Are We Fair? Quantifying Score Impacts of Computer Science Exams with Randomized Question Pools

Max Fowler
mfowler5@illinois.edu
University of Illinois
Urbana, Illinois, USA

David H. Smith IV
dhsmith2@illinois.edu
University of Illinois
Urbana, Illinois, USA

Chinedu Emeka
cemeka2@illinois.edu
University of Illinois
Urbana, Illinois, USA

Matthew West
mwest@illinois.edu
University of Illinois
Urbana, Illinois, USA

Craig Zilles
zilles@illinois.edu
University of Illinois
Urbana, Illinois, USA

ABSTRACT

With the increase of large enrollment courses and the growing need to offer online instruction, computer-based exams randomly generated from question pools have a clear benefit for computing courses. Such exams can be used at scale, scheduled asynchronously and/or online, and use versioning to make attempts at cheating less profitable. Despite these benefits, we want to ensure that the technique is not unfair to students, particularly when it comes to equivalent difficulty across exam versions.

To investigate generated exam fairness, we use a Generalized Partial Credit Model (GPCM) Item-Response Theory (IRT) model to fit exams from a for-majors data structures course and non-majors CS0 course, both of which used randomly generated exams. For all exams, students' estimated ability and exam score are strongly correlated ($\rho > 0.7$), suggesting that the exams are reasonably fair. Through simulation, we find that most of the variance in any given student's simulated scores is due to chance and the worst of the score impacts from possibly unfair permutations is only around 5 percentage points on an exam. We discuss implications of this work and possible future steps.

CCS CONCEPTS

• **Social and professional topics → Student assessment**.

KEYWORDS

randomized exams, fairness, assessment, programming, exam generation

ACM Reference Format:
Max Fowler, David H. Smith IV, Chinedu Emeka, Matthew West, and Craig Zilles. 2018. Are We Fair? Quantifying Score Impacts of Computer Science Exams with Randomized Question Pools. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

There exist two challenges to large scale assessment in computing courses. First, computer science departments globally continue to experience increased enrollment of majors as well as increased numbers of non-majors enrolling in computer science or programming courses [5, 6, 10, 17, 25]. This increase in demand has also led to the expansion of online programs [20, 22], especially in light of the challenges presented by the COVID-19 pandemic. Given these challenges, computer-based assessment presents many advantages for large scale computing courses. Computer-based assessment allows for automated grading, reducing grading burden for staff and allowing for instant feedback [1, 13, 30]. Computer-based assessment also allows us to assess our students in more authentic settings, in particular on machines with access to compilers/interpreters where students can debug their code. Finally, computer-based assessment facilitates both online as well as asynchronous assessment.

The adoption of online and/or asynchronous assessment, however, makes exam security a concern [7, 26]. One typical method of providing exam security is to use versioned exams [14, 15, 28**?** , 29]. Automatically generating these versioned exams can be accomplished by selecting questions at random from question pools as well as by using item generators to generate unique instances of parameterized questions [**?**]. With enough question versions, each student can be presented a completely unique exam.

This approach to assessment does, however, come with a downside. Specifically, there is no guarantee that generated exams are all the same exact level of difficulty. Existing work with automated item generators suggests faculty can generally write item generators that generate variants of similar difficulty [8], but this is harder to guarantee when using pools of "similar" but manually written questions. Despite our best efforts, we *may* introduce unfairness in our versioned exams by imperfectly constructing our question pools.

On a practical level, we are interested in exploring unfairness in the context of versioned exams. In this experience report, we analyze computer-based exams generated with randomized question pools from two computer science courses using Item Response Theory (IRT) models to detect and quantify possible unfairness.

The rest of the report is organized as follows. In Section 2, we briefly lay out related work on exam versioning and exam fairness. We describe our data in Section 3 and our analysis methods in Section 4. We present the results for our analysis in Section 5 and

discuss how these results may be helpful to future courses with version exams in Section 6. Finally, we conclude.

2 RELATED WORK

2.1 Automatic Generation of Exams and Exam Questions

There is a significant body of work on question and exam generation. These include both individual question generators, randomized pools, and whole exam randomization. We briefly review some of this work below.

Automatic item generation is the creation of groups of similar questions using parameterized question templates. Attali's work with Educational Testing Service's *Quick Math* generator found merely 2 of 57 automatically generated items included difficulty mismatches, and of those, less than a percent of students encountered sufficiently different generated questions [3]. Use of templates also empowers faculty to write their own question generators. Chen's work investigating the difficulty variance of such generators found that faculty can almost always write item generators that produce questions of equivalent difficulty [8]. Further, techniques proposed by de Chiusole et al. allow for automatic determination of item equivalence, meaning item generators can be more easily examined and constrained to generate questions of equivalent difficulty [12].

Similar generation was used by Rusak and Yan to generate entire exams for their Probability for Computer Scientists course [24]. While the domain of generation was relatively constrained — mathematics problems appropriate for a sophomore CS course — their generation pipeline is readily extendable. Instructors provide question skeletons and a set of dependent parameters for generation to choose from where needed. The exam generator then generates a random exam for each student, ensuring no combination of dependent parameters and generated parameters is repeated. Their generation pipeline is reasonably successful, with 92.6% of generated problems generating without error [24].

Another approach to generating exams is suggested by Ashraf et al [2]. Rather than focus on question generation, which could use the techniques above or could use question pools, they focus on generating exams based on some desired learning outcomes or educational taxonomies. They developed a framework to allow for instructors to map their questions or item generators to different learning outcomes and levels of Bloom's taxonomy — effectively pools — and then generate exams ensuring specified amounts of coverage for topics and content levels.

2.2 Judging The Fairness of Exams

A number of methods have been proposed to investigate the fairness of exams, both on an item level and exam-wide level. We review some of these methods below.

Older exam generation methods use large banks of questions to generate exams. Hwang et al. developed a parallel test sheet algorithm for generating national exams that uses a large test bank of questions and their meta-data to generate equivalently difficult exams. Their algorithm uses Tabu search to enforce constraints on difficulty and covered topics, with examiners only needing to specify the type of exam, concepts to appear on the exam, and bounds for the length of the exam in minutes [18]. However, this approach does require sufficiently large test banks and data on the questions to generate sufficiently many exams.

Course	Semester	Exams (and points)
Data Structures	Spring 2019	40 points: Exam 0 70 points: Programming Exam 1 – 3 100 points: Theory Exam 1 – 3 300 points: Final Exam
CS0 Python	Fall 2019	150 points: Exam 1 – Exam 3 250 points: Exam 4 (Final)
CS0 Python	Spring 2020	150 points: Exam 1 – Exam 3 250 points: Exam 4 (Final)
CS0 Python	Fall 2020	150 points: Exam 1 – Exam 3 250 points: Exam 4 (Final)

Table 1: Semesters and exam point breakdowns for the four data sources we use for our analysis.

Davidson et al. advocate for using differential item functioning (DIF) as a way to validate questions on exams and investigate possible item bias exams may have for students of different demographics and backgrounds [11]. They applied DIF analysis to an archetypal, CS1 paper exam with code tracing and writing questions. While they only detected one question with statistically significant but functionally irrelevant bias, investigating exams for such bias going forward is one important dimension of exam fairness.

When reusing questions, historical performance on questions can be used to reduce unfairness in subsequent semesters. Sud et al. show how exams generated with question pools can have their difficulty variance reduced through Monte-Carlo based estimation of exam variants' scores and standard deviations using submissions for questions from previous semesters and exams. Variants that are too easy or too difficult can then be discarded as opposed to being presented to students [23]. The weakness of this approach is that new questions cannot have their difficulty estimated directly, although even reducing the variance of other pools is a worthwhile reduction in unfairness.

Clegg et al. propose simulating possible student mistakes as a way to investigate question unfairness as well as autograder configuration [9]. They present the results of observing student code mistakes from real student submissions and develop a set of mutation operators that can be applied to existing correct code to make it incorrect. They propose automating the process of making these mistakes and testing them, using the results of these tests to assign partial credit students get for types of mistakes. They propose this as an algorithmic way to more fairly provide points from autograding results.

3 DATA AND COURSES

For our analysis, we use data from two classes, a CS0 course for non-majors and a Data Structures course for majors. Both courses use PrairieLearn for hosting their exams [30]. The 2019 semesters were able to host their exams asynchronously with in-person proctoring, while the 2020 semesters used synchronous online proctoring due to COVID-19. Information about the courses is provided in Table 1. We briefly describe the two courses and their exam structure below.

The CS0 course, taught in Python, was organized as a flipped class that covered one major topic each week. Students were assigned weekly readings and accompanying assignments of multiple-choice and true/false questions to complete before lecture. The

weekly 90-minute lecture used peer instruction to reinforce concepts, and the weekly 80-minute lab consisted of practice activities, designed for solo or pair work in 2019 and then increasingly small group work when shifted online due to COVID. Each course topic also had a weekly homework assignment that consisted of a mix of small programming (i.e, a small function's size) and short answer (e.g. 'What does this for loop print?') questions.

Each semester, the CS0 course had five proctored exams, Exams 0 through 4. All the exams had a 50-minute time limit, except for the final exam (Exam 4) which had a time limit of three hours. Exams typically consisted of 20–30 question pools (40–45 on final exams), from which questions were randomly selected. In this way, each student received their own exam. Pools were constructed of questions selected to be of similar difficulty and content, per instructors' experience. Further, some question pools used item generators, which guaranteed exam uniqueness at least in the specific parameters of generated questions. In this work, we focus on Exams 1 through 4, as Exam 0 is worth a small amount of credit and is primarily to acclimate students to the testing platform.

The Data Structures course uses C++ to teach students how to build and utilize a myriad of common data structures, including queues, balanced trees, hash tables, and graphs. The course featured three lectures a week covering course content and one lab session a week where students worked on programming exercises in a collaborative setting. Every two weeks, students completed a small programming project, while students also had a small daily problem to complete for practice and extra credit.

The exams in the course were also generated with randomized pools and had multiple different structures. Exam 0 was a small introduction to how different questions in the course are presented. Three of the exams were programming exams, with a small number of questions focused on writing C++ code to implement/use data structures with a 110 minute time limit. Another three of the exams were theory exams, with randomized pools selecting from multiple-choice and numerical answer questions about data structures and data structure theory with 50 minute time limits. Finally, the course's 3-hour final exam included both theory and programming exam style questions. We consider all of the exams in our analysis to see if different kinds of exam configurations are more or less fair.

Both courses allowed for interpreter or compiler access during exams. This allowed students to write, run, and test code before submitting answers to programming-based questions. Providing this more authentic programming experience during assessment is part of the motivation for computer-based testing to begin with.

Given the different forms of exams between the two classes, we are interested to see how different exam structures contribute to the fairness of randomized question pools. Beyond that, the amount of data we use is simply to give us access to more exams worth of data and more chances to investigate possible unfairness in exams created with randomized pools.

4 METHODS

Below, we detail our analysis approach. The pipeline used is presented in Figure 1.

For obtaining student ability levels, we used an Item-Response Theory (IRT) model from the R's mirt package. Given a (potentially sparse) set of student scores for a collection of questions, an IRT model characterizes both the questions and the students. We used a two-parameter IRT model that estimates both a question's difficulty and its discrimination (i.e., its ability to differentiate between high and low ability students). While a *dichotomous* IRT model requires student scores to be either correct (1) or incorrect (0), we used mirt's Generalized Partial Credit Model (GPCM) to support polytomous scores.

GPCM is an extension on the Partial Credit Model (PCM) which allows for a non-uniform discrimination of questions on an exam. This is important for our purposes for two reasons. First, exams in both classes allow for partial credit, which is non-dichotomous. Secondly, the rate at which students get a certain amount of partial credit *may* — and likely does — vary between different ability levels. For example, it is reasonable to expect that students of low ability are more likely to get less partial credit than students with high ability.

The first step in our analysis is to transform item scores in our exam data from a continues grading scale to one that is discrete. This was done by rounding the scores to the nearest whole number which, though slightly reducing the accuracy of our analysis, should have a negligible impact on estimated student ability. The purpose of this transformation is to satisfy the requirement of GPCM that the set of possiable item scores be a relatively small number of discrete values in order for the model to converge reliably.

Then, ability is fit for each exam using GPCM so that we snapshot the student's latent ability on that exam at that time. This way, these methods can possibly be used to adjust students' scores during the semester should they receive an unfair exam as opposed to having to wait until the end to analyze all the exams at once. We perform this analysis to characterize the spread and correlation of student ability versus exam score.

After fitting the model and thus gaining the estimated student ability, item difficulty, and item discrimination scores, we use we use *mirt*'s simdata capability to simulate students taking a variety of exams. Specifically, for each exam, we generate 100 randomized exam permutations. Then, for each of these randomized exam variants we simulate 500 attempts per student. As IRT is probabilistic in nature, running these simulations will generate a distribution of scores for each of the exam variants that a student with a given ability would likely to achieve. To measure exam fairness for a given exam, we calculate the ratio between score dispersion attributable to exam permutations and dispersion attributable to student performance or chance, given by:

$$SS\,Ratio_i = \frac{\sum_{j=1}^{n}\sum_{k=1}^{m}(s_{ijk} - \mu_{ij})^2}{\sum_{j=1}^{n}\sum_{k=1}^{m}(s_{ijk} - \mu_i)^2} \tag{1}$$

where,

- i: A given student i
- j: The permutation j out of 100 possible permutations (n)
- k: Attempt k out of the 500 (m)
- s_{ijk}: Student i's kth attempt on the j permutation of the given exam

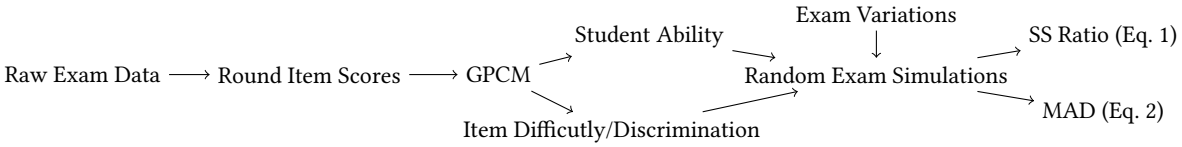


Figure 1: Analysis Pipeline

- μ_i: The student's mean across all simulated attempts for the given exam, regardless of permutation
- μ_{ij}: The student's mean across all simulated attempts for permutation j of the given exam

The ratio's numerator is represented the within-permutation variance whereas Ratio's numerator and denominator represent within permutation and across-permutation variance respectively. This ratio indicates the degree to which a student score is explained by the dispersion that is expected between students' attempts and how much is due to the permutations themselves. A ratio close to 1 indicates that each generated exam permutation has a similar mean and standard deviation as all other permutations, with a ratio of exactly 1 indicating that all of the dispersion in scores is due to differential student performance or chance on individual attempts. A ratio close to 0 indicates that the exam permutations had notably different means and standard deviations for student i, with a ratio of exactly 0 indicating all of the dispersion in scores is due to the permutations and that none of the permutations have score distributions which overlap for student i. To summarize with an example, an exam with an SS Ratio of 80% has 80% of the given student's score variance attributable to chance and only the remaining 20% attributable to question differences in generated permutations.

To quantify the difference in students' scores, we use the simulated data to calculate the mean absolute deviation (MAD) as a measure of points lost or gained during a specific exam attempt. The MAD for a given student i on a specific exam is given by:

$$MAD_i = \frac{1}{n}\sum_{j=1}^{n} |(\mu_{ij} - \overline{\mu_{ij}})| \tag{2}$$

where,

- i: A given student i
- j: The permutation j out of 100 possible permutations (n)
- μ_{ij}: The student's mean across all simulated attempts for permutation j of the given exam
- $\overline{\mu_{ij}}$: The mean of *all* means μ_{ij} for student i and all possible n (100) permutations of the given exam

For example, for a given student on exam 1, we first calculate the mean of every permutation j we simulated for that student, all of the μ_{ij} for exam 1. Then, we calculate the mean of all of these means; the mean of the means of the students' permutation scores on exam 1, $\overline{\mu_{ij}}$. We then use Equation 2 to calculate the mean absolute deviation of that students' mean permutation scores. It is important to note that MAD does not specify directionality and, as such, should be interpreted as the average number of points gained or lost between their average scores of their simulated attempts on each of the exam variants. When we present MAD, we do so in percentage points out of 100: for example, a MAD of 11 means on average that student received or lost 11 more percentage points on a given exam than average for that student's simulated attempts across permutations.

MAD and variance together are both important for judging fairness and impact on students' scores. As stated, MAD is used to identify the number of points lost or gained between exam variants but does not indicate whether the deviation in scores is due to the natural variance that exists between student attempts or if it is attributable to the variance from the exam variants. The SS Ratio, meanwhile, does attribute the point loss to either exam generation or student behavior/chance. An exam is unfair if the SS Ratio is low in that the generated permutation dictates most of the obtained score. This unfairness is worse for exams with a high MAD, as the majority of the difference in final score is due to the exam permutation received. In practice, reducing exam variance to zero is impractical. The goal should be to keep as high an SS Ratio as possible, with ideally low MADs so that potential unfair exam generation is less costly to students. In Section 5, we present the SS Ratio and MAD for all students as violin plots to present both spread and density.

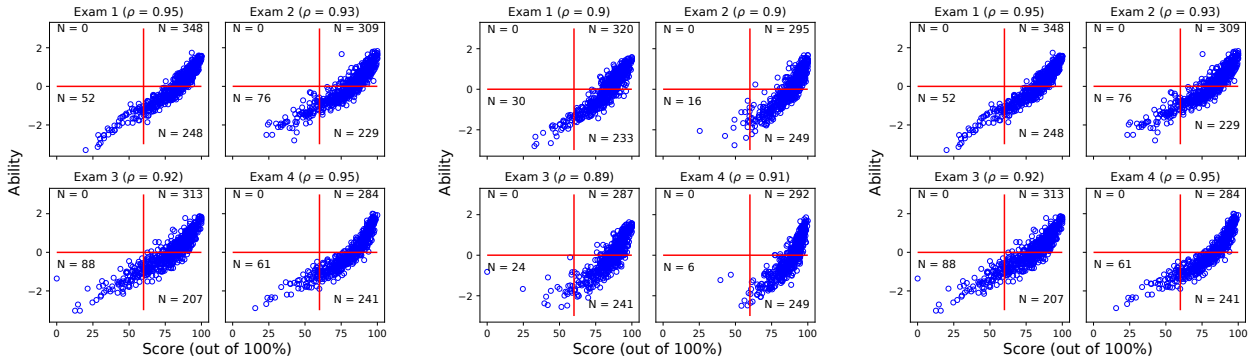
5 RESULTS

5.1 Comparing student ability to their exam scores

First, we present results from our GPCM on the three semesters of CS0 data. Each figure provides student ability, between 3 and -3, on the y-axis and the students' percent score out of 100% on the x-axis. Each plot also features two red lines set at the midpoint of student ability for that exam (functionally 0) and an exam score of 60%. The correlation between estimated ability and score, Spearman's ρ, is given next to each exam's name. The total count for each quadrant of the red axes is given in the corners of each plot. Fall 2019 is shown in Figure 2a, Spring 2020 in Figure 2b, and Fall 2020 in Figure 2c.

Across all exams and semesters, the CS0 exams have strong correlations between ability and scores. The lowest correlation is ρ = 0.89 for Spring 2020's exam 3, which still means strong correlation for Spearmen's ρ. This lower correlation corresponds to more extreme outliers appearing in Spring 2020 versus other semesters. The upward curve present for CS0 distributions may be due to a ceiling effect, with many students receiving high scores.

With respect to analyzing these outliers, Spring 2020's Exam 2 and 3 in Figure 2b and Fall 2020's Exam 1, 3, and 4 from Figure 2c have some interesting cases in the bottom left quadrant, with students of abilities similar to their classmates appearing to perform notably worse. Most of these outliers appear to occur for students who have relatively fewer correct questions than the average student on a given exam.



(a) Fall 2019 **(b) Spring 2020** **(c) Fall 2020**

Figure 2: CS0 Exam Scores vs Ability

The same plots are presented for Data Structures in Figure 3. As with CS0, all exams show a strong correlation between student ability and score. Most of the exams have a correlation $\rho > .9$, with the final ($\rho = 0.69$) as an outlier, suggesting the final may have been easier for students of lower ability to receive higher scores on than typical. All exams still show strong correlation, implying the courses' exams are reasonably fair. Theory exam 1 appears to have been particularly difficult, but no more unfair than other exams.

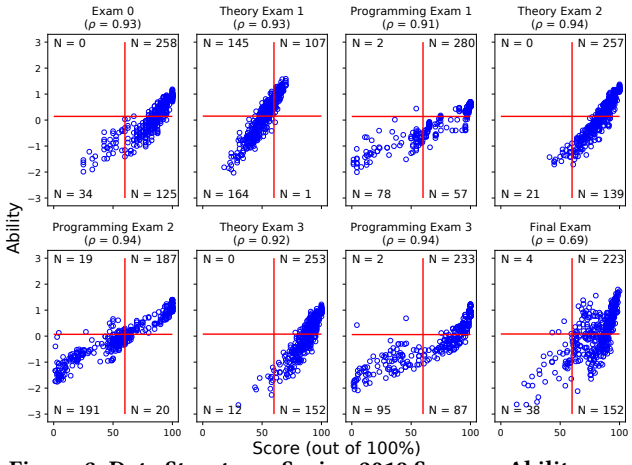


Figure 3: Data Structures Spring 2019 Score vs Ability

5.2 Simulated results: how much variance is due to student performance rather than exam fairness?

The simulated results are provided as both the MAD and the SS-Ratios. The CS0 course's MADs are given in Figure 4. CS0 exams got better with respect to the possible MAD across each semester, with higher possible score gaps in Fall 2019 decreasing through Fall 2020. The worst possible gaps were in Fall 2019's exam 3 and final, a little over 5 percentage points. The CS0 average MAD was 4.65 in Fall 2019, 2.28 in Spring 2020, and 2.37 in Fall 2020.

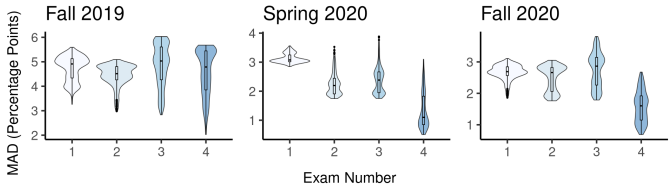


Figure 4: The mean absolute deviation in percentage points for CS0 exams. At worst, there was a slightly above 5 percentage point mean deviation in Fall 2019.

We also present the density of SS Ratios as a measure for how much of a students' score difference was due to chance as opposed to exam permutation. Figure 5 shows these variances for the three semesters of CS0. The CS0 exams got better across semesters, with Fall 2019 having an average of 32.4% and Spring 2020 and Fall 2020 having an average of 69.3% and 71.4% respectively.

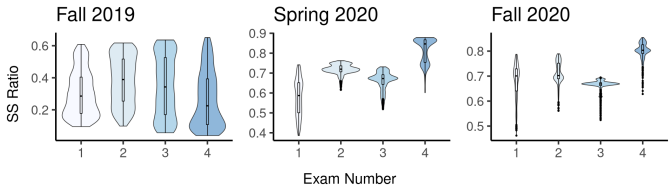


Figure 5: The SS ratio for CS0 students' simulated scores.

The MADs and variance ratio for the data structures course are presented in Figure 6 and Figure 7 respectively. The exams are labeled in the order they were offered in the course, with exam 0 as 1 and the final as 8. The data structures MAD is large in some instances — for example, an up to 19 percentage point score difference on the third exam, programming exam 1. Due to the small number of questions on data structures' programming exams, we attribute these large gaps to IRT. Specifically, students who succeeded on all the questions would be assumed perfect and simulate as such, creating a larger gap between these simulated perfect students and more common student behavior.Data Structures' average MAD was 2.77 percentage points and SS Ratio was 90.6%.

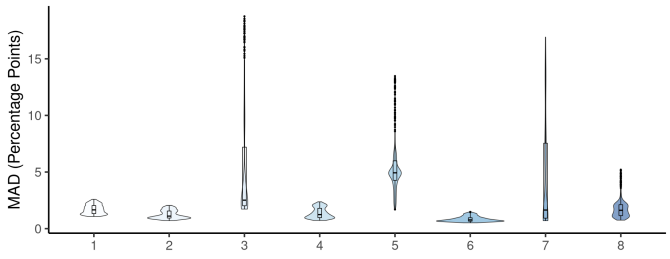


Figure 6: The mean absolute deviation for the Data Structures exams.

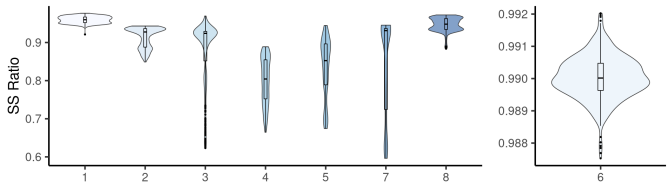


Figure 7: The SS ratio for the Data Structures exams. Exam 6 had so little variance caused by different exam permutations that it is plotted separately.

6 DISCUSSION

Ultimately, the apparent fairness of the generated exam permutations is reasonably pleasing. The worst semester in this respect was Fall 2019, but even this variance in score is less than half a letter grade. While large between permutation variance is unfair, most of this variance was below half a letter grade for a given exam. Fall 2019 having the worst variance makes sense to us as this was the first semester this CS0 course was taught; in subsequent semesters, the instructor had data with which to trim pools down and better balance pool difficulty. While not provable from this data, it is likely the reduction in unfairness is largely due to exam construction, because we know the instructor adjusted pools to remove outlier questions.

The results from the Data Structures course are surprising. Initially, we thought those exams may show more variance in scores due to permutation due to those exams tending to have less overall questions than the CS0 exams. However, the Data Structures exams had the least variance attributable to exam permutations out of the semesters studied. On examination, this may be because Data Structures' exam questions were commonly analogous, with the same question featuring different numbers constituting a "new" question in the pool. As such, there is likely a smaller chance of faculty making incorrectly balanced pools, similar to how item-generators tend to produce questions of similar difficulty in Chen et al.'s work [8]. The largest MADs in Data Structures were on programming exams (3, 5, and 7 in Figure 6), which had comparatively fewer questions than other exams. This may indicate that small numbers of questions on exams allow for high variance in scores even when the generation is fair.

For our purposes, the low score impact even from the worst variance in students' scores suggests we are safe to continue using randomized pool exams. However, we would still like to see if this affected final grades, and if so, compensate students who received exams less easy than their similar ability peers. Totally eliminating unfairness is likely unfeasible, so a possible barometer to use going forward will be to ensure exam unfairness is no worse than exam precision, such as the $\frac{1}{3}$ final letter grade precision estimated by Scott et al. in a large introductory physics course [27].

Our work does not address how to compensate students for poor scores due to chance. One potential solution may be found in the use of second-chance testing wherein students can elect to take a second exam in order to replace or improve their score [16, 19]. This, in combination with carefully balanced question pools and frequent low-stakes exams [4, 21], could be used to both lower the chances and minimize the negative impact of a student being dealt a potentially unfair exam.

6.1 Limitations

As this is an experience report, there is a limit to what we can say with our results. Even though the results from this dataset give us little room for concern in *these* randomized exams, our results do not necessarily generalize to all versioned exams generated from question pools. A larger study is in order to map out trends in these kinds of examinations, as well as to investigate better ways to mitigate what unfairness does appear when using randomized pools, even if the score impact is low.

Some hedging is appropriate for the estimates of within exam variance. These estimates are simulated using an IRT model. As this variance has not been empirically determined, we cannot be sure that the estimates provided are perfect, and the actual score variance within these versioned exams in a real course will vary.

7 CONCLUSION

In this report, we investigate whether the exams generated from random pools used in two courses were fair. We determine this fairness by calculating how much variance in students' scores is due to student performance and not which randomly permuted exam they received. We also calculate the mean absolute deviation (MAD) in scores as a way to quantify how unfairly harder (or easier) an exam permutation may have been.

We use IRT simulations to simulate students taking 100 exam permutations 500 times each. Ultimately, most of our exams were reasonably fair, with 60% or more of the point difference between any exam a simulated student took being due to student performance and not the specific random exam they received. Data Structures had an average MAD of 2.77 percentage points and an average SS Ratio of 90.6%, while CS0 had an across semester average MAD of 3.10 and SS Ratio of 57.7%. Even our worst semester, Fall 2019's CS0 class, had only a slightly over a 5 percentage point impact on two exams being the worst result of getting specific exam permutations.

Our work gives us confidence that these randomized pool exams can be reasonably fair and have gotten more fair for the CS0 course as time has gone on. We are interested in refining these methods on a larger selection of courses that use versioned exams to investigate if the trends seen here appear in more and varied courses. Following, we hope to use exam fairness detection in the future as a way to award points to students who receive the rare unfair exam variant.

REFERENCES

[1] Kirsti M Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (2005), 83–102. https://doi.org/10.1080/08993400500150747 arXiv:https://doi.org/10.1080/08993400500150747

[2] Ashraf Amria., Ahmed Ewais., and Rami Hodrob. 2018. A Framework for Automatic Exam Generation based on Intended Learning Outcomes. In *Proceedings of the 10th International Conference on Computer Supported Education - Volume 2: CSEDU,*. INSTICC, SciTePress, 474–480. https://doi.org/10.5220/0006795104740480

[3] Yigal Attali. 2018. Automatic Item Generation Unleashed: An Evaluation of a Large-Scale Deployment of Item Models. In *Artificial Intelligence in Education*, Carolyn Penstein Rosé, Roberto Martínez-Maldonado, H. Ulrich Hoppe, Rose Luckin, Manolis Mavrikis, Kaska Porayska-Pomsta, Bruce McLaren, and Benedict du Boulay (Eds.). Springer International Publishing, Cham, 17–29.

[4] E. G. Bailey, J. Jensen, J. Nelson, H. K. Wiberg, and J. D. Bell. 2017. Weekly Formative Exams and Creative Grading Enhance Student Learning in an Introductory Biology Course. *CBE—Life Sciences Education* 16, 1 (March 2017), ar2. https://doi.org/10.1187/cbe.16-02-0104

[5] Betsy Bizot and Stu Zweben. 2019. *Generation CS, Three Years Later.* Technical Report. Computing Research Association. https://cra.org/generation-cs-three-years-later/

[6] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: The Mixed News on Diversity and the Enrollment Surge. *ACM Inroads* 8, 3 (July 2017), 36–42. https://doi.org/10.1145/3103175

[7] Binglin Chen, Matthew West, and Craig Zilles. 2018. How Much Randomization is Needed to Deter Collaborative Cheating on Asynchronous Exams?. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale* (London, United Kingdom) *(L@S '18)*. Association for Computing Machinery, New York, NY, USA, Article 62, 10 pages. https://doi.org/10.1145/3231644.3231664

[8] Binglin Chen, Craig Zilles, Matthew West, and Timothy Bretl. 2019. Effect of Discrete and Continuous Parameter Variation on Difficulty in Automatic Item Generation. In *Artificial Intelligence in Education*, Seiji Isotani, Eva Millán, Amy Ogan, Peter Hastings, Bruce McLaren, and Rose Luckin (Eds.). Springer International Publishing, Cham, 71–83.

[9] Benjamin Clegg, Siobhán North, Phil McMinn, and Gordon Fraser. 2019. Simulating Student Mistakes to Evaluate the Fairness of Automated Grading. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training* (Montreal, Quebec, Canada) *(ICSE-SEET '19)*. IEEE Press, 121–125. https://doi.org/10.1109/ICSE-SEET.2019.00021

[10] Computing Research Association. 2017. *Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006.* Technical Report. https://cra.org/data/generation-cs/

[11] Matt J. Davidson, Brett Wortzman, Amy J. Ko, and Min Li. 2021. *Investigating Item Bias in a CS1 Exam with Differential Item Functioning.* Association for Computing Machinery, New York, NY, USA, 1142–1148. https://doi.org/10.1145/3408877.3432397

[12] Debora de Chiusole, Luca Stefanutti, Pasquale Anselmi, and Egidio Robusto. 2018. Testing the actual equivalence of automatically generated items. *Behavior Research Methods* 50, 1 (Feb. 2018), 39–56. https://doi.org/10.3758/s13428-017-1004-5

[13] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 328. https://doi.org/10.1145/1384271.1384371

[14] Chinedu Emeka and Craig Zilles. 2020. Student Perceptions of Fairness and Security in a Versioned Programming Exam. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 25–35. https://doi.org/10.1145/3372782.3406275

[15] Lena Feinman. 2018. Alternative to Proctoring in Introductory Statistics Community College Courses. *Walden Dissertations and Doctoral Studies* (Jan. 2018). https://scholarworks.waldenu.edu/dissertations/4622

[16] Oscar E. Fernandez. 2021. Second Chance Grading: An Equitable, Meaningful, and Easy-to-Implement Grading System that Synergizes the Research on Testing for Learning, Mastery Grading, and Growth Mindsets. *PRIMUS* 31, 8 (2021), 855–868. https://doi.org/10.1080/10511970.2020.1772915 arXiv:https://doi.org/10.1080/10511970.2020.1772915

[17] Daniel T. Fokum, Daniel N. Coore, Eyton Ferguson, Gunjan Mansingh, and Carl Beckford. 2019. Student Performance in Computing Courses in the Face of Growing Enrollments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 43–48. https://doi.org/10.1145/3287324.3287354

[18] Gwo-Jen Hwang, Hui-Chun Chu, Peng-Yeng Yin, and Ji-Yu Lin. 2008. An innovative parallel test sheet composition approach to meet multiple assessment criteria for national tests. *Computers & Education* 51, 3 (2008), 1058–1072. https://doi.org/10.1016/j.compedu.2007.10.006

[19] Jason W. Morphew, Mariana Silva, Geoffrey Herman, and Matthew West. 2020. Frequent mastery testing with second-chance exams leads to enhanced student learning in undergraduate engineering. *Applied Cognitive Psychology* 34, 1 (2020), 168–181. https://doi.org/10.1002/acp.3605

[20] Neil P. Morris, Mariya Ivancheva, Taryn Coop, Rada Mogliacci, and Bronwen Swinnerton. 2020. Negotiating growth of online education in higher education. *International Journal of Educational Technology in Higher Education* 17, 1 (Nov. 2020), 48. https://doi.org/10.1186/s41239-020-00227-w

[21] George Nakos and Anita Whiting. 2018. The role of frequent short exams in improving student performance in hybrid global business classes. *Journal of Education for Business* 93, 2 (2018), 51–57. https://doi.org/10.1080/08832323.2017.1417231 arXiv:https://doi.org/10.1080/08832323.2017.1417231

[22] Shailendra Palvia, Prageet Aeron, Parul Gupta, Diptiranjan Mahapatra, Ratri Parida, Rebecca Rosner, and Sumita Sindhi. 2018. Online Education: Worldwide Status, Challenges, Trends, and Implications. *Journal of Global Information Technology Management* 21, 4 (2018), 233–241. https://doi.org/10.1080/1097198X.2018.1542262 arXiv:https://doi.org/10.1080/1097198X.2018.1542262

[23] Matthew West Paras Sud and Craig Zilles. 2019. Reducing Difficulty Variance in Randomized Assessments. In *2019 ASEE Annual Conference & Exposition*. ASEE Conferences, Tampa, Florida. https://doi.org/10.18260/1-2--33228

[24] Gili Rusak and Lisa Yan. 2021. *Unique Exams: Designing Assessments for Integrity and Fairness.* Association for Computing Machinery, New York, NY, USA, 1170–1176. https://doi.org/10.1145/3408877.3432556

[25] Mehran Sahami and Chris Piech. 2016. As CS Enrollments Grow, Are We Attracting Weaker Students?. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 54–59. https://doi.org/10.1145/2839509.2844621

[26] Mohammad A Sarrayrih and Mohammed Ilyas. 2013. Challenges of online exam, performances and problems for online university exam. *International Journal of Computer Science Issues (IJCSI)* 10, 1 (2013), 439.

[27] Michael Scott, Tim Stelzer, and Gary Gladding. 2006. Evaluating multiple-choice exams in large introductory physics courses. *Physical Review Special Topics - Physics Education Research* 2, 2 (July 2006), 020102. https://doi.org/10.1103/PhysRevSTPER.2.020102

[28] Mariana Silva, Matthew West, and Craig Zilles. 2020. Measuring the Score Advantage on Asynchronous Exams in an Undergraduate CS Course. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 873–879. https://doi.org/10.1145/3328778.3366859

[29] Michael P. Watters, Paul J. Robertson, and Renae K. Clark. 2011. Student Perceptions of Cheating in Online Business Courses. *Journal of Instructional Pedagogies* 6 (Sept. 2011). https://eric.ed.gov/?id=EJ1097041

[30] Matthew West, Geoffrey L. Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based Online Problem Solving with Adaptive Scoring and Recommendations Driven by Machine Learning. 26.1238.1–26.1238.14. https://peer.asee.org/prairielearn-mastery-based-online-problem-solving-with-adaptive-scoring-and-recommendations-driven-by-machine-learning