The Journal of Computing Sciences in Colleges

Papers of the 15th Annual CCSC Southwestern Conference

March 26th, 2022
University of California – Irvine
Irvine, CA

Baochuan Lu, Editor
Southwest Baptist University

Mariam Salloum, Regional Editor
UC Riverside

Volume 37, Number 10 **April 2022**

The Journal of Computing Sciences in Colleges (ISSN 1937-4771 print, 1937-4763 digital) is published at least six times per year and constitutes the refereed papers of regional conferences sponsored by the Consortium for Computing Sciences in Colleges.

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

Following is a listing of the contact information for the members of the Board of Directors and the Officers of the Consortium for Computing Sciences in Colleges (along with the years of expiration of their terms), as well as members serving CCSC:

Karina Assiter, President (2022), (802)387-7112, karinaassiter@landmark.edu.

Chris Healy, Vice President (2022), chris.healy@furman.edu, Computer Science Department, 3300 Poinsett Highway Greenville, SC 29613.

Baochuan Lu, Publications Chair (2024), (417)328-1676, blu@sbuniv.edu, Southwest Baptist University - Division of Computing & Mathematics, 1600 University Ave., Bolivar, MO 65613.

Brian Hare, Treasurer (2023), (816)235-2362, hareb@umkc.edu, University of Missouri-Kansas City, School of Computing & Engineering, 450E Flarsheim Hall, 5110 Rockhill Rd., Kansas City MO 64110.

Cathy Bareiss, Membership Secretary (2022), cathy.bareiss@betheluniversity.edu, Department of Mathematical Engineering Sciences, 1001 Bethel Circle, Mishawaka, IN 46545.

Judy Mullins, Central Plains Representative (2023), Associate Treasurer, (816)390-4386, mullinsj@umkc.edu, UMKC, Retired.

Michael Flinn, Eastern Representative (2023), mflinn@frostburg.edu, Department of Computer Science Information Technologies, Frostburg State University, 101 Braddock Road, Frostburg, MD 21532.

David R. Naugler, Midsouth Representative(2022), (317) 456-2125, dnaugler@semo.edu, 5293 Green Hills Drive, Brownsburg IN 46112.

Grace Mirsky, Midwest Representative(2023), gmirsky@ben.edu, Mathematical and Computational Sciences, 5700 College Rd. Lisle, IL 60532.

Lawrence D'Antonio, Northeastern Representative (2022), (201)684-7714, ldant@ramapo.edu, Computer Science Department, Ramapo College of New Jersey, Mahwah, NJ 07430.

Shereen Khoja, Northwestern Representative(2024), shereen@pacificu.edu, Computer Science, 2043 College Way, Forest Grove, OR 97116.

Mohamed Lotfy, Rocky Mountain Representative (2022), Information Systems & Technology Department, College of Engineering & Technology, Utah Valley University, Orem, UT 84058.

Tina Johnson, South Central Representative (2024), (940)397-6201, tina.johnson@mwsu.edu, Dept. of Computer Science, Midwestern State University, 3410 Taft Boulevard, Wichita Falls, TX 76308.

Kevin Treu, Southeastern Representative (2024), (864)294-3220, kevin.treu@furman.edu, Furman University, Dept of Computer Science, Greenville, SC 29613.

Bryan Dixon, Southwestern Representative (2023), (530)898-4864, bcdixon@csuchico.edu, Computer

Science Department, California State University, Chico, Chico, CA 95929-0410.

Serving the CCSC: These members are serving in positions as indicated:
Bin Peng, Associate Editor, (816)584-6884, bin.peng@park.edu, Park University - Department of Computer Science and Information Systems, 8700 NW River Park Drive, Parkville, MO 64152.
George Dimitoglou, Comptroller, (301)696-3980, dimitoglou@hood.edu, Dept. of Computer Science, Hood college, 401 Rosemont Ave. Frederick, MD 21701.
Carol Spradling, National Partners Chair, (660)863-9481, carol.spradling@gmail.com, 760 W 46th St, Apt 208, Kansas City, MO 64112.
Megan Thomas, Membership System Administrator, (209)667-3584, mthomas@cs.csustan.edu, Dept. of Computer Science, CSU Stanislaus, One University Circle, Turlock, CA 95382.
Ed Lindoo, Associate Treasurer & UPE Liaison, (303)964-6385, elindoo@regis.edu, Anderson College of Business and Computing, Regis University, 3333 Regis Boulevard, Denver, CO 80221.

CCSC National Partners

The Consortium is very happy to have the following as National Partners. If you have the opportunity please thank them for their support of computing in teaching institutions. As National Partners they are invited to participate in our regional conferences. Visit with their representatives there.

Platinum Partner

Google Cloud
GitHub
NSF – National Science Foundation

Silver Partners

zyBooks

Bronze Partners

Mercury Learning and Information
Mercy College

A Scalable Approach for Detecting Exam Similarity in CS Courses*

Niema Moshiri
Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093
niema@ucsd.edu

Abstract

During the COVID-19 pandemic, CS courses at many institutions faced a sudden shift to fully-online instruction. Despite this shift in modality, many courses continued to administer single-student exams to assess student learning, yet these online exams typically lacked the ability to proctor students to enforce rules against collaboration. We describe a scalable approach for detecting exam similarity in online exams. The approach involves an Exam Similarity Score we define, and it includes empirical and theoretical approaches for gauging significance in the distribution of pairwise similarity scores across the class. We then present our experiences from utilizing the approach in an Advanced Data Structures course across two quarters, with more than 400 students in each offering of the course. We show that the approach was able to successfully detect cases of exam collaboration without significant investigative effort from the course instructional staff. An open source Python implementation of the approach can be found on GitHub: https://github.com/niemasd/MESS

1 Introduction

As COVID-19 spread rapidly throughout the world, the majority of universities suddenly shifted to fully remote course instruction. As a result of limited time to adapt, many instructors continued to administer single-student exams reminiscent of those utilized in-person. To combat cheating, many universities employed remote proctoring services, but student reception to such platforms was mixed, with some students comfortable with the surveillance and others feeling as though their privacy had been infringed upon [5].

The following question naturally arises: Can instructors detect exam collaboration purely from student responses? An intuitive approach is to compare the exams of all pairs of students and explore instances of surprisingly high similarity in responses: students who collaborated may have a higher-than-expected proportion of identical responses. Further, similarities in correct responses are expected, but excessive similarities in *incorrect* responses may be more telling.

In this paper, we describe a scalable approach for detecting exam similarity in online exams, which involves an Exam Similarity Score we define. We include empirical and theoretical approaches for gauging significance in the distribution of pairwise similarity scores across a class. We then present our experiences from utilizing the approach in an Advanced Data Structures course across two quarters, with over 400 students enrolled in each offering of the course. We show that the approach was able to successfully detect cases of exam collaboration without significant investigative effort from the course instructional staff, relying solely on exam responses submitted by the students.

2 Related Work

Lin and Levitt proposed a scalable algorithmic approach for detecting highly similar exams by modeling a student's answer on a particular question using a multinomial logit and computing the theoretical expected number of matching correct and incorrect responses [4]. When the authors applied their algorithm to exams in a general science course, they found strong evidence of cheating by a large proportion of students in the class, and they concluded that students studying together could not explain their findings. Further, they found that matching incorrect answers proved to be a stronger indicator than matching correct answers. However, their approach was limited to multiple choice questions with at most four options, whereas in CS courses, exam questions typically rely heavily on problem-solving with open-ended responses (e.g. mathematical or short-answer questions). Further, the authors only *describe* the approach: they do not provide a tool that *implements* the approach for general use.

Our work expands upon this approach by generalizing to any arbitrary

question type for which instructors can compare responses for equivalence. This benefits instructors in terms of exam assessment design as well as reduces the chance of spurious identical incorrect student responses on non-multiple-choice questions (which may improve signal of similarity detection). Further, our work provides a statistical test for determining significantly high similarity scores, with multiple user-selectable choices for multiple hypothesis test correction (e.g. Bonferroni [2] or Benjamini-Hochberg [1]).

3 Methods

3.1 Exam Similarity Score (ESS)

Let Q denote the set of all questions on the exam, and let $|Q|$ denote the total number of questions on the exam. Let $w(q)$ denote the number of students who submitted an incorrect response for a given question $q \in Q$. For a given pair of students x and y who submitted the *same* incorrect response for a given question $q \in Q$, let $d(q, x, y)$ denote the number of students who submitted a *different* incorrect response for q. Let $p(q, x, y) = d(q, x, y)/w(q)$ if students x and y submitted the same incorrect response for question $q \in Q$, otherwise $p(q, x, y) = 0$ (e.g. if x or y submitted correct responses, or if x and y submitted different incorrect responses). We define the Exam Similarity Score (ESS) between students x and y as follows:

$$S(x, y) = \frac{\sum_{q \in Q} p(q, x, y)}{|Q|} \tag{1}$$

The ESS has a minimum of 0, which implies no identical incorrect responses, and a maximum that approaches 1, which implies identical and unique incorrect responses on all exam questions.

3.2 Similarity Significance Detection

The ESS motivates a simple systematic approach for detecting exam similarity: compute the ESS across all pairs of students, sort pairs of students in descending order of ESS, and select pairs of students with statistically significantly large scores to investigate further as suspected cases of potential collaboration. However, a critical question arises: how does one determine an ESS threshold above which scores will be deemed as "significantly large"?

In a hypothetical course with infinite students, assuming that the vast majority of the possible pairs of students did *not* collaborate on an exam, we would expect the distribution of ESSs computed across all pairs of students in the class to fit closely to the null distribution, with pairs of students who *did* collaborate appearing as outliers with larger ESSs than one would expect.

As the number of students decreases, the fit would worsen, especially at the ends of the distribution due to reduced sampling. We explored the empirical distributions of ESSs for the Midterm and Final Exams across two offerings of an Advanced Data Structures course, both with over 400 enrolled students. As expected, when the distributions are plotted in log-scale, the ends of the distributions are noisy, but there is a consistent close-to-linear stretch for central values of the ESS distribution (Fig. 1, dashed).

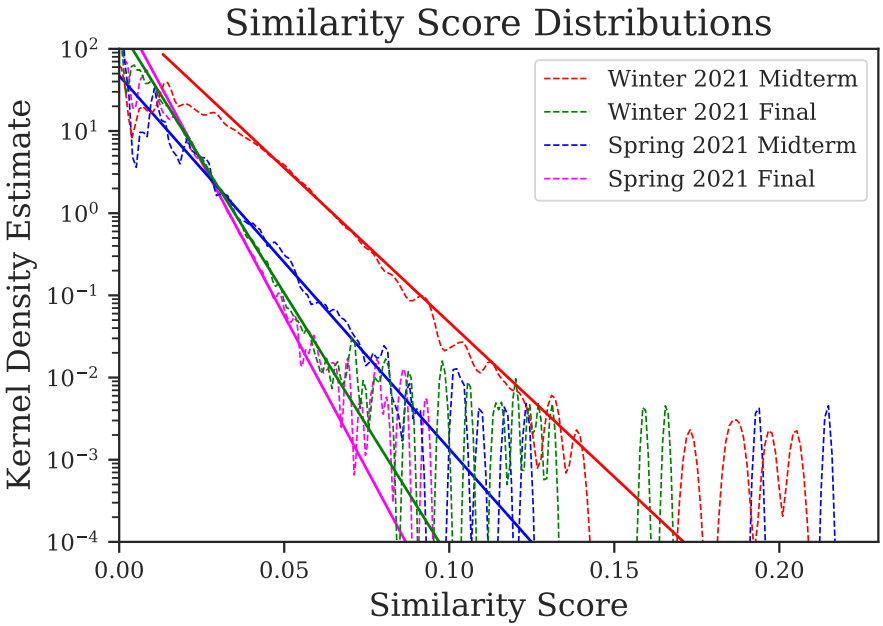

Figure 1: Kernel Density Estimates (KDEs) of Exam Similarity Scores (dashed), and Probability Density Functions (PDFs) of best-fit Exponential distributions (solid).

The Probability Density Function (PDF) of an Exponential distribution with rate parameter λ and location parameter μ is the following:

$$f_X(x) = \lambda e^{-\lambda(x-\mu)} \tag{2}$$

Therefore, the log of the PDF of an Exponential distribution with rate parameter λ and location parameter μ is the following:

$$\log\left(f_X(x)\right) = \log\left(\lambda e^{-\lambda(x-\mu)}\right) = \log\left(\lambda\right) - \lambda\left(x-\mu\right) \tag{3}$$

Thus, given a line $y = mx + b$ regressed from the log of the KDE (log-KDE) of samples from an unknown Exponential distribution, the parameters of the Exponential can be estimated as follows:

$$\lambda = -m \quad \text{and} \quad \mu = \frac{\log(\lambda) - b}{m} \tag{4}$$

This motivates a simple approach for computing theoretical p-values for ESS values computed from all pairs of students:

1. Compute the KDE of the distribution of ESSs
2. Regress a line from the near-linear segment of the log-KDE
3. Estimate the Exponential parameters from the line
4. Compute p-values from the Exponential distribution

The statistical test is one-sided: specifically, the p-value associated with a given ESS x is the probability of observing an ESS greater than or equal to x purely by chance. Therefore, the p-value for a given ESS x is simply the area under the Cumulative Distribution Function (CDF) of Exponential(λ, μ) for the range $X \geq x$.

When we compute a p-value for every possible pair of students and check each p-value for statistical significance, we are performing multiple simultaneous hypothesis tests. To control the False Discovery Rate (FDR), we can perform a correction (e.g. Bonferroni [2] or Benjamini-Hochberg [1]) to compute an FDR-controlled adjusted p-value, also known as a q-value. The resulting q-values can be compared against a statistical significance threshold, e.g. $q \leq 0.05$, to provide an automated similarity detection algorithm.

3.3 Tool Implementation

We have implemented the exam similarity significance detection approach, including multiple hypothesis test correction in the form of q-value calculation, as an open source cross-platform Python tool that depends on NumPy [3], SciPy [6], and seaborn [7]. The tool is available on GitHub: `https://github.com/niemasd/MESS`

The tool takes as input a spreadsheet containing the student responses in the TSV (tab-delimited) format. The top row is a header row, and each subsequent row of the spreadsheet corresponds to a single student. The leftmost column should contain a unique student identifier (e.g. email address, student ID, etc.), the second leftmost column should contain a comma-separated list of questions the student answered correctly, and each remaining column should

correspond to a single exam question: cell (i, j) of the spreadsheet (excluding the two leftmost columns and the top row) should be the response student i submitted for question j. All popular platforms for recording exam responses (e.g. course LMS, Google Form, etc.) are supported by the tool as long as spreadsheets are downloaded in or converted to the described TSV format. We will provide helper conversion scripts for popular exam platforms. Note that the tool does not distinguish between questions and sub-questions: each prompt for which the student submits a single response is considered to be a single "question" by the tool.

The tool outputs a spreadsheet in which each row represents a single pair of students and which has the following columns: (1) student 1 of the pair, (2) student 2 of the pair, (3) the ESS computed from the pair, (4) the p-value of the ESS, and (5) the q-value obtained by adjusting the p-value for multiple hypothesis test correction. The tool also outputs a figure containing the KDE of the ESS distribution along with the PDF of the best-fit Exponential distribution, as depicted in Figure 1.

4 Results

We utilized the described Similarity Significance Detection approach to detect collaboration in the Midterm and Final Exams administered in the Winter and Spring 2021 offerings of an Advanced Data Structures course, with each course offering having over 400 enrolled students. As can be seen in Figure 1, the PDF of the Exponential distributions estimated from the near-linear segments of each distribution easily distinguish the bulk of the ESS distribution from visually clear outliers.

Further, when using the Benjamini-Hochberg approach for multiple hypothesis test correction [1], the similarity detection algorithm did not yield any False Positives: every pair of students that had a corrected q-value below a threshold of 0.05 was ultimately found responsible for violating Academic Integrity on the respective exam by the university's Academic Integrity Office. However, the automated similarity detection algorithm yielded a few False Negatives, i.e., pairs of students whose ESS values, despite visually appearing to be outliers and ultimately being found responsible by the Academic Integrity Office, did not pass our filtering criterion of $q \leq 0.05$.

Ultimately, using a combination of the automated q-value threshold algorithm and manual visual inspection of the ESS distribution, we were able to detect 14 and 12 pairs of students who collaborated on at least one exam in the Winter and Spring 2021 offerings of the course, respectively. Importantly, this required less than one hour of manual inspection of students submissions from the course instructor for each exam.

5 Discussion

We found that the exam similarity detection approach described in this manuscript worked quite well: in Advanced Data Structures classes with over 400 students (and thus over 80,000 possible pairwise comparisons), the approach was able to quickly (and, with no False Positives and 5–10 False Negatives, automatically) detect exam collaboration purely from similarities in student responses to exam questions. Importantly, this approach required very little manual labor from the course instructional staff: the Python tool bubbled up pairs of students with surprisingly large ESS values, and manual inspection of the complete exams of just those few pairs led to discovery of collaboration.

Importantly, the approach leverages itself nicely to the open-ended problem-solving nature of problems typically seen in CS exams: by designing questions in which the range of possible incorrect responses is quite large, any identical incorrect responses submitted by a pair of collaborating students will be reasonably unique with respect to the class (at least more so than in the case of multiple choice questions), which will yield even larger ESS values.

A key limitation of this approach is that it assumes that an Exponential distribution is an appropriate model for the distribution of ESS values, but as can be seen in Figure 1, even the left end of the true distribution deviates from the theoretical PDF of the estimated Exponential. Further, for the bulk of the distribution, the PDF of the estimated Exponential is actually *above* the true distribution, meaning the estimated p-value (and corrected q-value) for a given ESS value is actually an *over*-estimate. This likely explains the False Negatives when using a filtering criterion of $q \leq 0.05$. This limitation can be improved upon in multiple ways in future work. First, the definition of the ESS was largely motivated by intuition, yet simple modifications to or extensions of the ESS may yield a probability distribution that better fits an Exponential distribution. Further, the Exponential distribution was chosen because its PDF is a reasonably close fit to the observed KDE of ESS values and is quite simple to estimate from data, but a more complex yet better-fitting probability distribution can be proposed to define the distribution of ESS (or an extension/modification of ESS) under the null hypothesis. Lastly, the proposed approaches of estimating the rate parameter λ and the location parameter μ were selected due to their simplicity, but the estimation of the location parameter μ can be modified to yield slightly reduced values (i.e., the PDF would shift downwards), and/or the the estimation of the rate parameter λ can be modified to be slightly increased (i.e., the PDF would be steepened): the result would be a PDF that doesn't quite match the linear portion of the empirical KDE as nicely as with the simple estimation approach, but that may better capture the right tail of the distribution, which is the only region of interest when detecting exam similarity significance.

Further, the ESS ignores instances in which a pair of students submitted correct responses. However, in CS exams, it is common to include open-ended questions that may have multiple possible correct answers. For example, in an Advanced Data Structures course, one might ask the student to provide a single element that, when inserted into a given AVL Tree, will result in a specific number of AVL rotations. Such questions are not only good pedagogy in terms of testing students' fundamental understanding of the concepts, but the variation in possible student responses may provide insights into potential collaboration: if two students have identical correct responses that are reasonably unique compared to the rest of the class on multiple questions on an exam, they may have collaborated. One could imagine an extension of the ESS that accounts for uniquely identical correct responses as well.

Lastly, the ESS only checks for response equivalence, but one could imagine an approach that utilizes a distance metric (e.g. Hamming or Edit Distance for text, or Euclidean Distance for numerical). However, it is likely that appropriate distance metrics will be unique to individual questions based on the space of theoretically possible reasonable responses to the question, so it is unclear if such an approach would be generalizable to arbitrary exams.

6 Conclusion

In this paper, we defined an Exam Similarity Score (ESS) and described a scalable approach for detecting exam similarity in online exams using the ESS. We then presented our experiences from utilizing the approach in an Advanced Data Structures course across two quarters, with more than 400 students in each offering of the course. Our work shows that it is possible to detect collaboration on exams solely from student-submitted responses, and we believe that this work and any downstream works expanding upon it can improve the detection of academic dishonesty in online exams without the need for remote proctoring services or other forms of real-time student monitoring.

References

[1] Yoav Benjamini and Yosef Hochberg. "Controlling the False Discovery Rate: a Practical and Powerful Approach to Multiple Testing". In: *J. R. Statist. Soc. B* 57.I (1995), pp. 289–300.

[2] Jelle J. Goeman and Aldo Solari. "Multiple hypothesis testing in genomics". In: *Statistics in Medicine* 33.11 (Jan. 2014), pp. 1946–1978. ISSN: 10970258. DOI: 10.1002/sim.6082.

[3] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585 (Sept. 2020), pp. 357–362. ISSN: 14764687. DOI: 10.1038/s41586-020-2649-2. arXiv: 2006.10256.

[4] Ming-Jen Lin and Steven D. Levitt. "Catching Cheating Students". In: *Economica* 87.348 (Oct. 2020), pp. 885–900. ISSN: 14680335. DOI: 10.1111/ecca.12331.

[5] Neil Selwyn et al. "A necessary evil? The rise of online exam proctoring in Australian universities". In: *Media International Australia* (Apr. 2021), pp. 1–16. ISSN: 1329878X. DOI: 10.1177/1329878X211005862.

[6] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17 (Feb. 2020), pp. 261–272. ISSN: 15487105. DOI: 10.1038/s41592-019-0686-2. arXiv: 1907.10121.

[7] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

Cohesive and Backward-Designed CS1 Programming Assessments for Better Student Engagement*

Carolyn Pe Rosiene[1] and Joel A. Rosiene[2]
[1]Department of Computing Sciences
University of Hartford
W. Hartford, CT 06117
rosiene@hartford.edu
[2]Department of Computer Science
Eastern Connecticut State University
Windham, CT 06226
rosienej@easternct.edu

Abstract

The ultimate goal of this work is to increase student engagement in a CS1 course. This is done by creating incremental and related labs and programming assignments via backward design. The approach results in a cohesive series of assessments where students do not need to relearn the application domain each and every time. A new feature for the final product is incrementally introduced based on the relevant topic covered in class. A survey taken at the end of the semester shows that students were receptive to this approach; over 90% of the students indicated favoring this approach.

1 Introduction

The goal of this work is to develop a series of programming assessments that are related but where each is unique in exercising a different topic in a CS1 course. We apply backward design [1, 2] by identifying the goal of the final product, focusing on a particular topic according to course coverage, and designing separate programming assessments administered throughout the semester making sure that each contribute to the final product. In CS, many approaches have been taken to increase student engagement including using real-world assignments [3], flipped classrooms [4], and blended learning [5]. The traditional approach to designing and planning units of instruction has been: (1) identify a topic or chunk of content that needs to be covered; (2) plan a sequence of lessons to teach that content; and (3) create an assessment to measure the learning that should have taken place in those lessons. In this traditional approach, the assessment (labs or programming assignments) are an afterthought; planned after the lessons have been created and most likely delivered. This innovation reverses the procedure by (1) first deciding and identifying what students should know at the end of the semester; (2) then assessments (in the form of labs and programming assignments) are created to measure that learning; and lastly, (3) lessons are planned to successfully complete each of the programming assignments.

It is evident that in order for this to be orchestrated well, careful planning on the sequence of labs and programming assignments is required. The individual tasks each need to be self-contained but also contribute to the final product.

2 Motivation and Background

The motivation for trying the aforementioned approach is to increase student engagement in a freshman-level programming course. As CS educators, we know that student success hinges heavily on course delivery techniques; the other contributing factor is the student's own interest and personal motivation. To help the former aspect, we, as concerned educators, look for different ways to improve our pedagogy. Developing practices to help student engagement is crucial, especially for freshman-level courses, because it is usually the time during a student's career in college when they decide to stay in the major or switch majors. In creating interesting lessons and assessments throughout the semester, we hope to pique the student's interest, and make them enthusiastic about the field they have chosen.

CS 115, Fundamentals of Computing II, is a follow-on course to CS 114, Fundamentals of Computing I. In CS 115, we build on top of the basic programming essentials (variables, loops, conditionals, classes, and arrays) by delving

into attributes of object-oriented programming and more advanced constructs (inheritance, abstract classes, polymorphism, recursion, exceptions). In CS 114 and CS 115, deliverables typically include several programming assignments, projects, or assessments. Usually, these are disparate and unrelated projects exercising different programming fundamentals. The goal is to unify these assessments and form a cohesive and unifying theme to tie different programming assignments throughout the semester so that students do not need to be introduced to a new domain (new game, new requirements, new perspectives, etc.) every week or so. This promotes student engagement in that they are already familiar with the domain (what the problem is) and only need to focus on how to approach the problem (how to solve the problem).

Following Fox and Doherty's work [6] where they clearly specify student learning outcomes; create meaningful assessments; and make sure of strong project management, we start by listing our learning outcomes for CS 115:(1) Exercise algorithmic thinking; (2) Declare and perform fundamental operations on a two-dimensional array; (3) Understand the characteristics of an object-oriented programming language; (4) Apply the object-oriented design methodology to solve a problem and to take an object-oriented design and code it in Java; (5) Understand and use interfaces, polymorphism and inheritance; (6) Implement exception handling, write code that throws an exception, and write an exception handler; and (7) Implement file I/O.

CS 115 is a second course in a series of a three-course sequence, CS 114, CS 115 and CS 220 (Data Structures). This innovation ultimately affects those students moving into the next course, CS 220, with better retention and students becoming more excited about the field they have chosen. This approach was implemented in Spring 2021 with 12 students responding (out of the 13 enrolled) to the end-of-semester survey in CS 115. A summary of data collected is presented.

3 Backward-Designed Strategy and Assessments

To accomplish the curriculum planning, we chose to use the game Nonogram [7] (also known as Picture Crossword or Picross, among others) as the general theme for the semester. At the end of the semester, students complete a robust program that allows a user to play a Nonogram puzzle (complete with a graphical user interface) using all the building blocks collected throughout the semester via programming assessments they have already completed.

In the Spring 2021 semester, we implemented a series of programming laboratories and assignments to innovate the pedagogy in CS 115. This was a standard 14-week course which met twice a week. Course content is as prescribed by the department and covered more advanced object-oriented tech-

niques, along with some non-object-oriented-focused constructs like sorting, searching, and file I/O using Java as the base programming language.

Students' assessments for the course included labs, assignments, quizzes, and tests. Their final product uses bits and pieces of code assigned throughout the semester as labs or other assignments. Labs are meant to be completed while in class with a partner to exercise pair programming. They do not account too much of the student's final grade because of the nature of the work assigned. However, two of the five labs assigned will help them complete the final program (Assignment 4). Quizzes are take home and given frequently. There were short assessment reviewing the lesson of the day. Thirteen quizzes were administered all together. Assignments are long-term work, assigned to be completely individually. Four assignments were distributed throughout the semester with the last one as the culminating task to wrap up the Nonogram implementation. Tests are hour-long, in-class assessments to make sure that students determine whether students have learned what they were expected to learn.

Out of the 6 labs and 4 assignments, 6 of these (2 labs and 4 assignments) were assessments that contribute to this backward-designed approach (refer to Table 3). Certain learning goals did not lend themselves directly to the development of the game, so these exceptions were given as labs (remaining 4 labs).

Table 1: CS 115 Assessments Based on Nonogram

Lab/Assignment	Week #	Goals/Tasks
Assignment 1	2	Develop clues for each row and each column
Lab 2	5	Create puzzle interface
Assignment 2	7	Present the puzzle solution using GUI
Assignment 3	9	Read puzzle solutions from a file
Lab 5	12	Recursively read file folders
Assignment 4	13	Implement a playable game

4 Student Feedback

In order to measure the effectiveness of the backward-designed assessment approach, three additional 5-point Likert-scale items were added to end-of-semester course evaluation surveys:

1. Nonogram theme: Using a common theme (Nonogram) helped me think about each individual lab or assignment because the puzzle is familiar after the first introduction.

2. Additional Nonogram Features: I plan to add more features to my final Nonogram project on my own.

3. Nonogram in resume: I am proud of my Nonogram project and will include this in my resume when looking for an internship or a job in the future.

The first question evaluates students' perspective of the cohesiveness of the labs and assignments, while the last two questions evaluates students' future intent and, hopefully, gives an idea of their engagement albeit most students think the game is lame and is not something an 18- or 19-year old might prefer to spend time playing. 92% responded, 12 out of 13 student.

As one can see on Figure 1, an overwhelming number of students reacted positively to using the same theme over and over throughout the course of the semester; 11 out of 12 students agreed or strongly agreed to the use of the same domain. The second question which asks whether students would spend their summer or free time working on the same project by adding their own features did not receive as good feedback; 3 out of 12 students disagreed or strongly disagreed. This was to be expected since most students view this as an academic endeavor and the choice of type of game might not be their cup of tea. Other reasons might be that students already have made plans for their summer break. The last question asked whether students would be proud enough to advertise this work in their resume; surprisingly enough, 10 out of 12 students agreed or strongly agreed. This is a strong indication that students are satisfied and proud of their work so much so that it deserves to be mentioned in their resumes.

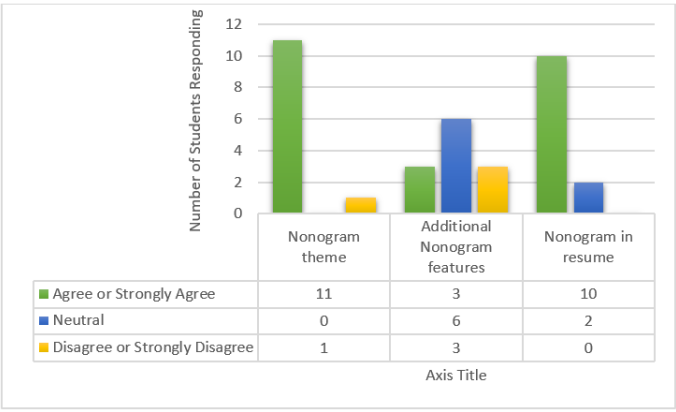

Figure 1: Nonogram-specific CS 115 Survey Results

5 Conclusion and Future Work

A common thread to most, if not several, programming assessments appear to have an impact on student engagement. This theme-based series of assignments and labs was born out of a backward-designed approach to create lessons and assessments. From our experience, students were receptive to having the same domain appear several times throughout the semester, helping create a common thread to their labs and assignments.

References

[1] Grant Wiggins and Jay McTighe. What is backward design. *Understanding by design*, 1:7–19, 1998.

[2] Mary Whitehouse. Using a backward design approach to embed assessment in teaching. *School Science Review*, 95(352):99–104, 2014.

[3] Daniel E. Stevenson and Paul J. Wagner. Developing real-world programming assignments for cs1. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITICSE '06, page 158–162, New York, NY, USA, 2006. Association for Computing Machinery.

[4] Gina Sprint and Erik Fox. *Improving Student Study Choices in CS1 with Gamification and Flipped Classrooms*, page 773–779. Association for Computing Machinery, New York, NY, USA, 2020.

[5] Lilin Gong, Yazhao Liu, and Wei Zhao. Using learning analytics to promote student engagement and achievement in blended learning: An empirical study. In *Proceedings of the 2nd International Conference on E-Education, E-Business and E-Technology*, ICEBT 2018, page 19–24, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Bruce E Fox and John J Doherty. Design to learn, learn to design: Using backward design for information literacy instruction. *Communications in Information Literacy*, 5(2):7, 2012.

[7] Daniel Berend, Dolev Pomeranz, Ronen Rabani, and Ben Raziel. Nonograms: Combinatorial questions and algorithms. *Discrete Applied Mathematics*, 169:30–42, 2014.

Beyond Big O: Teaching Experimental Algorithmics*

Michael Shindler Michael T. Goodrich Ofek Gila
Michael Dillencourt
University of California, Irvine
{mikes, goodrich, ogila, mbdillen}@uci.edu

Abstract

We present a supplement to traditionally-taught topics with experimental explorations of algorithms.

1 Introduction

Algorithms courses are traditionally taught with an emphasis on general design techniques (like divide-and-conquer and dynamic programming) and the formal analysis of algorithms, e.g., see [8, 10, 14]. This is a valuable part of Computer Science education, which we are not proposing replacing. Instead, we are proposing here how one might supplement traditional algorithms instruction with projects or even an entire course in *experimental algorithmics*.[1]

Experimental algorithmics [18] studies the design, implementation, and experimental evaluation of algorithms and data structures. This topic has its own journal (JEA), which began in 1996 [19], its own U.S. conference (ALENEX), which began in 1999 [9], and its own European conference (SEA), which began in 2003 [11]. Experimental algorithmics can provide insights into the performance of algorithms that go beyond formal analysis, to address issues such

[1]See `https://www.ics.uci.edu/~goodrich/teach/cs165/` for a syllabus, schedule, lecture slides, readings, and detailed projects for a recent offering of our course.

as "real world" running times, the size of constant factors, and how well an algorithm achieves various optimization goals. Algorithm engineering and experimentation is also a useful skill for practicing computer scientists, as it can lead to better theoretical analysis and it can also demonstrate where such analysis is misleading, e.g., see [25]. Further, students using formal analysis alone may not fully appreciate how asymptotic running times manifest themselves in the real-world performance of algorithms. We describe in this paper methodologies for supplementing traditional algorithms instruction with modules or even an entire course in experimental algorithmics.

Our methodologies and modules for experimental algorithmics address the issues of discovering an algorithm's real-world running time and how well algorithms achieve optimization goals. Our course also covers the use of algorithms to test models of the real world; this last question is covered in the appendix.

A common and recurrent theme throughout all of our projects is the use of log-log plots as an analysis tool; hence, a side benefit of our projects is that students gain a deeper understanding and appreciation for log-log plots, both in general and in the experimental analysis of algorithms.

1.1 Related Work

Experimental algorithmics is a research field that is too deep to review here. In terms of general background, we refer the interested reader to the textbook by McGeoch [18] or the guide by Johnson [12], as well as past proceedings for ALENEX and SEA, or past issues of JEA. For a review of the related area of *algorithm engineering*, please see [24].

There has also been previous work on integrating experimental analysis. Ángel Velázquez-Iturbide and Debdi [1] describe a set of projects for experimenting with greedy algorithms, but they do not address their asymptotic analysis using best-fit functions. Berque *et al.* [5] describe a workbench, which they call KLYDE, for experimental algorithm analysis. Their system includes a tool for asymptotic-time analysis but does not include best-fit asymptotic functions or the analysis of other performance variables. Matocha [17] describes a set of projects that emphasize technical writing and the scientific method while also addressing experimental algorithm analysis but does not stress best-fit functions for analysis purposes. Sanders [23] describes his experiences in teaching empirical analysis of algorithms, focusing primarily on running time analysis, but he also does not include best-fit functions for asymptotic running times. In addition, there has also been previous work on introducing experimental algorithm analysis earlier in the Computer Science curriculum, such as for CS1 or CS2 courses. Such courses, however, either have a limited focus on performance (see [2]) or mainly analyze algorithms with different algorithmic complexity experimentally (see [13] and [26]). These courses don't empha-

size the real-world applications of algorithm design as much, e.g., they don't highlight the importance of constant factors, input distributions, heuristic optimization algorithms, etc.

1.2 Our Contributions

We present methodologies that supplement traditionally-taught algorithms topics with experimental explorations of algorithms. Such supplementation can either be as a part of a traditional algorithms course or as an additional project-oriented course. Our course required students to implement a variety of algorithms, run benchmarks, plot their results, compare alternative implementations, and draw conclusions. In particular, we provide projects that address the three distinct questions of algorithm analysis mentioned above.

We feel that the skills learned and practiced in these projects should appeal to a variety of Computer Science students with varying desired career paths.

In this paper, we provide two project types. We report on our experiences about how well students achieved the learning outcomes for the specific projects. These projects can be adjusted for the target audiences at other universities.

2 Project 1: Running Times

In the first type of project, students are asked to implement several sorting algorithms, of which students are expected to have seen only one or two prior to doing this project. For example, Bubble and Insertion sort can be chosen as examples students have seen, and parameterizations of Spin-the-bottle sort and Shellsort can be used as ones the students may not have seen.

All of these algorithms are relatively easy to program—the goal here is not to test students' ability to implement complicated algorithms. Instead, we wish to teach techniques of experimental algorithmics. Students are asked to experiment with each algorithm using ever increasing input sizes and various arrangements. Students are then asked to plot their results on a log-log scale in order to find expected running times and find a best-fit line equation for their data. An example set of charts that were produced empirically for this project are shown in Figure 1. This set of experiments shows an example of a student who empirically determined expected running times close to $O(n^2)$ for bubble sort, insertion sort, and spin-the-bottle sort using uniform random inputs, and close to $O(n^{3/2})$ and $O(n^{6/5})$ for two versions of Shellsort. It also shows that for almost-sorted inputs, insertion sort runs in almost linear time—much faster than its worst-case bound (consistent with its $\Theta(n+I)$ time complexity, where I is the number of inversions)—but bubble sort and spin-the-bottle perform consistent with their respective bounds. Therefore, a student with such data

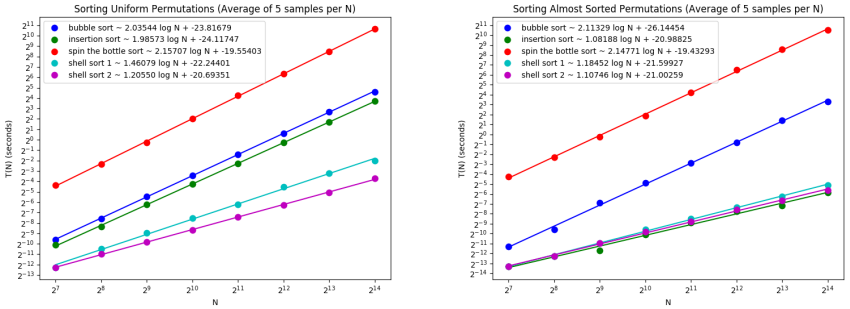

Figure 1: Project 1 charts.

should also learn a lesson as to how input distributions can impact real-world running times.

Further, we feel that this project serves as a good introduction to experimental algorithmics. The students likely know the "correct answer" to at least two of the algorithms (Bubble Sort and Insertion Sort), and this can serve as a "sanity check" for their early work.

Alternative choices and options for this type of project include the use of other sorting algorithms or other input distributions. One could also use a different problem, such as selection, connected components in a graph, minimum spanning trees, or big-integer multiplication. The only requirement is that there be multiple algorithms with varying running times for solving the same problem.

3 Project 2: Optimization

The second type of project deals with empirically testing how well algorithms achieve an optimization goal. From a pedagogical point of view, there are a number of challenges to overcome for this project. An ideal project addresses a non-trivial (NP-hard) optimization problem for which multiple heuristic algorithms exist and are easy to program. Furthermore, the relative quality of solutions must be efficiently computable.

One example is to have students test heuristic algorithms for bin packing, where we are given a set of items with (normalized) sizes between 0 and 1 and asked to pack them into as few bins of size 1 as possible without overflowing any bin [7]. Rather than have the student empirically compare algorithms based on how close they get to the optimal number of bins (which is computationally difficult to determine), we instead recommend using a *waste* parameter, which

is the number of bins used by an algorithm minus the total size (i.e., the sum) of all the items. We had students implement the first-fit and best-fit heuristics, both in an online setting and in the setting in which we can first sort elements in decreasing order of size. Students implemented each and measured the waste produced by each on randomly generated item lists of various sizes.

An outcome chart for a set of such experiments is the following:

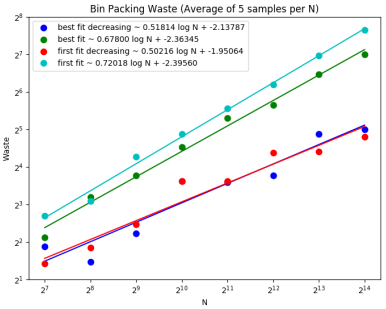

Figure 2: Bin packing waste plots.

Thus, this data confirmed that best fit decreasing and first fit decreasing both empirically achieve an an asymptotic waste that is approximately $O(n^{1/2})$, with their unsorted versions empirically achieving asymptotic waste bounds that are approximately $O(n^{2/3})$.

Alternatives for this type of project include changing the input distribution or exploring a different optimization problem.

A third project, focusing on modeling real-world phenomena, has been omitted for space. Details for that will be made available in an online appendix.

4 Observations on Student Performance

In this section, we report on our observations of how well students performed the above tasks in a recent offering of our course incorporating the above projects. The course consisted primarily of seniors. We required expositions of the algorithms studied, plots of performance, and conclusions comparing the relative performances. Central to this plotting requirement was the use of log-log plots. The report was graded on having log-log plots with appropriate regression analysis, clarity and conciseness in describing algorithms, gathering an appropriate amount of data, and clear prose analysis of results.

Our desired learning objectives for the students, by project, along with its success rate in our pilot offering, is as follows. The data follows the entire

enrollment of the class of 103 students. Each student was given the option to opt out of being included in our study, and none elected to do so.

1. **Students should be able to use meaningful, random sample inputs and interpret running time data on this data by noticing which types of input produce better running times for different algorithms.** The comparison of different sorting algorithms' running times tested on almost-sorted and on uniformly-permuted random inputs should get students thinking about this consideration. Based on their reports, over 80% of the students achieved this learning outcome.

   Students should be able to use log-log plots and a best-fit line to analyze running-time growth. Based on their reports, over 90% of the students in the course met this objective.

2. **Students should understand how algorithms can be used to optimize objective functions.** Because of a prerequisite for our course, students should have had experience with NP-hard optimization problems prior to our course. To assess this learning outcome, we checked to see whether students were correctly describing the measurement of waste by their algorithms. Based on their reports, over 85% of students achieved this learning outcome.

   Students should be able to compare heuristic algorithms based on how well they optimize an objective function. We measured this by determining whether students discussed the relative performance of the different bin-packing heuristics, including whether they commented on how the best-fit and first-fit heuristics performed better on pre-sorted inputs than on unsorted inputs. Based on their reports, 77% of the students in the class achieve this learning outcome.

5 Conclusion

Based on our experiences, we believe that our projects, or similar projects, are useful supplements to the topics taught in a traditional algorithms course. Further, our project-based course appears to be a useful addition to any Computer Science curriculum that includes project-based courses as requirements for graduation (as is done at our institution) or as electives.

References

[1] J. Ángel Velázquez-Iturbide and O. Debdi. "Experimentation with optimization problems in algorithm courses". In: *2011 IEEE EUROCON - International Conference on Computer as a Tool*. Apr. 2011, pp. 1–4. DOI: 10.1109/EUROCON.2011.5929294.

[2] Doug Baldwin and Johannes A. G. M. Koomen. "Using Scientific Experiments in Early Computer Science Laboratories". In: *SIGCSE Bull.* 24.1 (Mar. 1992), pp. 102–106. ISSN: 0097-8418. DOI: 10.1145/135250.134532. URL: http://doi.acm.org/10.1145/135250.134532.

[3] Albert-László Barabási and Réka Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.5439 (1999), pp. 509–512. ISSN: 0036-8075. DOI: 10.1126/science.286.5439.509. eprint: https://science.sciencemag.org/content/286/5439/509.full.pdf. URL: https://science.sciencemag.org/content/286/5439/509.

[4] Vladimir Batagelj and Ulrik Brandes. "Efficient generation of large random networks". In: *Physical Review E* 71.3 (2005), p. 036113.

[5] Dave Berque et al. "The KLYDE Workbench for Studying Experimental Algorithm Analysis". In: *Proceedings of the Twenty-fifth SIGCSE Symposium on Computer Science Education*. SIGCSE '94. Phoenix, Arizona, USA: ACM, 1994, pp. 83–87. ISBN: 0-89791-646-8. DOI: 10.1145/191029.191065. URL: http://doi.acm.org/10.1145/191029.191065.

[6] Marco Bressan et al. "Counting Graphlets: Space vs Time". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM '17. Cambridge, United Kingdom: ACM, 2017, pp. 557–566. ISBN: 978-1-4503-4675-7. DOI: 10.1145/3018661.3018732. URL: http://doi.acm.org/10.1145/3018661.3018732.

[7] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. "Approximation Algorithms for NP-hard Problems". In: ed. by Dorit S. Hochbaum. Boston, MA, USA: PWS Publishing Co., 1997. Chap. Approximation Algorithms for Bin Packing: A Survey, pp. 46–93. ISBN: 0-534-94968-1. URL: http://dl.acm.org/citation.cfm?id=241938.241940.

[8] Thomas H. Cormen et al. *Introduction to Algorithms*. MIT press, 2009.

[9] Michael T. Goodrich and Catherine C. McGeoch, eds. *Algorithm Engineering and Experimentation (ALENEX)*. Vol. 1619. Lecture Notes in Computer Science. Springer, 1999. ISBN: 3-540-66227-8. DOI: 10.1007/3-540-48518-X. URL: https://doi.org/10.1007/3-540-48518-X.

[10] Michael T. Goodrich and Roberto Tamassia. *Algorithm Design and Applications*. Wiley Publishing, 2014.

[11] Klaus Jansen et al., eds. *Experimental and Efficient Algorithms, Second International Workshop, WEA 2003, Ascona, Switzerland, May 26-28, 2003, Proceedings*. Vol. 2647. Lecture Notes in Computer Science. Springer, 2003. ISBN: 3-540-40205-5. DOI: 10.1007/3-540-44867-5. URL: https://doi.org/10.1007/3-540-44867-5.

[12] David S. Johnson. "A Theoretician's Guide to the Experimental Analysis of Algorithms". In: *Data Structures, Near neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges* 59 (2002), pp. 215–250.

[13] Jason King. "Combining Theory and Practice in Data Structures Algorithms Course Projects: An Experience Report". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 959–965. ISBN: 9781450380621. URL: https://doi.org/10.1145/3408877.3432476.

[14] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson, 2006.

[15] Donald Ervin Knuth. *The Stanford GraphBase: a Platform for Combinatorial Computing*. ACM Press New York, 1993.

[16] Jure Leskovec and Rok Sosič. "SNAP: A General-Purpose Network Analysis and Graph-Mining Library". In: *ACM Trans. Intell. Syst. Technol.* 8.1 (July 2016), 1:1–1:20. ISSN: 2157-6904. DOI: 10.1145/2898361. URL: http://doi.acm.org/10.1145/2898361.

[17] J. Matocha. "Laboratory experiments in an algorithms course: technical writing and the scientific method". In: *32nd Annual Frontiers in Education*. Vol. 1. Nov. 2002, T1G–T1G. DOI: 10.1109/FIE.2002.1157917.

[18] Catherine C McGeoch. *A Guide to Experimental Algorithmics*. Cambridge University Press, 2012.

[19] In: *J. Exp. Algorithmics* 1 (1996). Ed. by Bernard M. E. Moret. ISSN: 1084-6654.

[20] Mark EJ Newman. "Power Laws, Pareto Distributions and Zipf's Law". In: *Contemporary Physics* 46.5 (2005), pp. 323–351.

[21] Mark Ortmann and Ulrik Brandes. "Triangle Listing Algorithms: Back from the Diversion". In: *Algorithm Engineering and Experiments (ALENEX)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2014, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=2790174.2790175.

[22] A. Rényi and P. Erdős. "On Random Graphs". In: *Publ. Math* 6 (1959), pp. 290–297.

[23] Ian Sanders. “Teaching Empirical Analysis of Algorithms”. In: *SIGCSE Bull.* 34.1 (Feb. 2002), pp. 321–325. ISSN: 0097-8418. DOI: 10.1145/563517.563468. URL: http://doi.acm.org/10.1145/563517.563468.

[24] Peter Sanders. “Algorithm engineering–an attempt at a definition using sorting as an example”. In: *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2010, pp. 55–61.

[25] Richard T. Snodgrass. “On Experimental Algorithmics: An Interview with Catherine McGeoch and Bernard Moret”. In: *Ubiquity* 2011.August (Aug. 2011), 1:1–1:14. ISSN: 1530-2180. DOI: 10.1145/2015996.2015997. URL: http://doi.acm.org/10.1145/2015996.2015997.

[26] Jason Strahler et al. “Real-World Assignments at Scale to Reinforce the Importance of Algorithms and Complexity”. In: *CCSC NE* (). URL: https://par.nsf.gov/biblio/10158622.

A Project 3: Modeling the Real World

The third type of project uses algorithms to empirically measure how well various models of real-world phenomena match parameters found in actual data sets. The goal here is to further utilize performance plots as comparison tools, but now the algorithms are part of the data-gathering process. Ideally, the algorithms chosen for this type of project build upon algorithms that were taught in a prerequisite traditional algorithms course.

In our case, we chose *network science* as the application domain, that is, the study of the actual networks that arise from the Internet, social networks, computer networks, biological networks, etc. This is a rich and growing field of study, for which Computer Science has much to offer. Thus, focusing on this topic for our third type of project has the added benefit of providing useful domain knowledge for students in addition to further developing their skills in experimental algorithmics. In more detail, for this project, we focus on asking the students to design and implement algorithms that can compute the following statistics:

- **Diameter**: the length of a longest shortest path between two nodes in the graph. Students may choose to either compute this exactly or use a heuristic algorithm based on repeated breadth-first searches starting from random vertices or vertices far from the starting point of a previous breadth-first search.

- **Clustering-coefficient**: the ratio of three times the number of triangles over the number of paths of length 2 in a graph. It is used by social scientists to determine the degree to which a social network is separated into tightly-knit groups. To compute it, the students need to count the number of triangles in a graph, which is an interesting problem of growing interest in its own right, e.g., see [21].

- **Degree distribution**: for each possible degree in a graph, the number of vertices in the graph with that degree. This parameter is known to exhibit a power law [20] in many real-world networks,[2] so the goal of this component of the project is for the students to see if the graph in question has a degree distribution that exhibits a power law, by using the now-familiar log-log plot.

For reference graphs to use, one possibility is to use random graphs, which is the choice we took in our first implementation:

[2]Recall that data exhibits a power law if its frequency distribution can be characterized by a function $P(x) = cx^{-\alpha}$, for constants $c, \alpha > 0$. Typically, $2 < \alpha < 3$. See also, e.g., `https://en.wikipedia.org/wiki/Power_law`.

1. **Erdős-Rényi [22] random graphs**: these graphs, $G(n, p)$, are defined in terms of n vertices and the parameter, p, such that each pair of vertices in the graph is independently and uniformly chosen with probability p at random to form an edge. So as to avoid making the graph too dense, we recommend choosing p to be small, e.g., $p = (2 \ln n)/n$.

2. **Preferential-attachment [3] random graphs**: these graphs are defined by starting with two vertices connected by an edge and adding vertices to that at each step we add one new vertex v with a constant, d, number of edges back to previous vertices so that the probability a previously added vertex u receives a new edge from v is proportional to the (current) degree of u.

Using these definitions requires $\Theta(n^2)$ time to compute a given instance, even for sparse graphs. Fortunately, Batagelj and Brandes [4] provide simple linear-time algorithms for generating such graphs. Thus, in the final project, students generate random graphs for various numbers, n, of vertices. They then compute the diameter, clustering coefficient, and degree distribution for each graph. Students then plot average diameters and clustering coefficients for these graphs as a function of n, and students also plot the degree distribution for a specific graph instance to determine if it obeys a power law. In the former case, the growth rates tend to be so small that students are asked to plot the results using a lin-log scale, but they should still use a log-log scale for degree-distribution plots. The expected results are that Erdős-Rényi random graphs should not display a power law for their degree distributions, whereas preferential-attachment random graphs should.

An example set of charts produced from this project is as follows:

Thus, this latter chart empirically confirms a Erdős-Rényi random graph without a power law for its degree distribution and a preferential-attachment random graph with one.

Alternatives for this type of project include the following:

- Instead of using random graphs, use real-world graphs, such as found at the Stanford GraphBase [15] or SNAP [16].[3]

- Use other types of network science statistics, such as centrality measures, number of paths of length k, H-index, or numbers of small subgraphs of certain types, i.e., "graphlets" [6].

- Instead of problems in network science, choose a different application domain, such as machine learning, machine vision, or natural language processing.

[3]See also `https://snap.stanford.edu/data/`.

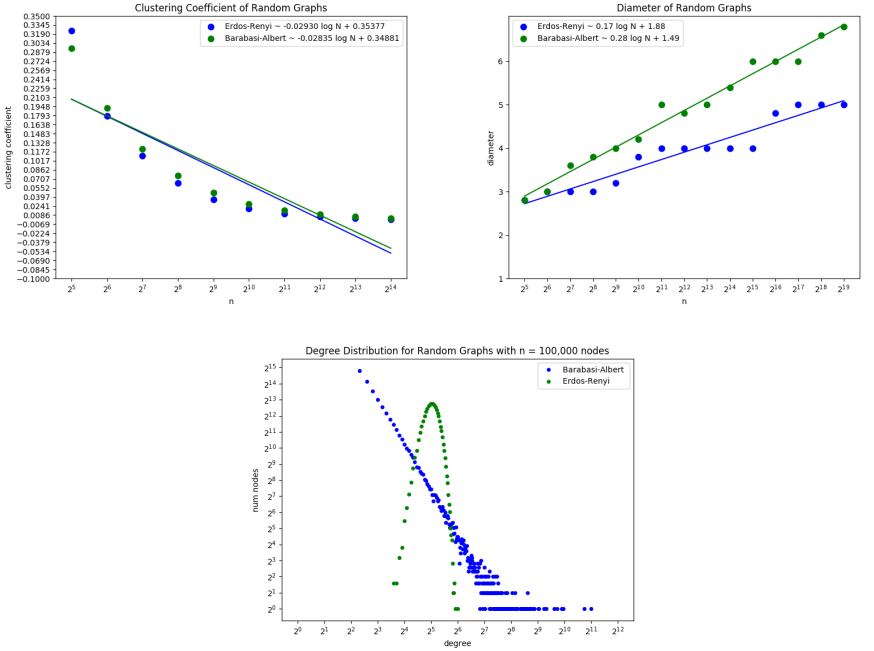

Figure 3: Project 3 charts.

A.1 Student Performance

3. (a) **Students should understand and be able to use mixed linear-log plots.** This learning outcome is a component of the third project when measuring average diameter of graphs, since, by the *small worlds* phenomenon, real-world graphs and popular random graphs tend to have small diameters. While students were told to plot this on a lin-log scale, completion of the analysis requires them to understand what the line fitting they will be doing means, as opposed to merely finding a new way to graph the output of their programs. This learning outcome also is designed to reinforce the related learning outcome for log-log plots. Based on our reading of student reports, over 90% of students achieved this learning outcome.

   (b) **Students should be able to determine whether the degree distributions of sample of graphs of increasing size exhibit a power-law distribution.** Students were taught the general concept of power-law distributions [20] and asked to determine for their random graphs, whether the distribution of degrees exhibited a power-law distribution. Based on our reading of the reports, over 80% of the students were able to successfully do this.

A.1.1 Sample Student Submissions

In Project 1, sorting algorithms, one of the questions asked of the students is to describe which sorting algorithms were most sensitive to input size and distribution, and which were least sensitive. Below are two student responses taken from their reports. In the first response, the student displays an accurate interpretation of the data. Conversely, in the second response, the student displays a clear lack of understanding of the sorting algorithm performance.

1. First student response:

   Insertion sort performs consistently worse on reverse input by a constant factor, and insertion sort performs significantly faster on almost-sorted input (the slope is much less).
   Merge sort has the same performance on all input permutations.
   All four shell sort versions have worse performance on uniform random input, and the same performance for reverse and almost-sorted input. The only exception is shell sort 3 having the same performance for all three input permutation types.
   Hybrid sort 1's asymptotic performance is most similar to insertion sort

with reverse input being slightly slower than uniform input and almost-sorted input being significantly faster. This is because a large part of hybrid sort 1 is insertion sort.
Hybrid sort 2's performance on different input permutation types are balanced. Hybrid sort 3 has equal performance in almost-sorted and reversed input, and uniform input is slightly worse. This is similar to merge sort because hybrid sort 3 uses mostly merge sort.

2. Second student response:

   Least sensitive to input size - Insertion sort
   Most sensitive to input size - Hybrid sort
   Least sensitive to distribution - Shell Sort (does not care at all)
   Most sensitive to distribution - Merge Sort (it's perfect)

An Authoring Process to Construct Docker Containers to Help Instructors Develop Cybersecurity Exercises*

Jack Cook[1], Richard Weiss[2], Jens Mache[3], Carlos García Morán[3], Justin Wang[4]

[1]New York University, Brooklyn, NY 11201
jcc9838@nyu.edu
[2] The Evergreen State College, Olympia, WA 98505
weissr@evergreen.edu
[3]Lewis & Clark College, Portland, OR 97219
{jmache, carlos}@lclark.edu
[4]Marquette University, Milwaukee, WI 53233
hsiaoan.wang@marquette.edu

Abstract

As instructors, we are more likely to use exercises that we can modify or that we helped to develop. The problem we address is how to help instructors create their own hands-on exercises. This paper describes the authoring process for the creation of cybersecurity exercises and the experience of creating two very different exercises. One of these exercises is about vulnerable Web services and leverages the LAMP stack and the other is about cryptography and ransomware and uses VNC. They both use Terraform to create Docker containers. We address the issues of creating exercises that involve multiple containers and can be run in a cloud environment, as well as on a single server.

This paper describes the process of creating these cybersecurity exercises in our framework. The streamlined process enabled the development of totally new exercises much faster than previously experienced. In the case of the ransomware exercise, it was two weeks from start to finish, compared to months for previous exercises.

1 Introduction

The development of new security exercises is a cornerstone to cybersecurity education. A number of platforms for teaching cybersecurity through hands-on exercises have been developed in last 15 years. Many of them have more than a dozen exercises. Yet, they are not truly scalable from the perspective of developing a community unless they facilitate the ability of knowledgeable users to modify existing exercises or contribute new ones. There are only a couple of frameworks that are designed to make this easy. In this paper, we examine the creation of two very different exercises in order to reflect on how to help instructors to create their own exercises in the domain of cybersecurity. Cybersecurity exercises have some special requirements: 1) exercises may rely on installing specific versions of software, including ones with vulnerabilities, 2) software environments need to be complete, i.e. more than just the vulnerable applications, and 3) exercises should run on a variety of platforms, e.g. AWS, Azure, Google Cloud and desktop.

Even when a platform provides help for creating exercises, there still is going to be a learning curve. We have tried to make that learning curve as gentle as possible in our platform. EDURange uses two powerful tools Docker containers and Terraform. These are commonly used in IT for creating and configuring flexible computing environments. The use of Docker containers is becoming more popular than the use of virtual machines (VMs), especially when multiple virtual computing environments would be needed. Similarly, Terraform is becoming popular because it works with multiple cloud frameworks to configure hardware and software and interacts well with Docker. We have constructed a layer on top of both of those, so as to minimize the prerequisite knowledge that instructors would need in order to create or modify exercises. A prerequisite that instructors would need is some familiarity with the Linux command line interface. However, we believe that this is less of an issue for most cybersecurity instructors. Thus, we have not developed a graphical interface for creating exercises, although that would certainly be possible.

The two new exercises that we developed were Ransomware and WebFu. The learning goal of Ransomware is to teach some of the basics of cryptography in a context that would be very clear and motivating to students. It also promotes the security mindset because it illustrates a failure mode. One generally thinks of cryptography as protecting secret information, but in this context it is about abusing it to prevent the owner from accessing data. WebFu teaches the basics of SQL injection. This exercise was developed as a gentle introduction to the topic and as our first attempt at a user interface that is different from the command line. While there are many CTF challenges that are based on SQL injection, we wanted an exercise for an introductory Web security course that would use a Web interface rather than the command line.

In our experience, students sometimes struggle because of their limited understanding of SQL databases and while there are many tutorials on SQL, they generally focus on how to use the language rather than how to abuse it. We also wanted students to be aware of code injection and to recognize that code is also data. Both of these exercises demonstrate that EDURange can accommodate a wider range of exercises, not just ones limited to using the command line interface in a bash shell.

2 Related Work

The other academic frameworks that we consider are Labtainers, DETERLab, SecKnitKit, Security Injections, NICE-challenge, and KYPO.

Of these, the only one that has addressed the issue of instructor-generated scenarios is Labtainers [7]. Labtainers has a collection of base Docker images that can be combined in a variety of ways to produce new exercises using Docker-Compose. This has some pros and cons. One advantage is that they have implemented a GUI that is aware of the base containers and allows the user to select them and compose them. The disadvantage is that if the user wants to go beyond the existing types of exercise, then they need to be familiar with Docker-Compose syntax, craft a unique Dockerfile, and also define networking rules from scratch. Labtainers can be used anywhere that has Docker installed, which could be a laptop or a Cloud environment.

In comparison, EDURange [9] takes a hybrid approach to this problem, by providing templates for instructors to modify while also allowing the use of custom container images. As a result, instructors can either provide their own pre-configured images, or extend the base SSH server with a list of their own bash scripts. One disadvantage is that there is no GUI, so exercise designers need to be familiar with basic Docker commands. Nevertheless, they don't need to know Docker Compose and instead can use JSON to combine Docker commands. EDURange can be used anywhere that has Docker installed, which could be a laptop or a Cloud environment.

DETERLab [4] also allows instructors to design their own exercises. It uses a combination of bash scripts and NS scripts. The NS scripts are not a commonly used format. There is not much documentation on the procedure for creating new exercises. The platform is tied to specific hardware. While it is free, there are resource limitations.

NICE-challenge has on the order of 100 exercises and has a staff of developers. The advantage for the instructor is that there is no expense and little effort to use the exercises. The disadvantage is that it is not possible for instructors to modify or contribute exercises or to host exercises on their own hardware resources. As with DETERLab this limits scaling because the

hardware resources are not easily expanded.

Security Injections [6], SecKnitKit [5], and SEED [1] are also valuable. While, an instructor cannot contribute or modify exercises, they are scalable in terms of the number of instances of a course. Security Injections does not require provisioning of VMs or containers, so it is easier to use than the other systems. SecKnitKit does use VMs that can be run locally on the instructor's hardware. SEED has one large VM that the students run and an associated textbook.

KYPO [8] is a very interesting system in terms of the exercises provided, and it is open source. However, it is not easy for instructors to add exercises or to run it on their own hardware.

There are several free non-academic frameworks such as Portswigger and overthewire.org. Instructors cannot modify or extend them, and they are generally harder to integrate into a course, in terms of assessment and prerequisite material.

The tools that make EDURange extensible, portable and scalable for instructors are Docker and Terraform. Section 3 steps through the process one would go through in EDURange to create a new exercise. Then, in sections 4 and 5, we discuss the particular requirements for those exercises.

3 Recipe for Creating New Cybersecurity exercises

Developing good hands-on exercises and homework assignments can be a difficult and time-intensive task. One standard method is backward design [10]. The author of an exercise would start with specifying the learning goals and develop a high-level description. They would translate the goals into concrete objectives and create a plan for assessing them. In the case of hands-on exercises, the objectives and assessment are often realized as tasks and criteria for determining that those tasks have been completed satisfactorily. Once the tasks have been described, they need to be implemented by creating the hardware and software environment. In EDURange, we use a collection of containers running on Ubuntu. The author of a scenario would describe each container in detail. They would specify the software and services they should provide. They specify accounts for students and services, and create files/artifacts that the students need to retrieve. The author also needs to configure the network, e.g. assigning IP addresses and redirecting ports. All of this should be done using scripts (mostly bash), so that it is easy to modify the exercise and create new containers.

Another approach that we have seen is to start with an existing virtual environment, where the instructor wants students to learn to work in that environment. In this approach, the instructor must then create the goals and

learning objectives, usually based on introspection to understand why that environment is important and what the essential goals and objectives are. Then, the author could generate tasks that would demonstrate those objectives in that environment. An example of such an environment is Metasploit on Kali Linux. Metasploit is a penetration testing framework. It is easy to find VMs for both Kali (the attacker) and Metasploitable (the target). Both of these can be ported to containers and networked together. Student accounts can be created in the Kali container.

Both of these approaches are reasonable. In practice, we often see a hybrid or middle-first approach where there are some ideas for exercises that are attractive, and they suggest goals and objectives. The part that EDURange can help with in both of these approaches is to make it easy to configure the virtual environment.

4 Developing an SQL-injection exercise (WebFu) using the LAMP stack

The goal of this exercise was to teach SQL injection in a hands-on fashion. This led to defining objectives, such as dumping tables from a database and bypassing a basic Web Application Firewall (WAF). These needed to be translated into an implementation in a concrete environment. For WebFu, the author chose the MySQL database system, and created the database schema and then the queries. The next section describes the experience of applying the tools in EDURange.

4.1 Applying the Tools

First, the author copied Terraform templates from another scenario and changed the container names to match the new scenario name. The author also copied the YAML file containing the assessment questions for the students and the Markdown file containing the scenario's student guide. Of course, the text in these files needed to be changed for the new scenario, but that was not difficult. Next, the author made a copy of the existing EDURange base image from DockerHub, which is based on a minimal Ubuntu installation. The author then set up a LAMP stack by extending the image with a) a MySQL database server; b) an Apache web server; and c) a PHP installation. These elements formed the infrastructure of the web application. After populating the database tables with data (made up of public datasets and the hidden artifacts or flags), the author pushed this modified image to EDURange DockerHub repository and edited a single line in the Terraform template to invoke it. Lastly, the author wrote a bash script for starting the MySQL and Apache services at scenario

launch time and added it to the JSON file. The development of this infrastructure took about a month (the author was a full-time student and this was a side-project). However, this set up can now be reused to create a wide range of scenarios for practising web security auditing skills. Potential labs include a website vulnerable to Cross-site Request Forgery (CSRF), Server-side Request Forgery (SSRF), Cross-site scripting (XSS), Local File Inclusion (LFI), and many other techniques. With the current infrastructure-as-code, the only task left to the author is writing or copying the vulnerable application(s).

On the scenario's development, it must be said that gaining familiarity with the Docker workflow was challenging at first. This was where most of the time was spent, since the author had a background in Linux system administration, but not in container-related technologies such as Docker. Every time the Docker image was changed, a commit needed to be made for the new container which resulted in a new image. Then, this image had to be tagged and pushed to the DockerHub repository. Finally, the instance of EDURange had to pull from the remote repository to update its changes. This is similar to Git's workflow, and the method of container deployment results in an agile and effective process. Once the author became familiar with this, the process went more quickly.

When trying to deploy this exercise in the classroom, the author ran into an unexpected problem. The exercise was being run on a cloud environment, but students needed to connect through the school's network through HTTP. Students were experiencing problems connecting, and it turned out that an internal firewall was blocking malicious traffic (i.e., the SQL injection strings) over plain-text HTTP. The solution was to use HTTPS, but we wanted to avoid requiring an instructor to create and deploy an SSL/TLS certificate. Using Terraform's "bind" property for SSL support and redirecting ports (from the container to the host) were the most significant changes. Using "bind" allowed us to easily set up HTTPS for the web application. The Let's Encrypt directory with the certificate and the private key on the host VM was made available to the guest container through a bind mount. This removed the need for creating and maintaining additional SSL certificates. What's more important about this, is that the whole process occurs at the scenario's creation time, thus not having the private key stored in the repository's image, ensuring confidentiality. Lastly, the port redirection implemented with the Terraform API spared us from having to write and maintain iptables rules. This was particularly helpful due to how easy it was to add redirection rules in the Terraform file.

4.2 Scripts and Files

This exercise required a working database. The tables were created using scripts. How and where to run the scripts was specified in a JSON file. The

author found it is easy to use an existing file from another exercise as a template, but ideally this would be produced by a user interface that would prompt the user for the information and produce the JSON file. The JSON file defines a list of the containers to be provisioned for this scenario, as well as three categories of other files that are used by Terraform to create the container: user files, system files, and global files. For each type of file, Terraform will take different actions to copy or execute them in order to prepare the scenario environment. Terraform copies a list of "User Files" into each students' home directory, "System Files" are executed once at scenario launch time for system configuration, and "Global Files" are added to the "/bin" folder so they can be run in a bash shell.

4.3 Using Docker and Terraform in WebFu

Terraform is a scripting platform most commonly used by system administrators and cloud/DevOps engineers to create and configure (provision) virtual machines (VMs) In the case where networks of VMs are needed, Terraform can be used with a VM orchestration configuration file which is similar to the Docker YAML-based language for defining networks of containers. One of the common uses of Terraform is to modify the state of a VM running on a Cloud by applying rules while the VM is running. This takes the place of the administrator logging in to all of the VMs in the network and running update commands. However, this is not how we are using it. Instead, we focus on rapidly setting up containers and configuring them.

Our general approach is to create a base Docker image and write configuration files to customize the image for each specific exercise. One could imagine using Docker scripts to do this, but there are potential problems with synchronization. For example, when configuring a network, some steps are dependent on others. Instead, EDURange uses Terraform scripts to create containers and network them together. In this case, the network must be configured before the containers can use it, otherwise there will be errors. Docker scripts do not provide a simple and reliable way to do that, while Terraform does. The Terraform scripts can be generated by EDURange as JSON files. This makes it possible to implement a user interface that would make it easier for instructors and authors to create their own scenarios. In EDURange currently, we have defined our exercises using Terraform templates that can be copied and adjusted. At the lowest level, this allows exercise developers to write bash scripts which modify an existing docker image to create the desired environment.

In practice, once contributors have written their desired scripts, they can just list them in JSON format to apply them and extend the docker image. This can be contrasted with other testbeds, in which manual editing of a Dockerfile or a NS file is required in addition to preparing low level scripts. Alternatively,

authors can create their own Docker images and incorporate them by modifying a single line to reference them.

Two Terraform templates are used to configure the virtual network. One of these templates defines how the host appears to the external network. It defines the IP address and external network, allowing Docker to expose ports publicly, as well as an internal network for hosting potentially vulnerable containers. Secondly, at least a single container Terraform file must be copied, which in the case of most EDURange exercises is the file "nat.tf.json". All of this can and will be automated. This file provisions a container with a basic SSH server running. The container is connected to both the external and internal networks, and Terraform will automatically add any of the students' user accounts to it, as well as any additional scripts listed in the JSON User Files. For all of the base Docker containers, the OpenSSH package must be installed because Terraform uses it to install files. For most of the exercises, SSH is also used by students to interact with the container. In one of the new exercises, VNC is used for student interaction.

At this point, with a new folder created and templates copied, contributors can make a choice of how to proceed based on the requirements of their scenario. If their scenario does not require the installation of new software or specialized containers beyond the capabilities of the base SSH server, then they can write bash scripts and list files in the JSON description file as their only means of customizing the scenario. On the other hand, if they need containers that are running databases, web servers, or other complex applications, then they can choose to build a Docker image that fulfills their requirements and list that image in the Terraform file instead of writing any configuration scripts.

With those steps done, the scenario would be ready to be tested. In the remainder of this paper, we describe the experience of a different author in creating a ransomware exercise.

5 Developing a Ransomware Exercise

In the midst of ever increasing incidents that are caused by ransomware attacks around the world, it is critical for students who are learning about cybersecurity to understand that a ransomware attack is based on asymmetric key encryption. This exercise mimics the execution of a ransomware attack. The goals are for students to learn how an adversary can weaponize public key cryptography and how that can be deployed on a vulnerable system. The newly added ransomware exercise introduces students to the foundations of ransomware and asymmetric key encryption. Through this scenario, the students learn how asymmetric key pairs can be weaponized and how end users can potentially stop such an attack by interrupting the corresponding cyber kill-chain.

5.1 Converting an Existing Exercise with Novel Requirements

This is an example of converting an exercise that was developed independently and then ported to EDURange. The first version of the exercise was developed on Windows [3]. It consisted mainly of a collection of Python scripts that installed a key pair, encrypted files, popped up some windows, and then decrypted the files if the user complied with some file modifications. The author developed this into an exercise with learning objectives and tested it on a Docker container for Windows. The last step was to integrate the container into EDURange.

We made some adjustments and converted the script into a Linux compatible program to ensure that it can be deployed within EDURange. We also adapted an existing ubuntu-VNC desktop docker container [2] to make the experience more realistic while providing the users a visual effect as the program gets executed. This was a significant extension because the previous exercises had only used the command line interface with SSH, so this involved a major change to exercise structure. The authoring process took about two weeks (part-time) starting from the Python scripts for Windows.

6 Results

The development of the SQL injection exercise was spread over 1-2 months, including 1 month for developing the infrastructure. At the end of that time, it was used in the classroom. Developing the exercise only required about 150 lines of PHP and HTML code, and about 250 lines of Terraform templates, mostly copied. The exercise process was very flexible and iterative because Terraform keeps track of interconnected components.

The development of the Ransomware exercise was even more rapid (two weeks), but it has not yet been tested in the classroom. Most importantly, the ease of importing a unique and pre-existing exercise to our platform illustrates the potential for adding many more exercises that are not just based on the current ones.

Both new exercises introduced completely novel interfaces for student interaction - a web application in the case of SQL Injection and a VNC desktop in the case of the Ransomware. In both cases, the authors had access to EDURange developers and were able to ask questions, but now this expands the range of topics that can be taught.

7 Conclusion and Future Work

Two new exercises were developed rapidly by people who were not familiar with the EDURange framework, which demonstrates that the framework has the flexibility for instructors to create new exercises. In one case, they were able to learn enough about the framework and develop an exercise in a matter of weeks. In the other case, it took a little over a month. In both cases, the authors had some specific learning goals in mind: in one case, teaching SQL injection, in the other, teaching how ransomware works and how it connects to modern cryptography. They differed in terms of the starting point. In one case, there was already a script that could be used on Windows, and that had to be translated from Windows to Linux and then integrated into EDURange. In the other case, everything had to be created from scratch. Potentially most cases will fall in between these two.

We expect that many instructors who teach cybersecurity have some tools that they often use and are familiar with. In that case, finding or constructing Docker containers with those tools and targets would be relatively fast, even more so if Linux is already being used. If the instructors are developing something new, provided the scope is reasonable, they should still be able to develop something in less than one term and have it ready for the next one.

This process has uncovered several new features that we want to add to EDURange. We plan to automate all of file copying and editing described in Section 3. Beyond that, we would create a GUI that could run those scripts. In addition, the Ransomware author has thought of a new exercise to be added. The exercise was inspired by the challenge-based learning pedagogy where an improperly configured Linux image and applications will be presented to the students. By mitigating the challenges or adjusting the configuration of the image, the student will receive flags to enter into the forms within EDURange for a score. This exercise will add system security and proper privilege configuration of users and the file system infrastructure to our list of topics. In addition to the image, a descriptive list of exercise objectives will be provided to the students as guiding reference so that they are properly informed of what the ideal configuration should be.

References

[1] Wenliang Du. Seed labs: Using hands-on lab exercises for computer security education (abstract only). In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, page 704, New York, NY, USA, 2015. Association for Computing Machinery.

[2] github.com/fcwu. Docker-ubuntu-vnc-desktop. `https://github.com/fcwu/docker-ubuntu-vnc-desktop`, 2021.

[3] github.com/ncorbuk. Python-ransomware. `https://github.com/ncorbuk/Python-Ransomware`, 2021.

[4] Jelena Mirkovic and Terry Benzel. Teaching cybersecurity with DeterLab. *IEEE Security & Privacy*, 10(1):73–76, 2012.

[5] Ambareen Siraj and Sheikh Ghafoor. Crest-security knitting kit: Readily available teaching resources to integrate security topics into traditional cs courses (abstract only). In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 1058, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Blair Taylor and Siddharth Kaza. Security injections@towson: Integrating secure coding into introductory computer science courses. *ACM Trans. Comput. Educ.*, 16(4), June 2016.

[7] Michael F. Thompson and Cynthia E. Irvine. Individualizing cybersecurity lab exercises with labtainers. *IEEE Security & Privacy*, 16(2):91–95, 2018.

[8] Jan Vykopal, Radek Ošlejšek, Pavel Čeleda, Martin Vizváry, and Daniel Tovarňák. Kypo cyber range: Design and use cases. In Cardoso J., Cardoso J., Maciaszek L., Maciaszek L., van Sinderen M., and Cabello E., editors, *Proceedings of the 12th International Conference on Software Technologies - Volume 1: ICSOFT*, pages 310–321, Madrid, Spain, 2017. SciTePress.

[9] R. Weiss, F. Turbak, J. Mache, and M. E. Locasto. Cybersecurity education and assessment in EDURange. *IEEE Security & Privacy*, 15(3):90–95, 2017.

[10] Grant Wiggins and Jay McTighe. What is backward design. *Understanding by design*, 1:7–19, 1998.

2022 CCSC Southwestern Conference Committee

Michael Shindler, Conference Chair ..... University of California, Irvine
Megan Thomas, Papers Chair ..... California State University, Stanislaus
Mariam Salloum, Authors Chair ..... University of California, Riverside
Adam Blank, Posters Chair ..... California Institute of Technology
Michael Shindler, Panels/Tutorials Chair .University of California, San Diego
Paul Cao, Lightning Talk Chair ..... University of California, San Diego
Michael Shindler, Partner's Chair ..... University of California, Irvine

Regional Board — 2022 CCSC Southwestern Region

Michael Doherty, Region Chair ..... University of the Pacific
Niema Moshiri, Treasurer/Registrar ..... University of California, San Diego
Bryan Dixon, Regional Representative .... California State University, Chico
Angelo Kyrilov, Webmaster ..... University of California, Merced
Colleen Lewis, Past Region Chair ..... Harvey Mudd College
Youwen Ouyang, Past Conference Chair ..... California State University, San Marcos