionsFF	1	1	2020-09-19 02:38:52	2020-09-19 02:38:52
28542	5283	RegularExpressionsFF	2	1	2020-09-19 02:39:14	2020-09-19 02:39:14
28543	5283	RegularExpressionsFF	3	1	2020-09-19 02:40:16	2020-09-19 02:40:16
28544	5283	RegularExpressionsFF	4	1	2020-09-19 02:40:24	2020-09-19 02:40:24
28545	5283	RegularExpressionsFF	5	0	2020-09-19 02:40:40	2020-09-19 02:40:40
28546	5283	RegularExpressionsFF	5	0	2020-09-19 02:40:53	2020-09-19 02:40:53
28547	5283	RegularExpressionsFF	5	0	2020-09-19 02:40:55	2020-09-19 02:40:55
28548	5283	RegularExpressionsFF	5	0	2020-09-19 02:40:58	2020-09-19 02:40:58
28549	5283	RegularExpressionsFF	5	0	2020-09-19 02:41:01	2020-09-19 02:41:01
28550	5283	RegularExpressionsFF	5	0	2020-09-19 02:41:04	2020-09-19 02:41:04
28551	5283	RegularExpressionsFF	5	0	2020-09-19 02:41:05	2020-09-19 02:41:05
28552	5283	RegularExpressionsFF	5	0	2020-09-19 02:41:07	2020-09-19 02:41:07
28553	7528	RegEXandRegLangStarFF	0	0	2020-09-19 02:41:34	2020-09-19 02:41:34
28554	7528	RegEXandRegLangStarFF	0	0	2020-09-19 02:41:36	2020-09-19 02:41:36
28555	7528	RegEXandRegLangStarFF	0	0	2020-09-19 02:41:38	2020-09-19 02:41:38
28556	7528	RegEXandRegLangStarFF	0	0	2020-09-19 02:41:41	2020-09-19 02:41:41
28557	7528	RegEXandRegLangStarFF	0	0	2020-09-19 02:41:42	2020-09-19 02:41:42
28558	7528	RegEXandRegLangStarFF	0	1	2020-09-19 02:41:46	2020-09-19 02:41:46
28559	7528	RegEXandRegLangStarFF	1	0	2020-09-19 02:41:52	2020-09-19 02:41:52
28560	7528	RegEXandRegLangStarFF	1	0	2020-09-19 02:41:56	2020-09-19 02:41:56
28561	7528	RegEXandRegLangStarFF	1	0	2020-09-19 02:41:57	2020-09-19 02:41:57
28562	7528	RegEXandRegLangStarFF	1	1	2020-09-19 02:41:59	2020-09-19 02:41:59

Figure 3.3: Example of pi_attempts from CS4114

events for a single interface element (a state) into a single row, and we keep track of the total time spent on the events, the number of events, a description, a type of exercise to identify what kind of event it is. For example, a series of clicks within a slideshow can be bundled into a single line, showing summary information like the total time elapsed to complete these events and a total number of slideshow events recorded consecutively.

Figure 3.4 shows an output of abstracted data after processing the raw event stream data from CS 4114 Spring 2021. A student's personal information is anonymized using the "user ID" identifier. The figure shows that user id 812 starts the first study session on January 21st by opening Module 01.03 and then moving on to the LanguagesFS exercise. In the short span of 4 seconds, the user generates one event associated with the window state and three events associated with the FF state. On January 22nd, the user starts the second study session by opening the same module, working on the same slides for 43 seconds, then

has a window close followed by a window open 72 seconds later. Following that, the user spends 288 seconds interacting with the LanguageFS slide, generating 27 slide events. We can grasp the idea of how frequently and how diligently students study over the semester by looking at the abstracted interaction data. Of course, we can always refer back to the raw event stream data when a deeper analysis is required, but the abstracted interaction data gives instructors a higher level of understanding as to how students behave. Nonetheless, even data at this level can be overwhelming for instructors.

session	user ID	Inst Book	Event name	Event Description	Start time	End Time	Action Time	Exercise Type	Number of events
1	812	950	Window open	"User looking at 01.03 Introduction to Formal ...	2021-01-21 20:10:10	2021-01-21 20:10:10	Reading time: 1.0 sec	NaN	1
1	812	950	FF event	Attempted to solve LanguagesFS frame	2021-01-21 20:10:11	2021-01-21 20:10:14	3.0 seconds	LanguagesFS	3
2	812	950	Window open	"User looking at 01.03 Introduction to Formal ...	2021-01-22 01:37:42	2021-01-22 01:37:42	Reading time: 1.0 sec	NaN	1
2	812	950	FF event	Attempted to solve LanguagesFS frame	2021-01-22 01:37:43	2021-01-22 01:38:26	43.0 seconds	LanguagesFS	4
2	812	950	Window close	"User is no longer looking at 01.03 Introducti...	2021-01-22 01:38:30	2021-01-22 01:38:30	Away time: 72.0 sec	NaN	1
2	812	950	Window open	"User looking at 01.03 Introduction to Formal ...	2021-01-22 01:39:42	2021-01-22 01:39:42	Reading time: 12.0 sec	NaN	1
2	812	950	FF event	Attempted to solve LanguagesFS frame	2021-01-22 01:39:54	2021-01-22 01:44:42	288.0 seconds	LanguagesFS	27
2	812	950	Window close	"User is no longer looking at 01.03 Introducti...	2021-01-22 01:44:46	2021-01-22 01:44:46	Away time: 18.0 sec	NaN	1
2	812	950	Window open	"User looking at 01.03 Introduction to Formal ...	2021-01-22 01:45:04	2021-01-22 01:45:04	Reading time: 36.0 sec	NaN	1
2	812	950	FF event	Attempted to solve GrammarIntroFS frame	2021-01-22 01:45:40	2021-01-22 01:47:41	121.0 seconds	GrammarIntroFS	9

Figure 3.4: Abstracted interaction data from CS4114

We characterize an approximate compression ratio between the number of rows in event-level data and the number of rows in session-level data by presenting statistics about how big these data sets are in terms of the number of rows. Table 3.1 shows the number of rows for both raw and abstracted data, the compression ratio as well the total number of sessions abstracted. The compression ratio was measured by calculating the percentage decrease.

$$\text{Compression Ratio} = \frac{\#\text{ of rows in session data} - \#\text{ of rows in raw data}}{\#\text{ of rows in raw data}} * 100$$

About 90 percent of raw data is compressed into abstracted data for the CS4114 class, and 68 percent is compressed for the CS5040 course. We hypothesize that the huge variability in the number of sessions depends on how steadily students study in one sitting.

We use a series of python scripts to generate the processed versions of the data. One script

Table 3.1: Compression ratio statistics between a raw event stream data and abstracted data: number of rows

Class	Raw data	Abstracted data	Compression Ratio	Total Number of Sessions
CS4114 Fall 2020	763,320	101,437	86.7%	3442
CS4114 Spring 2021	1,638,155	126,412	92.3%	3490
CS5040 Fall 2020	447,153	140,332	68.6%	5242
CS5040 Spring 2021	298,138	92,969	68.8%	2543

merges the data extracted from the database with MySQL commands into a CSV file, and another does the grouping of events to create the abstracted sessions CSV file. In future work, these should be replaced with a single Ruby-based processing script, since OpenDSA's server-side framework uses Ruby on Rails.

Chapter 4

Behaviors

Chapters 2 and 3 discussed the background of OpenDSA, collecting log data of students, characteristics of log data, and abstracting event-level data. This chapter presents our effort to redefine abstracted interaction data into meaningful state data to analyze learning behaviors. We are motivated to do this by the fact that even the session-level data abstraction presented in Chapter 3 is rather a lot of information for an instructor to absorb. The work we present in this chapter shows a few possible ways to utilize abstracted data.

4.1 Different types of behaviors

In this section, we propose various types of behaviors that students are likely to exhibit while studying materials in OpenDSA. We then discuss how to recognize those behaviors from the session data. Finally, we seek to find a dissimilarity in overall performance measures between groups of students according to their study behavior. Students in our courses under study are assigned to one of two bins based on their final course scores. The first group, which we call the "higher performance" group, would be the students who received total points over the semester above some course-specific threshold. The second group, which we refer to as the "lower performance" group, consists of students with less points than the threshold. The reasoning behind the dividing points and how they are determined will be explained in Chapter 5.

In order to find a dissimilarity in these two groups, we determine four behaviors that we hypothesize are important in distinguishing performance discrepancy. A behavior is defined as a stream of transitions from one state to another, and each state represents a collection of associated events as explained in Section 2.1.4, such as a window transition state or exercise state.

4.1.1 Activity Sessions Behavior

For our first example, we propose a behavior that would naturally occur if a student opens a module and interacts with exercises or other activities such as slideshows. We count the number of transitions from document state to window state and followed by FF state (for PI framesets) or PE state (for other exercise types) for each student in the whole semester. We can easily detect any transitions between states for each unique user id from abstracted interaction data described in Chapter 3. For example, when a particular user triggers a document event followed by opening a module event, we look to see if the user transitioned from the window event to the FF event or PE event. If no other events are triggered between transitions, we can assume that the user followed the pattern we are looking for; therefore, we increment the activity sessions count by one.

An example behavior that we would expect to find as a valid transition is when a student opens a module and reads prose if there is any, then tries to solve exercises, whether a PIFrame, Khan Academy exercises, or proficiency exercises. Behaviors that we are not considering and therefore not counted include when a student closes a module without interacting with exercises (we only consider modules that have at least one exercise) or jumps between modules rather than thoroughly completing one module. Basically, we attempt to find how often the students engage with modules in a way that is possibly learning behavior.

However, as we explain later, not all activity sessions represent learning behavior.

4.1.2 Reading Prose Behavior

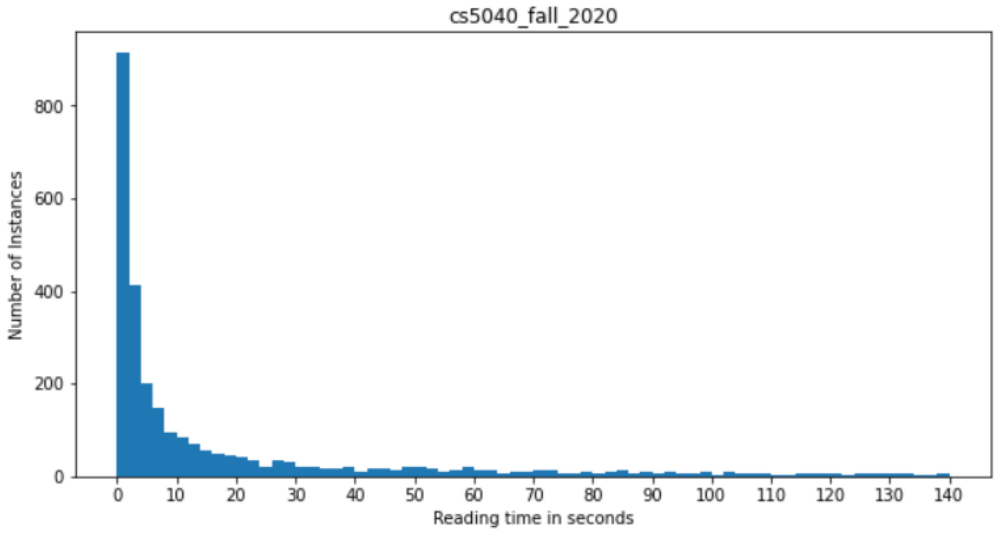

Figure 4.1: Histogram of reading prose time for CS5040 Fall 2020

Examining only the number of sessions in which students engage with exercises and slideshows is inadequate to evaluate students' behaviors, so we also consider time. A transition made from a document state to a window state or other states does not guarantee that a student was reading prose because there are other factors that we need to look for, such as students stepping away from the computer while keeping the module open. We note that using scrolling events would help to figure out which section of a module the individual was paying attention to so that researchers may investigate interaction logs with more detailed information. Unfortunately, this feature was developed after the course offerings we analyzed. Instead, we consider the estimated reading time, which is how long students spend time in the text section before jumping into solving exercises or slideshows.

The idea of Reading Prose behavior is from Fouh et al. [5], who proposed that the student

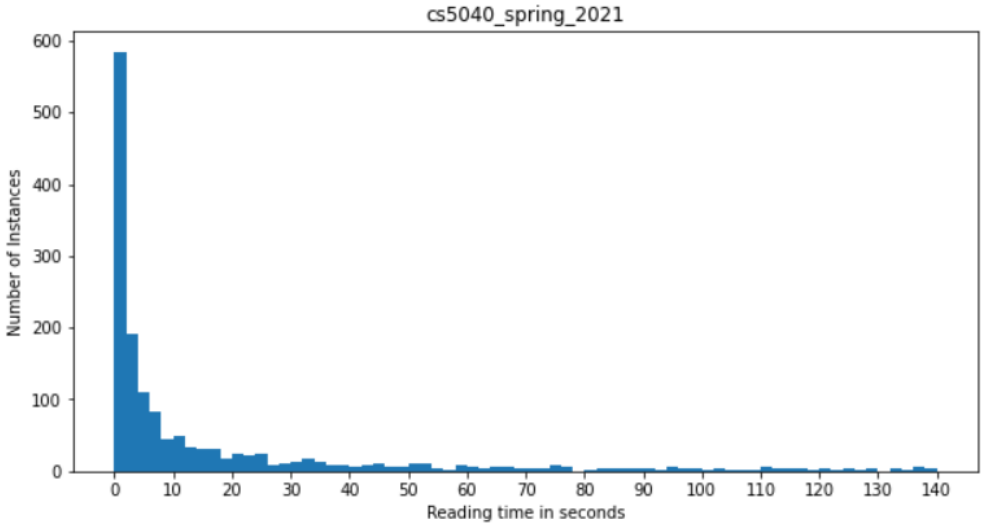

Figure 4.2: Histogram of reading prose time for CS5040 Spring 2021

reads the slide thoroughly if time spent on a each slide is between 8 and 15 seconds. Less than 8 seconds means that students did not read the text on the slide and advanced to the next slide, and greater than 15 seconds is that the student stepped away from the computer while the slideshow was up. Though his research focus was explicitly on time spent in each slide in a slideshow, we can borrow his idea of setting a threshold and investigating how long students stayed in reading contents.

We must consider how modules of each class are structured and we realized that CS4114 integrates most of the prose into PIFrames, meaning that there is not much prose students can read without engaging with exercises and PIFrames. So instead, we focus on CS5040's data only for the Reading Prose behavior.

We came up with a time range to accommodate the situation we face. If a student makes a transition from module opening to PE event or FF event within the threshold, we conclude that the person read the contents thoroughly. Any transitions that happened in less than or greater than the threshold are not counted as we specifically look for if individuals were actually reading. We draw two histograms of reading prose time of both semesters of CS5040

classes depicted in Figures 4.1 and 4.2. We identify three clusters by setting two dividing points as seen in the figures; 1) Not reading behavior where the reading time is less than 15 seconds, 2) Reading behavior where the reading time is in between 15 and 120 seconds, and finally, 3) Stepped Away behavior beyond 120 seconds which is assumed that students leave the modules open and generate no further activities.

To validate the accuracy of the time threshold for the Reading Prose behavior, we investigated several modules independently. We used the top 6 most used modules by students that contain both prose and exercises. Figures 4.3 and 4.4 show the load-to-exercise time (time difference between the initial module opening to the first exercise event) histogram and the structure of the Shell Sort module. In addition to the histogram of the Shell Sort module, all other histograms have shown a cluster that falls between 15 seconds to 120 seconds. Given the consistency of how the clusters were formed in different modules with various lengths of prose, we determined to use that time range as the threshold for the Reading Prose behavior. According to this figure, most students are not reading as most of them spent less than 15 seconds.

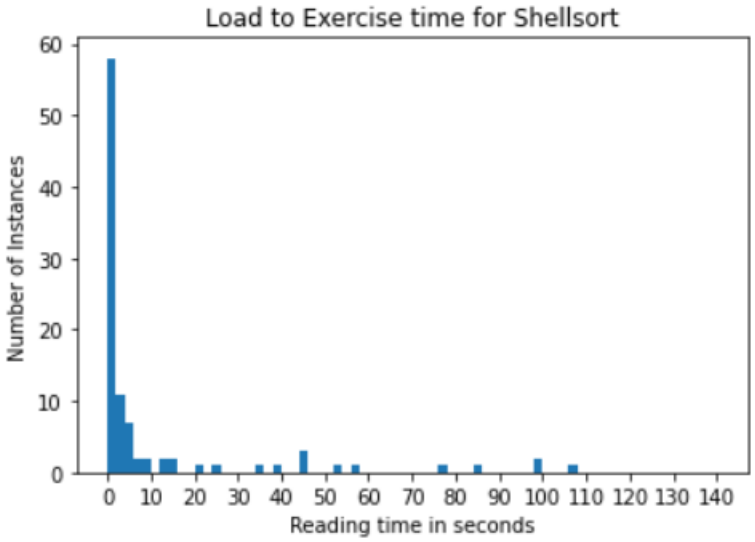

Figure 4.3: Histogram of reading prose time for Shell Sort Module

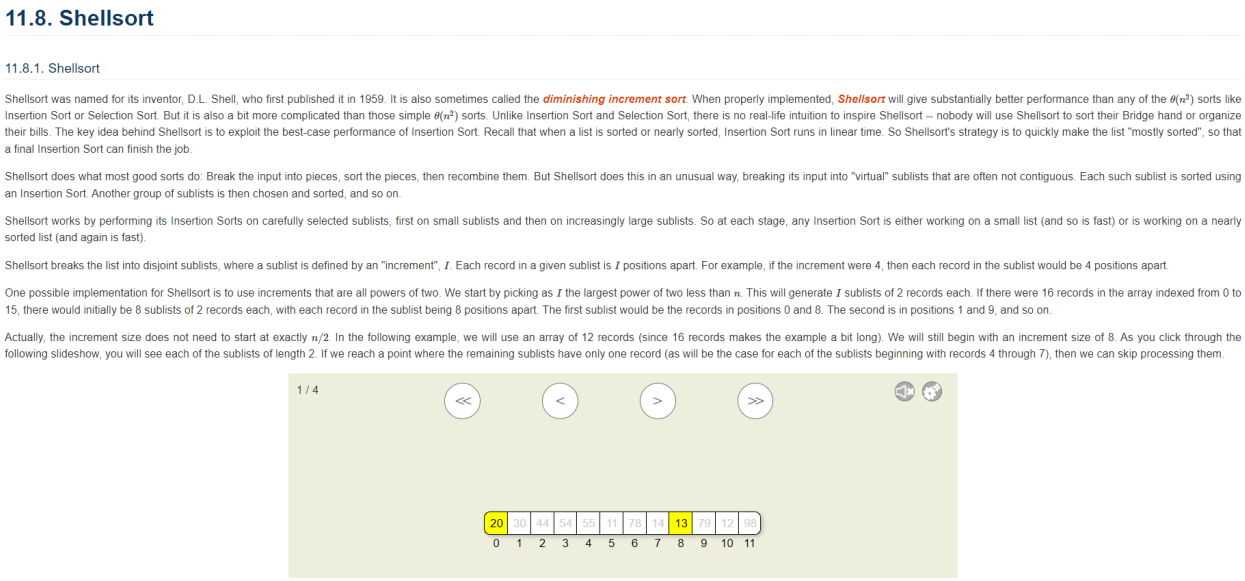

Figure 4.4: Screenshot of Shell Sort Module

In our abstracted interaction data shown in Figure 3.4, we have a column named “Action Time”, which is the computed value of the time difference between two consecutive events. If the column value contains the “Reading time” string, it indicates the time between opening the module and clicking on the exercise. We can take advantage of the value to count the number of times students read prose before moving on to the exercise states. We increment the number of occurrences count value of a student when the “Reading time” is in the threshold range.

4.1.3 Credit Seeking Behavior

The third behavior worth considering we call “Credit Seeking” behavior. Previous studies [5, 23] were conducted to identify such credit-seeking behaviors that differ from and possibly are in conflict with learning behaviors. These include jumping straight to the exercises without reading the text, clicking through slideshows as quickly as possible, and skipping to the end of slideshows.

Our third behavior focuses on recognizing credit-seeking behavior, especially the not-reading patterns. As opposed to our second activity type "Reading Prose", we are interested in finding students who did not read a majority of the text and skipped directly to the exercises. Instructors may give assignments to complete an OpenDSA module. To be specific, students earn credits from completing slideshows and solving exercises, not from reading the material. While instructors typically believe that reading the material is essential to learning, spending time on this might conflict with a student's goal of getting credit with the minimal effort.

Referring back to the histograms from Figures 4.1 and 4.2, we use 15 seconds for CS5040 as our threshold to identify credit-seeking behaviors whether students skip directly to the exercises. A similar approach from the reading prose behavior is used. We measure the total time a module was opened before a transition was triggered to engage with exercises. If it is less than the threshold of 15 seconds, we consider that as a credit-seeking pattern and record the number of patterns that occurred for each class student.

Note that our rule for recognizing "credit seeking" behavior is not exactly the inverse of "reading prose" behavior, because in both cases we rule out sessions over our threshold that we assume indicates the student is away from the computer. We cannot predict which behavior students are following in those cases.

4.1.4 PI Credit Seeking Behavior

We also propose a pattern of behavior related to completing PIFrames, which is PI Credit Seeking behavior. Modules for CS4114 tend not to have prose, but instead have nearly all content contained in PIFrames. PIFrame slides contain a brief text statement supported by visual components to help students understand computer science topics. Sometimes a subject of a module deals with complicated and complex problems. Although PIFrames have been

proved to be an excellent instructional method [13], a student may find the topic unattractive, especially for mathematical materials. That can cause students a great temptation to skip to the next slide without adequately reading and understanding the presented context. Since PIFrames do not allow skipping without answering the question on the slide, then such students are motivated to just guess the answer. The chances are higher if the slideshows are graded based on completion, meaning if students reach the end of the slide then they earn full credit. Baker et al. [30] suggests that students who engage in the gaming system behavior learn only 2/3 as much as students who do not engage in such behavior. Therefore it is imperative to investigate such behavior when analyzing the performance associated with the behavior.

As discussed in Section 4.1.2, we use a threshold of 8 seconds to determine the number of times students skipped a slide without reading a text statement by iterating through the interaction data and count the number of times the same FF events for the same exercise are recorded within 8 seconds. Instead of just counting raw occurrences of PI Credit Seeking behavior, we also consider the amount of effort the students put doing PIFrames. Otherwise a student might be recognized as having the most credit seeking behavior if the person simply interacted with PIFrames more than anybody else in the class. To prevent such cases, we consider the total number of PIFrame related events and how many of those events are identified as a credit seeking behavior. Again, we would expect to see two clusters; one cluster with higher ratio for the lower performance group and lower ratio for the higher performance group. For this pattern of behavior, we only consider CS4114's data since CS5040 modules did not utilize the PIFramesets.

Chapter 5

Courses

Based on the four types of behaviors, Activity Sessions Behavior, Reading Prose Behavior, Credit Seeking Behavior, and PI Credit Seeking Behavior explained in Section 4.1, we built analysis scripts to see if there are correlations between behaviors and performances and applied them to four course sections offered at Virginia Tech. These include two sections of CS4114 Formal Languages, and two sections of CS5040 Intermediate Data Structures and Algorithms. We will refer to the Fall 2020 CS4114 as CS4114F (CS4114 Fall) and the Spring 2021 CS4114 as CS4114S (CS4114 Spring), and the two CS5040 classes as CS5040F (CS5040 Fall 2020), and CS5040S (CS5040 Spring 2021) for the remainder of this paper.

Students in each course are grouped into high performers and low performers based on their overall GPA scores. Each section of this chapter discusses how we determine the cutoffs that separate students into two clusters (low performers and high performers). We then investigate whether there are differences between the groups with respect to the four behaviors to analyze whether the behaviors are impacting students' performance. For example, one can hypothesize that the high performer group would exhibit more patterns of Reading Prose behavior than the low performer group. If there are statistically significant differences between them, we can assume that the behavior is one factor affecting the performance.

5.1 CS4114 Fall 2020

Dr. Mostafa Mohammed was the primary instructor of CS4114F, and this class used OpenDSA as the primary resource for the class material. Dr. Mohammed's class had 68 undergraduate students, and the grade distribution of students is shown in Figure 5.1. We set two groups of students, higher performance and lower performance groups, with the initial points of division at 82 and 87. Students with more than 87 points are grouped in the higher performance group, and those with less than 82 points are in the lower performance group. The rationale behind the clustering is based on the grade distribution histogram. To ensure that both groups have the most balanced number of students and to observe meaningful differences between them, we discarded scores between 82 and 87. We tried a couple of alternatives, such as drawing the single point of division at 80, 85, and 90 but excluding scores between 82 and 87 results in the best in terms of noticeable observation between the groups.

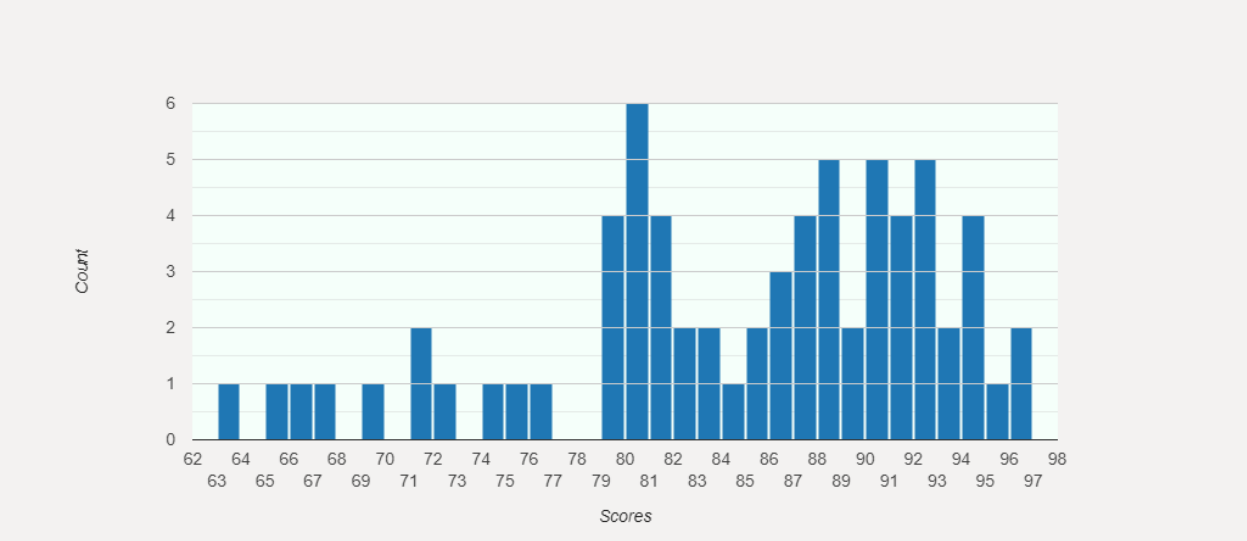

Figure 5.1: Grade Distribution of CS4114F class

5.2 CS4114 Spring 2021

Dr. Mohammed also taught CS4114S, and this class used OpenDSA as the primary tool for the class material. CS4114S had 66 students and grade distribution of the class is shown in Figure 5.2. Using the same process to cluster students into multiple bins was problematic for CS4114S since it had more diversity in grade distribution. As seen in Figure 5.2, there are three big groups 1) scores less than 80 except for one outlier data point between 5 and 10, which we dropped from the data, 2) scores between 81 and 89, and finally 3) scores more than 90. The experiments for CS4114S started with the three clusters and tried to identify the correlation between them with students' performances by applying an ANOVA. However, using ANOVA on the three groups could not figure a meaningful correlation as ANOVA indicated no significant differences among them. Instead, alternative experiments were conducted to cluster students into two groups, including dropping the entire center group and setting the dividing lines at 80%, or 90%. None of the experiments successfully classified the correlation between the patterns of behaviors and students' overall performances. At this point, we decided to use the same metric as CS4114F to group students into two bins. The HP group with more than scores of 87, and the LP group with less than 82. This clustering yielded a more balanced sample size in each group and produced better outcomes in terms of the differences in groups for all experiments conducted with the CS4114S dataset. The HP group has 24 students, and the LP group has 23 students. We expect to see more clear differences between clusters regarding the correlation between their performances and behaviors.

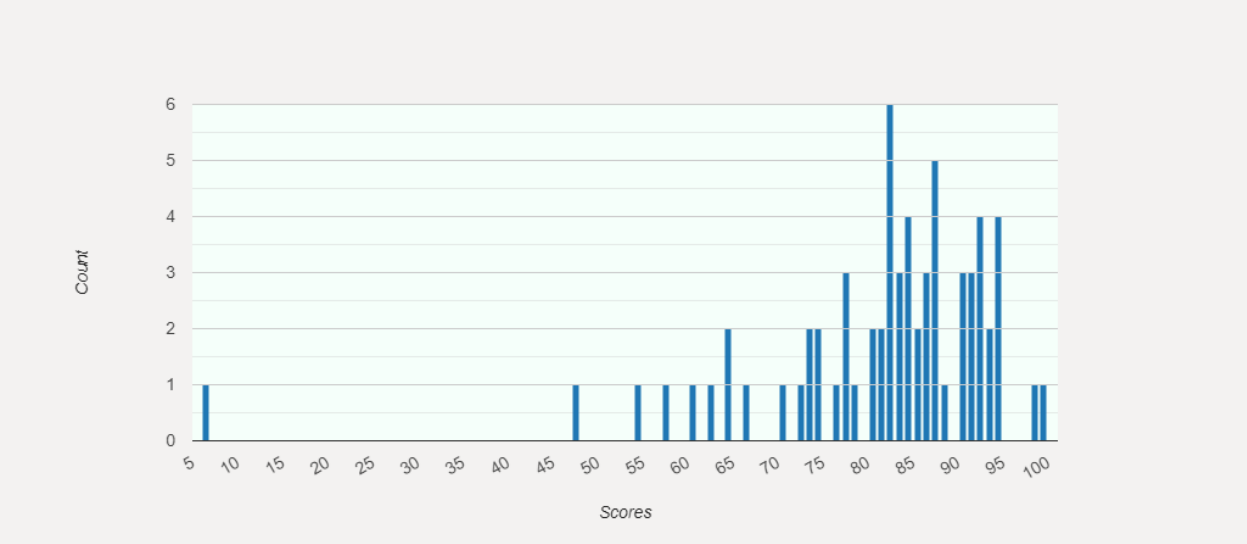

Figure 5.2: Grade Distribution of CS4114S class

5.3 CS5040 Fall 2020

Like Formal Languages classes, CS5040 classes used OpenDSA to cover the data structures and algorithms material, and OpenDSA was used as the primary resource. Dr. Bob Edmison was the instructor of the course, and he taught the class using modules from OpenDSA, administered an exam, and assigned homework from OpenDSA. The primary target audience for the course is computer science graduate students, but there were several students from materials science and engineering (MSE), computer engineering (CPE), electrical engineering (EE), and mining and minerals engineering (MINE). There were 35 students in the class, and their grade distribution is shown in Figure 5.3. About 75% of the student population belongs to the High-Performance Group (HP), representing students who completed the course with a final score of 90 or more. Another nine individuals belong to the Low-Performance Group (LP), students with a final score of less than 90. Unfortunately, due to some technical problems with OpenDSA infrastructure, one of the student's event log data was not recorded for the whole semester. The individual would be part of the HP group, and we disregarded

the person's data point due to missing interaction data. This should be a minor change since the majority population belongs to the HP group. The outlier who belongs to the less than a score of 35% is also disregarded. We also noticed the disproportion of the grade distribution and a smaller sample size that might lead to inaccurate results.

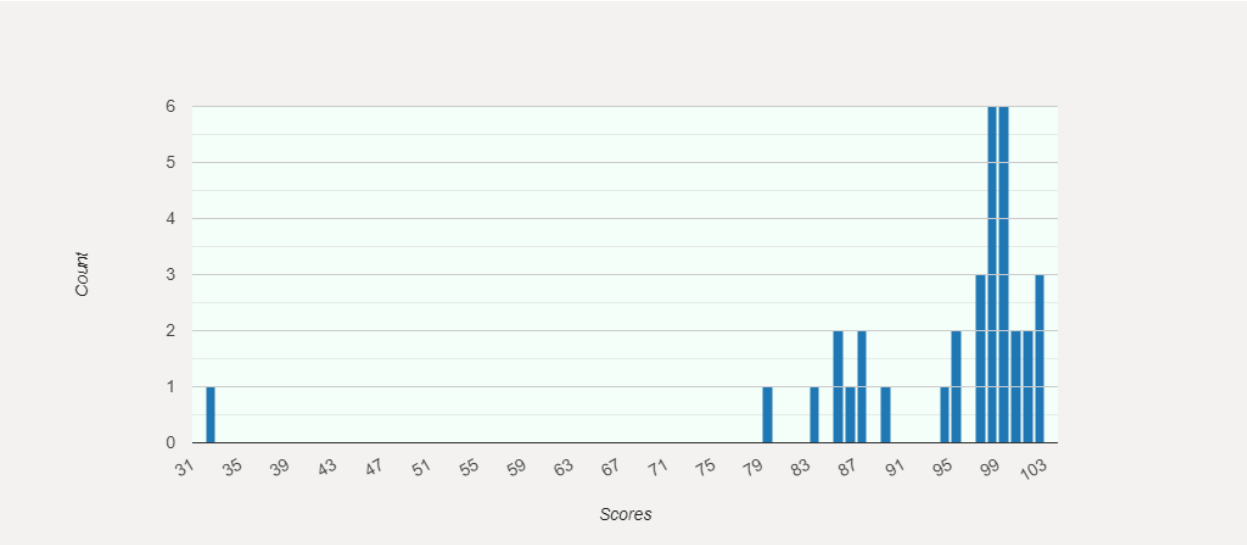

Figure 5.3: Grade Distribution of CS5040F class

The basic structure of modules in CS5040F differs from CS4114 classes. CS5040 modules contain a more traditional textbook-like appearance with a substantially large portion of the reading section supported by visualizations. The class uses conventional slideshows (instead of PIFrames), proficiency exercises, and khan academy exercises. Codeworkout exercises are added to the course to provide a drill-and-practice system that supports short programming exercises and brings a place where students can practice writing algorithms independently. A more detailed explanation of codeworkout can be found in [31].

5.4 CS5040 Spring 2021

We investigated the same patterns for the CS5040 Spring of 2021's interaction logs. Dr. Edmison also taught the course, and he used the same materials from the previous semester for his lecture notes and assigning homework. Spring of 2021, enrollment was 24 compared to 34 for CS5040F. The grade distribution is shown in the Figure 5.4. The score in the below 24% bin is discarded, which left us with a total of 23 student data points. The point of division for CS5040S is at a score of 94% by following the same process to choose a division point. The HP and LP groups' grades are distributed relatively equally based on the dividing line at 94. There were ten students in the HP group, and 13 students were classified as the LP group.

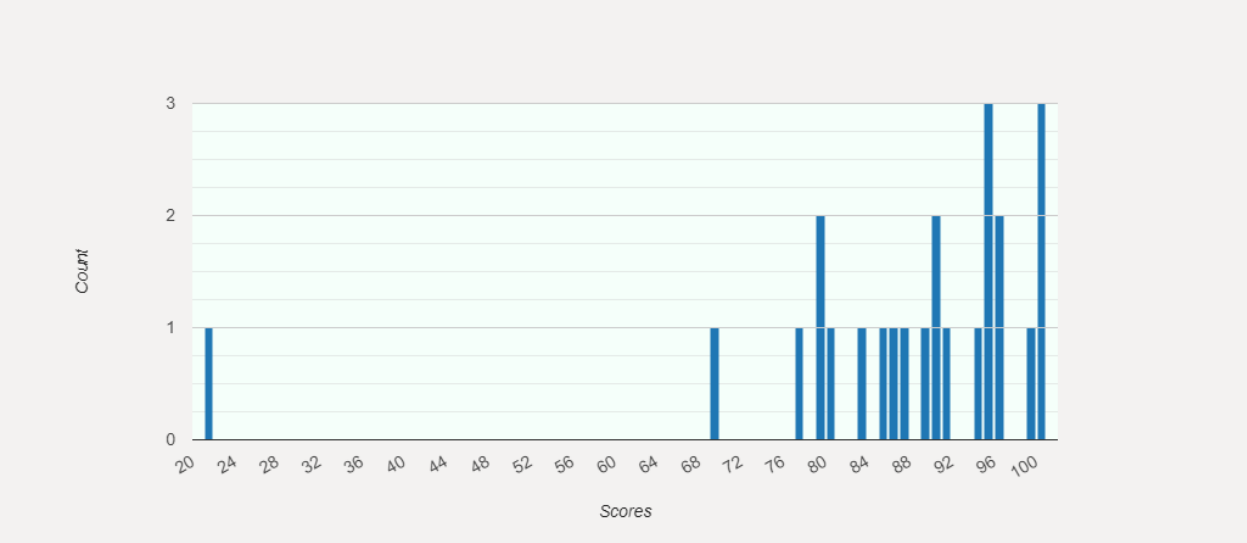

Figure 5.4: Grade Distribution of CS5040S class

To be more consistent with CS4114 in terms of sample size, we pooled both semesters of CS5040 data together, which makes a total sample size of 56, excluding two outliers who scored below 35. The grade distribution of concatenated data is shown in Figure 5.5. Notice the dividing line is at around a score of 94, which we used as a score threshold in separating

students into high/low performance groups. The rest of the experiments for CS5040 will be performed with the combined data.

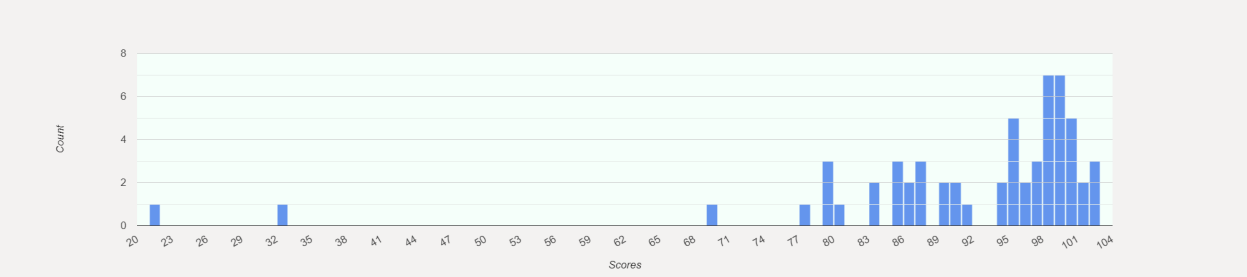

Figure 5.5: Grade Distribution of Two CS5040 Classes

Chapter 6

Evaluation

6.1 Activity Sessions Behavior

6.1.1 CS4114 Fall 2020

We expect that students who put more effort into doing exercises and otherwise using the interactive materials will perform better in the class than students who put in less effort. Activity Sessions behavior attempts to be a measure of the level of effort. We recognize that this measure has limitations since students could simply break their work into more sessions, for example. Table 6.1 summarizes the differences between the higher performance group (HP) and the lower performance group (LP). The table shows that the HP group has a higher Activity Sessions count than the LP group. The count values in the table indicate the number of times that the behavior was exhibited, in this case when students generate consecutive transitions of states starting from module states to exercise states. We observe that the average number of occurrences of the behavior for the high-performance group is 287, whereas the lower performance groups' average occurrences number is 182. This can be interpreted as the HP group following the pattern 58% more than the LP group. The lowest occurrence counts of the HP and LP groups were 32 and 26, respectively, and the highest counts were 958 and 438.

We also conducted two-sample t-tests to compare the means of occurrence count, and the

Table 6.1: CS4114F Activity Sessions Behavior Overview

	Higher Performance	Lower Performance	Entire Class
# of students	33	25	68
Lowest number of occurrence	32	26	26
Highest number of occurrence	958	438	958
Average number of occurrence	287	182	241
Average final score	91.47	76.09	84.8

result is represented in Table 6.2. The p-value is measured to be 0.013, indicating a significant difference between the average number of times that the Activity Sessions behavior was exhibited for students in the two clusters.

Table 6.2: CS4114F Activity Sessions Behavior T-test

	High Performance	Low Performance
Average Number of Occurrence	287	182
Variance	38464	13017
Std. Deviation	193	112
t Stat	2.56	
p-value (two-tailed)	0.013	

6.1.2 CS4114 Spring 2021

We conducted the same experiment on Spring 2021 CS4114's interaction logs. Again, we used the same logic to find a sequenced pattern: opening a module, studying a reading material, and solving exercises. As we cluster CS4114S's students into two groups, we observe average occurrences of Activity Sessions behavior as 441 and 231 for the HP group and LP group, respectively (See Table 6.3). Looking at the average occurrence numbers, we hypothesize that students with higher performance metrics tend to generate more Activity Sessions behavior.

We examine whether this behavior would affect differently for each of the groups by applying a t-test (See Table 6.4). Given that the P-value is $0.01 < 0.05$, there are significant differences

Table 6.3: CS4114S Activity Sessions Behavior Overview

	High Performance	Low Performance	Entire Class
# of students	24	23	65
Lowest number of occurrence	87	0	0
Highest number of occurrence	1065	768	1065
Average number of occurrence	441	231	345
Average final score	91.9	65.2	82.3

between groups; therefore, in addition to what we have seen in the CS4114F, this behavior correlates with students' performance in CS4114S. There is about a 63% difference in the average occurrence counts.

Table 6.4: CS4114S Activity Sessions Behavior Summary and T-test

	Higher Performance	Lower Performance
Average Number of Occurrence	441	231
Variance	78186	38309
Std. Deviation	274	196
t Stat	2.70	
p-value (two-tailed)	0.01	

6.1.3 CS5040

Because of how the CS5040's modules are structured, we modified the algorithm to find Activity Sessions behaviors. CS5040 depends more heavily on proficiency exercises and khan exercises than slideshows. Instead of transitions starting from opening modules and moving onto the PIFrames, we consider transitions to traditional slideshows, proficiency exercises, and khan academy exercises. Table 6.5 shows an overview of result for the class, and result from a t-test is shown in Table 6.6. The P-value is 0.85, indicating no significant difference between the groups, and there is no proof that the Activity Sessions behavior affects the student's overall performance. It is worth noting that the students in the HP

group generated less of the pattern, meaning that the LP group students generally generated activity sessions. The nature of CS4114 and CS5040 can also be considered with respect to the different outcomes. CS4114 focus on the theoretical concepts of Formal Languages, while CS5040 is a programming-intensive course, so we believe that investigating the nature of courses can improve the accuracy of outcomes.

Table 6.5: CS5040 Activity Sessions Behavior Overview

	High Performance	Low Performance	Entire Class
# of students	35	21	56
Lowest number of occurrence	59	82	59
Highest number of occurrence	545	609	609
Average number of occurrence	253	259	256
Average final score	98.61	84.51	93.32

Table 6.6: CS5040 Activity Sessions Behavior T-test

	High Performance	Low Performance
Average Number of Occurrence	253	259
Variance	9874	12441
Std. Deviation	98	109
t Stat	0.19	
p-value (two-tailed)	0.85	

6.2 Reading Prose Behavior

6.2.1 CS5040

The second pattern is designed to observe a behavior of spending enough time to read prose before jumping into the later section of a module. Ensuring that students spend plenty of time reading prose to understand the module's topic before solving exercises is viewed as a good way to study a module.

For CS5040, we expected to see more patterns that satisfy the second behavior criteria, which is that the time spent before jumping to the exercises is between 15 seconds to 120 seconds. The threshold range is chosen based on the data-driven approach explained in Subsection 4.1.2. This course generally has a higher reading volume, and students would have to spend some time going through the prose.

The average numbers of Reading Prose behavior were 18 and 15 for the HP group and the LP group, respectively (See Table 6.7). These numbers indicate that students exhibited a behavior of spending a good amount of time reading prose before engaging with exercises about 18 and 15 times for the HP and LP groups respectively. In line with the prediction we made in the first place that the HP group would exhibit more Reading Prose behavior, the outcome suggests that the HP group shows more Reading Prose behavior. The statistic shows that students with higher performance spent more time reading prose.

Table 6.7: CS5040 Reading Prose Behavior Overview

	High Performance	Low Performance	Entire Class
# of students	35	21	56
Lowest number of occurrence	0	0	0
Highest number of occurrence	61	69	69
Average number of occurrence	18	15	17

However, the P-value from Table 6.8 points out that the two groups still show no significant differences, and we cannot conclude that the second behavior of reading prose, again, is correlated to students' performance. This could be due to the small sample size.

Table 6.8: CS5040 Reading Prose Behavior T-test

	High Performance	Low Performance
Average Number of Occurrence	18	15
Variance	238	279
Std. Deviation	15	16
t Stat	0.61	
p-value (two-tailed)	0.54	

6.3 Credit Seeking Behavior

6.3.1 CS5040

The outcome of credit-seeking behavior of CS5040 is what we initially expected to observe. As seen in Table 6.9, the LP group's average credit-seeking pattern count is greater than the HP group's count by about 7%. On average, students in the LP group followed a pattern of credit-seeking by skipping prose and immediately engaging with exercises presented in a module. The HP group generated 2231 times of credit-seeking behavior, while the LP group generated 2386 times on average per person throughout the semester.

Table 6.9: CS5040 Credit Seeking Behavior Overview

	High Performance	Low Performance	Entire Class
# of students	35	21	56
Lowest number of occurrence	162	376	162
Highest number of occurrence	4708	6021	6021
Average number of occurrence	2231	2386	2289

Table 6.10 indicates that the correlation between the credit-seeking behavior and students' performance cannot be justified. The t-test indicates that the P-value is 0.66, representing that the mean values of the two clusters are not significant. Correspondingly, students' performance could not be justified by examining the credit-seeking behavior in CS5040.

Table 6.10: CS5040 Credit Seeking Behavior T-test

	High Performance	Low Performance
Average Number of Occurrence	2231	2386
Variance	974539	2022958
Std. Deviation	973	1388
t Stat	0.44	
p-value (two-tailed)	0.66	

6.4 Reading Prose vs Credit Seeking

As an additional step to measure the correlation between the students' behavior and their performance, we decided to compare the ratio between the Reading Prose and Credit Seeking behaviors of CS5040 classes. Since we obtained the occurrence counts of each student for both Reading Prose and Credit Seeking behaviors, we can calculate their ratio by dividing the number of times that a Credit Seeking behavior occurred by the number of Reading Prose behavior exhibited. Then we compute the average ratio values for the HP and LP groups to observe the relationship.

The CS5040 classes show that the ratio of the LP group is higher as shown in Table 6.11. The higher performance group's ratio of CS5040 is 237, which means that students in the group generated a behavior of Credit Seeking 237 times more than a Reading Prose behavior on average. In comparison, the LP group's ratio is 240, suggesting that students in the LP group tend to exhibit either less of a Reading Prose behavior or more of a Credit Seeking behavior. These ratios are nearly identical, and the high p-value bears this out.

Table 6.11: Comparison between the Reading Prose and Credit Seeking behaviors

CS5040	High Performance	Low Performance
Average Ratio	237	240
Variance	119385	30395
Std. Deviation	341	170
t Stat	0.044	
P-value	0.96	

6.5 PI Credit Seeking Behavior

6.5.1 CS4114 Fall 2020

We also studied a form of credit-seeking behavior among users of PIFrames in CS4114. We predict that the LP group would show more PI Credit Seeking behavior ratio. They would hunt for correct answers to advance to the next slide by guessing every possible combination of choices instead of fully understanding a text statement and visual components presented in a slide. The result indicates that the LP group has a higher ratio between the total number of PIFrame-related events and PI Credit Seeking occurrences. The LP group produced a higher PI Credit Seeking behavior ratio than the HP group, with about a 5% increase in the average ratio. This means that the LP group students followed the PI Credit Seeking behavior pattern by advancing to the next slide without understanding each slide's materials more than the HP group. According to the average ratio, the LP group is more credit seekers. However, the result from the t-test indicates that the P-value is 0.24, meaning that the correlation between a PI Credit Seeking behavior and students' performance is insignificant.

Table 6.12: CS4114F PI Credit Seeking Behavior Overview

	High Performance	Low Performance	Entire Class
# of students	33	25	68
Lowest Ratio	1.302	1.472	1.30
Highest Ratio	2.452	2.732	2.732
Average Ratio	1.790	1.883	1.810

Table 6.13: CS4114F PI Credit Seeking Behavior T-test

	High Performance	Low Performance
Average Ratio	1.79	1.883
Variance	0.07	0.10
Std. Deviation	0.265	0.310
t Stat	1.19	
p-value (two-tailed)	0.24	

6.5.2 CS4114 Spring 2021

PI credit-seeking behavior in CS4114 of Spring 2021 appears to have a similar result as CS4114F class. The LP group generated a higher average ratio of PI Credit Seeking behavior than the HP group (See Table 6.14, meaning that the LP group interacted with PIFrames for credits 12% more than the HP group did on average.

Table 6.14: CS4114S PI Credit Seeking Behavior Overview

	High Performance	Low Performance	Entire Class
# of students	24	23	65
Lowest Ratio	1.32	0	0
Highest Ratio	2.69	7.67	7.67
Average Ratio	1.67	1.82	1.7

In order to help understand how students of the groups respond to the behavior, we also conducted the t-test as shown in Table 6.15. The P-value for PI credit-seeking behavior of the clusters is 0.60, meaning a statistically non-significant difference between the clusters,

and the means of the different groups are insignificant. Therefore, we cannot assume that how many times students try to game the system for credit affects their final score or overall performance in a Formal Languages class.

Table 6.15: CS4114S PI Credit Seeking Behavior Summary and T-test

	Higher Performance	Lower Performance
Average Ratio	1.67	1.82
Variance	0.11	1.75
Std. Deviation	0.32	1.32
t Stat	0.54	
p-value (two-tailed)	0.60	

6.6 Summary

In this section of the thesis, we described the experiments conducted on the Formal Languages and Data Structures and Algorithm courses throughout the semesters of Fall 2020 and Spring 2021. In addition, we presented an analysis of students' interaction log data generated by using OpenDSA to distinguish the correlation between student interaction behaviors and performance.

We clustered students in each class into bins based on the final overall score. Based on the grade distribution, students in all of the classes are grouped into two bins: the higher and lower performance groups.

The behaviors of the groups were then analyzed on the criteria we created. We developed four types of patterns associated with the interaction logs and extracted 1) Activity Sessions behavior, 2) The Reading Prose behavior when the time between the initial module page load and the first attempt to solve exercise is in the range of 15 to 120 seconds for CS5040, 3) A Credit-seeking behavior when the load-to-exercise time is less than 15 seconds for CS5040,

and finally, 4) A PI Credit-seeking behavior, which is the time spent per slide is less than 8 seconds.

Table 6.16: Overview results for 2 types of behaviors for CS4114

		CS4114F		CS4114S	
		HP	LP	HP	LP
Activity Sessions	Avg. occurrence	287	182	441	231
	P-value	**0.013**		**0.01**	
PI Credit Seeking	Avg. ratio	1.79	1.88	1.67	1.82
	P-value	0.24		0.60	

Table 6.17: Overview results for 3 types of behaviors for CS5040

		CS5040	
		HP	LP
Activity Sessions	Avg. occurrence	253	259
	P-value	0.85	
Reading Prose	Avg. occurrence	18	15
	P-value	0.54	
Credit Seeking	Avg. occurrence	2231	2386
	P-value	0.66	
Reading Prose vs Credit Seeking	Avg. Ratio	237	240
	P-value	0.96	

The results from all classes are shown in Tables 6.16 and 6.17. The average occurrences numbers of the groups, average ratios, and the P-values from the t-test are displayed in the table. For CS4114 courses, the Activity Sessions behaviors of both classes have significant relationships to the performance of students in terms of their overall course grades. The PI Credit Seeking behaviors indicated no significant differences between the higher and lower performance groups. However, we can observe that the lower performance group tends to generate a higher PI Credit Seeking behavior ratio. For the CS5040, the majority of experiments show insignificant outcomes. None of the behaviors could tell the discrepancy

in students' performance in terms of whether there are significant differences between the means of the groups. The differences in the average number of times that behavior was observed are insignificant, and the P-values are higher than 0.05.

The works we presented in the thesis may not be ideal for measuring the correlation between performances and behaviors. However, the main goal of the research was to open the door to abstracting the raw interaction logs into session-level data, then to show how we can utilize the abstracted data in a meaningful way. Of those approaches, we have shown a few possible behaviors that can be observed by taking advantage of the abstracted session-level data. We hope that the efforts we put into this research can inspire future works so that researchers can tailor the instructional methods for learners.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we presented our efforts to evaluate students' behaviors within the OpenDSA infrastructure to suggest future guidelines to instructors or course designers. Understanding how students engage and interact with the eTextbook is essential in creating effective instructional methods and improving the existing system. Therefore, we presented our attempts to extract several common behaviors from students' interaction logs and abstract raw data into refined data for user interaction analysis purposes.

An overwhelming mass of clickstream data was hindering the understanding of students' behavior. To overcome such shortcomings, we abstracted the event-level data first by organizing it into a session, and defined activity states, including document state, reading state, and exercise state. Once we developed a strategy to abstract data, selected course's interaction logs were extracted from the database and processed to generate abstracted data. After we acquired all the event-level data and student interaction logs, we investigated the semantics of students' behaviors.

We identified four behavior patterns and hypothesized that these patterns would play a role in determining students' performance in a class. We expected to see a distinction between the higher-performance and lower-performance students. We tried to see the relationship

between performances and behaviors by examining how many times the interaction logs show patterns for the two experimental student groups. The Activity Sessions behavior we defined played a role in determining students' performance if they generated more Activity Sessions behaviors when studying in the Formal Languages class. On the other hand, we could not conclude which behavior would be an ideal pattern for students in CS5040 to follow, as most of the behaviors indicated no significant differences in the number of times a particular behavior was exhibited.

7.2 Future Work

OpenDSA interaction data could be used to study many more aspects of student performance, and there remains much work to be done. We propose potential research questions which we were regrettably unable to address within the scope of this thesis. We hope to see more evidence of the effectiveness of OpenDSA as an online learning platform by analyzing interaction log data.

7.2.1 Additional Learning Behaviors

The four behaviors we investigated in this study cannot explain all of the behavior observed while interacting within the OpenDSA. The existing mechanism to find a pattern of behavior relies on computing the difference between timestamps or searching for transitions from state to state. In addition to this technique, a more complex pattern of behaviors can be found, such as students frequently jumping in and out of different parts of a module to look at study aid materials elsewhere. A future study can also consider the case when students open multiple modules simultaneously, switching browser tabs, looking for materials from

different modules. Considering these behaviors would play an essential role in evaluating the correlation between study patterns and students' performance.

7.2.2 Time Consideration to Sessions

It seems reasonable to expect that time spent is an import influence on performance. Measuring the time spent in each state between the transitions within a session would solve many problems we faced during our experiments. One problem we had is determining if students were reading materials or not, or even if they were paying attention to the screen during periods where there are gaps between. Although we could not use the tracking scroll events as briefly mentioned in the earlier section, a future study can benefit from the feature. The existing interaction data are not capable of distinguishing students who are looking at the monitor, reading prose, and scrolling down while not generating logs from students who stepped away from the computer. Such a problem might be resolved by tracking the scrolling events that can tell whether students were actively interacting with a module even if no interaction logs were recorded.

Bibliography

[1] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *J. Educ. Resour. Comput.*, vol. 5, p. 4–es, sep 2005.

[2] S. H. Edwards, "Using software testing to move students from trial-and-error to reflection-in-action," *SIGCSE Bull.*, vol. 36, p. 26–30, mar 2004.

[3] S. Cooper, W. Dann, and R. Pausch, "Alice: A 3-d tool for introductory programming concepts," *J. Comput. Sci. Coll.*, vol. 15, p. 107–116, apr 2000.

[4] M. Kölling, "The greenfoot programming environment," *TOCE*, vol. 10, p. 14, 11 2010.

[5] E. N. Fouh Mbindi, *Building and evaluating a learning environment for data structures and algorithms courses*. PhD thesis, Virginia Tech, http://hdl.handle.net/10919/51951, Apr 2015.

[6] D. Nicholas, I. Rowlands, and H. R. Jamali, "E-textbook use, information seeking behaviour and its impact: Case study business and management," *J. Information Science*, vol. 36, pp. 263–280, 03 2010.

[7] E. Fouh, V. Karavirta, D. A. Breakiron, S. Hamouda, S. Hall, T. L. Naps, and C. A. Shaffer, "Design and architecture of an interactive etextbook – the opendsa system," *Science of Computer Programming*, vol. 88, pp. 22–40, 2014. Software Development Concerns in the e-Learning Domain.

[8] OpenDSA, "Opendsa." https://opendsa-server.cs.vt.edu/.

[9] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. A. Velázquez-Iturbide, "Exploring

the role of visualization and engagement in computer science education," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '02, (New York, NY, USA), p. 131–152, Association for Computing Machinery, 2002.

[10] A. Alharbi, F. Henskens, and M. Hannaford, "Integrated standard environment for the teaching and learning of operating systems algorithms using visualizations," in *2010 Fifth International Multi-conference on Computing in the Global Information Technology*, pp. 205–208, 2010.

[11] M. F. Farghally, K. H. Koh, H. Shahin, and C. A. Shaffer, "Evaluating the effectiveness of algorithm analysis visualizations," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, (New York, NY, USA), p. 201–206, Association for Computing Machinery, 2017.

[12] L. MALMI, V. KARAVIRTA, A. KORHONEN, J. NIKANDER, O. SEPPÄLÄ, and P. SILVASTI, "Visual algorithm simulation exercise system with automatic assessment: Trakla2," *Informatics in Education*, vol. 3, no. 2, pp. 267–288, 2004.

[13] M. Mohammed, C. A. Shaffer, and S. H. Rodger, *Teaching Formal Languages with Visualizations and Auto-Graded Exercises*, p. 569–575. New York, NY, USA: Association for Computing Machinery, 2021.

[14] N. Tax, N. Sidorova, R. Haakma, and W. M. van der Aalst, "Mining local process models," *Journal of Innovation in Digital Ecosystems*, vol. 3, no. 2, pp. 183–196, 2016.

[15] "Prom tools." https://www.promtools.org/doku.php.

[16] A. S. Carter, C. D. Hundhausen, and O. Adesope, "The normalized programming state model: Predicting student performance in computing courses based on programming

behavior," in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, (New York, NY, USA), p. 141–150, Association for Computing Machinery, 2015.

[17] A. S. Carter and C. D. Hundhausen, "Using programming process data to detect differences in students' patterns of programming," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, (New York, NY, USA), p. 105–110, Association for Computing Machinery, 2017.

[18] M. D. Byrne, R. Catrambone, and J. T. Stasko, "Evaluating animations as student aids in learning computer algorithms," *Comput. Educ.*, vol. 33, p. 253–278, dec 1999.

[19] E. Fouh, D. A. Breakiron, S. Hamouda, M. F. Farghally, and C. A. Shaffer, "Exploring students learning behavior with an interactive etextbook in computer science courses," *Computers in Human Behavior*, vol. 41, pp. 478–485, 2014.

[20] R. Baker, A. Corbett, I. Roll, K. Koedinger, V. Aleven, E. Haig, A. Hershkovitz, A. Caravalho, A. Mitrovic, and M. Mathews, *Modeling and Studying Gaming the System with Educational Data Mining*, vol. 28, pp. 97–115. 03 2013.

[21] R. S. Baker, A. T. Corbett, K. R. Koedinger, and A. Z. Wagner, "Off-task behavior in the cognitive tutor classroom: When students "game the system"," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, (New York, NY, USA), p. 383–390, Association for Computing Machinery, 2004.

[22] R. Baker, J. Walonoski, N. Heffernan, I. Roll, A. Corbett, and K. Koedinger, "Why students engage in "gaming the system" behavior in interactive learning environments," *Journal of Interactive Learning Research*, vol. 19, pp. 185–224, April 2008.

[23] D. A. Breakiron, "Evaluating the integration of online, interactive tutorials

into a data structures and algorithms course," Master's thesis, Virginia Tech, http://hdl.handle.net/10919/23107, May 2013.

[24] U.S. Department of Education, Office of Educational Technology, *Enhancing teaching and learning through educational data mining and learning analytics: An issue brief*. Washington, D.C, 2012.

[25] C. Watson, F. W. Li, and J. L. Godwin, "Predicting performance in an introductory programming course by logging and analyzing student programming behavior," in *2013 IEEE 13th International Conference on Advanced Learning Technologies*, pp. 319–323, 2013.

[26] E. S. Tabanao, M. M. T. Rodrigo, and M. C. Jadud, "Predicting at-risk novice java programmers through the analysis of online protocols," in *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, (New York, NY, USA), p. 85–92, Association for Computing Machinery, 2011.

[27] R. S. J. d. Baker, E. Duval, J. Stamper, D. Wiley, and S. B. Shum, "Educational data mining meets learning analytics," in *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, LAK '12, (New York, NY, USA), p. 20, Association for Computing Machinery, 2012.

[28] M. Mohammed, C. A. Shaffer, and S. H. Rodger, "Teaching formal languages with visualizations and auto-graded exercises," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, (New York, NY, USA), p. 569–575, Association for Computing Machinery, 2021.

[29] OpenDSA, "Opendsa system documentation." https://opendsa.readthedocs.io/en/latest/.

[30] R. Baker, A. Corbett, and K. Koedinger, "Detecting student misuse of intelligent tutoring systems," vol. 3220, pp. 531–540, 08 2004.

[31] S. H. Edwards and K. P. Murali, "Codeworkout: Short programming exercises with built-in data collection," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, (New York, NY, USA), p. 188–193, Association for Computing Machinery, 2017.