Experience Report: Preemptive Final Exams for Computer Science Theory Classes*

Michael Shindler
University of California Irvine
mikes@uci.edu
Matt Ferland, Aaron Cote, Olivera Grujic
University of Southern California
{mferland, aaroncot, grujic}@usc.edu

Abstract

We taught classes enacting a "preemptive final exam" grading mechanism. Students have multiple chances to display knowledge of topics, each being worth a portion of the final grade. The grade earned in each topic is some number of their best attempts out of a higher number of chances.

1 Introduction and Previous Work

As class sizes grow, it is becoming imperative to find more scalable evaluation methods. Recent concerns have included rubrics for large numbers of assignments [3], automated ways to detect students who are at risk of failing [1, 5, 6], support structures [7], and course management for instructors [4].

We introduce a grading mechanism we refer to as a "preemptive final exam," where students can display knowledge before the final exam date, allowing students to skip certain topics on the final, and saving course staff from grading those questions. This policy has precedence, such as some math teachers allowing students with an 'A' on the quizzes to skip the final exam, or allowing a comprehensive final exam to count for a larger percent of the grade if it

is a better score [8]. The latter method has improved the perception of fairness without having a large impact on class averages [8]. A related popular grading mechanism was in use at MIT's 6.034 (Artificial Intelligence) course, allowing students to use quiz scores in place of corresponding sections of their final exam. This method [10] is the basis for our preemptive final exam model. Mastery based classes offer something similar, but typically allow the students more than two chances at each topic [2]. Because of this, they tend not to scale as well to large classes that require proofs and algorithm creation rather than numerical or multiple-choice answers.

Our system is as follows. Every core topic is assigned a portion of the total class weight and can be fulfilled during one or more pre-final exams during the term. The final exam then offers another chance to demonstrate this mastery. Students who demonstrate their understanding early will have "preempted" the topic from their final while students who did not can raise their grade, but only by demonstrated improved understanding of the topic. We believe this achieves a focus on feedback wherein students receive timely remarks about what they did or did not understand about the material, with both a direction to go with their study and a chance to improve that part of their grade.

Students in all of our implementations typically increased their final grade after taking the final exam, often by a significant amount. In both implementations, over two thirds of the students improved their grade by at least 5%, and over a tenth of the students improved their grade by at least 15%.

In this report, we discuss courses that enacted this grading mechanism. We cover a year's worth of teaching at the University of Southern California, using one semester of an algorithms course and one semester of a Discrete Math course.

2 Course Setup

2.1 Introduction to Algorithms

After reviewing fundamental prerequisite concepts, this course typically covers some algorithm design paradigms: dynamic programming, greedy, and divide-and-conquer, followed by a midterm covering those three topics. The midterm has students solve one problem for each paradigm by designing an algorithm using it. We then cover network flow and some computational complexity, and conclude with a final exam.

For our offering, however, the course structure changed slightly. The lecture structure remained intact, as did the placement and purpose of the fundamentals quiz and midterm. We added a quiz in the last week of class, giving a chance to try the post-midterm topics in an exam environment and for credit towards "best of two." There was a final exam with five questions, one on each of the five topics.

40% of the grade was from homework, a fundamentals quiz, and overall midterm performance. The remaining 60% of the grade was separated into 5 sections, with one for each topic. Every topic had exactly two questions used for grading: one question on the final, and one question on the midterm or week 15 exam. The total grade for each topic would be the highest score of the two attempts. Exams contained only one question from each of the 5 topics, and each section was clearly labeled. For more details, please see the online appendix, available at `https://www.ics.uci.edu/~mikes/papers/CCSC_2020_Online_Appendix.pdf`

3 Discrete Mathematics

This class typically has two mid-semester exams, some take-home problem sets, and a final exam. We renamed the two mid-semester exams to be quizzes, and added a third one. Like the algorithms course, this quiz came in the last week of classes and covered material that would be on the final but had been lectured on after the cut-off point of the previous quiz. Each quiz was worth little on its own: the first and third were 5% each, and the second was 10%. The remaining grade that came neither from a quiz sum nor a take-home assignment came from breaking down the core topics to be learned in the course. There was a final exam that provided one last opportunity to demonstrate mastery of each topic.

Unlike algorithms, the topics weren't "best of two." Rather, we gave a number of opportunities for each topic. For example, each quiz had a (non-inductive) proof for the student to write, and the final exam had two. The best three of the five counted for 15% of the students' grade. This is partly because we don't believe that a single question in all of these topics covers the full mastery of the material. By contrast, once one has demonstrated the ability to design an algorithm using, say, dynamic programming, we believe students will retain that mastery, at least at the undergraduate level, and for at least the amount of time between the midterm and final exam.

The topics and breakdowns are presented in the online appendix.

4 Observed Outcomes

4.1 Positive Outcomes

In the algorithms course, 28 students out of 203 had an A without taking the final. Subjectively, the instructor for the course believes each of these students would have also earned an 'A' had the class grading been more traditional. We believe that the reported grade accurately reflects these students' knowledge and experience with the material. While it is conceivable that they, or any student preempting a final exam question, could have done poorly under a traditional model, we believe that the learning curve is such that the

students would be able to earn at least the same score with very high probability. Furthermore, it meant that these students would not be occupying office hours during finals period, freeing course staff for students who still have yet to demonstrate the mastery that they were (hopefully) working on.

Of course, the benefits to students go beyond the fact that some may skip the final exam, as the structure of the class creates a focus on feedback and improvement. Students can see what understanding they lack in an exam and have a chance to demonstrate it again, having learned from their mistakes. This focus on improvement is the point: showing that they learned something they had not previously demonstrated mastery of, and this is reflected explicitly in their grade. Students cannot hide a lack of knowledge on one topic by repeatedly demonstrating mastery of another.

We found the topics most influenced by this policy to be those given earlier in the term, as well as those where the class as a whole performed poorly in initial attempts. This was observed in both classes.

4.2 Particulars of Improvement at the Final

For the algorithms course, 157 students took the final. Not every student needed to attempt every question. For example, the vast majority of the class was happy with their score on the divide and conquer paradigm on the midterm. The breakdown by topic is reflected in the online appendix. The final itself could have been offered in a traditional algorithms course at our university. As such, students who had fewer questions to answer generally would leave earlier and did not appear to have a time advantage over students who had more to answer. This was true even of students with extra time accommodation, where over 90% finished early in both courses. We view the improvement, particularly in dynamic programming, as very encouraging. This is a key concept in algorithm design, and seeing that students are able to incorporate their feedback and learn from it is reassuring.

Just like with dynamic programming, we were encouraged by the improvements we saw on abstract topics from early in the term. Proofs, in particular, feel to many students to be some mysterious art form, yet with sufficient resources and incentive, they are able to develop skills in this area. For the first three topics, this should provide a cause for optimism for their future as computer scientists. Proofs are structured not unlike a computer program [9] and the connection between inductive proofs and recursive thinking should be obvious. There is a similar positive view for students to grasp some fundamental graph algorithms in what is for many of them their first year of university.

In both classes, most students who took the final saw some improvement in their grade. Over two thirds of students that took the final received at least a 5% improvement in their overall grade, and 10% received at least a 15% improvement. A breakdown is available in the online appendix.

4.3 Cautions for the Approach

Each semester we made the usual promise that 90% or above of the points would be sufficient for an 'A' in the class. For the algorithms class, nearly a third of the students finished in excess of 90%. The instructor recognized most of the names and, similar to the opinion of those who preempted the full final, believes that if we were assigning letter grades based on impression of their understanding, each would likely have had an 'A' as well. We believe this is the correct evaluation of each such student. Other instructors may have different views about what should constitute an A and should exercise caution in such promises. At the other extreme, we did not notice any students "optimizing for merely passing," although it could happen. Our view is that a student so optimizing is unlikely to have it secured by the final exam, and cannot rely on the curve for this model in the same way as for a traditionally evaluated course.

We believe our approach works under the assumption that a small number of samples is sufficient to demonstrate mastery. We believe this holds for undergraduate CS Theory, but it may not hold for other subject matters.

Instructors wishing to use this grading mechanism should be cautioned that we observed more regrade requests during the year. It appears some students now perceived the stakes to be higher on exams. Instructors should also be cautioned that there will be a grading crunch at the end, as all pre-finals need to be returned before exam week.

The lower rate of improvement for topics introduced late in the class in both offerings is a cause for concern. It is not clear if students need more time to integrate the difference or if they viewed late quizzes as a nearly free shot at taking part of their final exam twice. It also isn't clear which subset of the students viewed their feedback for such artifacts and if other means of returning the exams to students might be more effective.

Lastly, we caution instructors considering this method to carefully consider the discrete nature of the topics they are teaching. For the algorithm design course, this is less of an issue; in a discrete math class, however, we may want to cross categories more often, such as asking how many Hamiltonian Paths a graph has. Counting and discrete probability also have significant overlap.

5 Summary and Conclusions

We have introduced and piloted the use of a preemptive final exam model for two different Computer Science Theory courses. We believe we have shown that this has allowed students to better and more clearly demonstrate their competency in core topics of the course. The feedback loop created aids in this goal, as does the incentive structure. The mechanism also allowed better use of end-of-term class resources while providing a new incentive for students

to demonstrate mastery early in the term. This technique should be widely applicable across the field of Computer Science.

References

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 121–130, 2015.

[2] J B. Collins, Amanda Harsy, Jarod Hart, Katie Anne Haymaker, Alyssa Marie Hoofnagle, Mike Kuyper Janssen, Jessica Stewart Kelly, Austin Tyler Mohr, and Jessica OShaughnessy. Mastery-based testing in undergraduate mathematics courses. *PRIMUS*, 29(5):441–460, 2019.

[3] John Cigas, Adrienne Decker, Crystal Furman, and Timothy Gallagher. How am i going to grade all these assignments? thinking about rubrics in the large. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 543–544, 2018.

[4] David G Kay. Large introductory computer science classes: strategies for effective course management. *ACM SIGCSE Bulletin*, 30(1):131–134, 1998.

[5] Soohyun Nam Liao, Daniel Zingaro, Christine Alvarado, William G Griswold, and Leo Porter. Exploring the value of different data sources for predicting student performance in multiple cs courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 112–118, 2019.

[6] Soohyun Nam Liao, Daniel Zingaro, Kevin Thai, Christine Alvarado, William G Griswold, and Leo Porter. A robust machine learning technique to predict low-performing students. *ACM Transactions on Computing Education (TOCE)*, 19(3):1–19, 2019.

[7] Mia Minnes, Christine Alvarado, and Leo Porter. Lightweight techniques to support students in large classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 122–127, 2018.

[8] Ben Stephenson. The impacts of providing novice computer science students with a second chance on their midterm exams. *Journal of Computing Sciences in Colleges*, 27(4):122–130, 2012.

[9] Daniel J Velleman. *How to prove it: A structured approach*. Cambridge University Press, 2019.

[10] Patrick Henry Winston. Skills, big ideas, and getting grades out of the way. `http://web.mit.edu/fnl/volume/204/winston.html`.