— Independent Work Report, Spring 2018 —

A Modular Framework for Developing, Deploying, and Evaluating Game-Theoretic Strategy Design Exercises

Andrew Wonnacott
Advisers: Professor Dave Walker, Professor Matt Weinberg

Abstract

Student interest in Princeton's undergraduate course in algorithmic game theory, "Economics and Computing" (COS 445), has grown rapidly in recent years. This course employs small simulations to develop students' ability to use theoretical tools to analyze more real-world situations. These strategy design exercises require considerably more software infrastructure than would a usual theoretical problem set, including handout code for student self-evaluation and software autograders. While some frameworks for automated grading exist, none surveyed provided the specific features desired for repeatedly developing very similarly structured assignments and evaluating student submissions in the context of each other. In order to facilitate more automated grading of these assignments in this and future semesters, I have developed a framework of modular, extensible tools for building all of the necessary compoenents oif these strategy design assignments. Concurrently, in my role as a grader for COS 445, I used the framework to develop and evaluate four strategy design assignments, using assignment specifications developed with Professor Weinberg. This provides feedback from real student and instructor users which was used to improve the design. Empirical results demonstrate the importance of quality tooling to effective use of instructor time, low broken submission rates, and high student satisfaction.

Introduction

My work builds upon existing programming-based game theory exercises assigned in an existing Princeton course. The intention of these strategy design exercises is to teach students how to use game theory to analyze a real world situation and work rationally, within healthy incentive structures where students do not try to hurt each others' performance. Each past offering of the course has rebuilt these assignments from scratch, and all associated instructor tooling, resulting in unnecessarily high allocation of instructor time to grading. Additionally, the course has grown rapidly to become the single largest theoretical computer science course ever taught at Princeton.[1][2] In order to eliminate unnecessary annual replication of previous coursework development, improve course performance at scale, and improve the course experience for the students, I develop an extensible framework to improve these strategy design exercises.

Four Princeton University departments offer courses in game theory: economics, politics, mathematics, and computer science. While all of these courses teach the core concepts of game theory, each also connects this core to relevant examples and topics in its department. For instance, in the computer science department, COS 445, "Economics and Computing", places emphasis on the connections between game-theoretic results and algorithms, software, and other aspects of computer science. This is represented in the course topics, which focus on the relevance of game theory to matchings, auctions, cryptocurrencies, and other multi-actor computational systems.[3] It is also represented in the design of the problem sets, which contain, in addition to traditional proof-based exercises, programming-based exercises which connect course topics to prerequesite computer science skills. These programming-based exercises take a common form.

Students are presented with an iterated game, described both in the assignment handout and as a Java interface. A classic iterated game is the iterated prisoner's dilemma[4], in which two players must collaborate to receive the best outcome, but are each incentivized to defect, which results

[1] registrar data here
[2] More students complete COS 340 in a given year, but it is taught in both fall and spring semesters.
[3] https://www.cs.princeton.edu/ smattw/Teaching/cos445sp18.htm
[4] Figure: PD game -> Java iterface

in a less preferred outcome, and have acecss to the other player's play history when making their own decision. While this game is well-studied and models some real-world incentive structures, the course also draws on various real-world phenomenon to design iterated games, including, from semester to semester, a college admissions game from the perspective of stable matchings, a search advertisement bidding game using auction theory, an election informed by voting theory, a gerrymandering-prevention simulation inspired by fair division, and exploitation of a fictional cryptocurrency by analyzing blockchain incentives. These examples are not exhaustive, but illustrate the variety of course topics with potential for translation to iterated games.

Students then craft strategies to play the specified game, based on their own analyses of the game's equilbirium, and based on their own priors over the strategies which will be designed by their peers. These strategies are evaluated on both on their design quality and on their empirical performance. While design quality is evaluated on a purely individual basis, empirical performance is measured by simulating the play of the collection of submitted student strategies. For instance, for the iterated prisoner's dilemma, every student strategy plays one full game (consisting of many iterations of the prisoner's dilemma) against each other strategy, and strategies are scored based off the average payoff they achieved. For the search advertisement game, all the student strategies are concurrently asked to bid on the same search terms, and are scored based on their revenue from ad clicks (where each bidder knows the click-through rate for a term prior to bidding on that term) less their expenditures on advertising. To create a more realistic simulation, students are graded solely the performance of their own strategy, not on a curve of the performance of their peers. That is, student grades on these strategy design exercises are not zero-sum.

While the strategy design exercises assignments are popular with students[5], they also demanded considerable instructor resources be dedicated to building and grading software resoruces, which have historically been significantly rebuilt for each assignment. Worse still, due to course staffing changes, this work has not been preserved between semesters, resulting in inefficient allocation of department resources to producing new exercises without the institutional memory or software

[5]https://reg-captiva.princeton.edu/chart/index.php?terminfo=1184&courseinfo=012095 requires CAS login

resources previously developed. Due to the limited tooling provided in past semesters for students to evaluate their submissions, many submissions did not compile or caused runtime exceptions, requiring considerable additional instructor resources.

The goal of this project is to discover to what degree the development of new strategy design exercises can be simplified or automated, to improve the student resources available during the completion of these exercises, and to create institutional memory and decrease year-to-year repetition of the same work. This was accomplished by designing a reuasble framework which embodies the commonality between strategy design exercises, allowing instructors creating new exercises to change only the necessary, assignment-specific details, and which will include more advanced automated grading tools than previously existed for the course. Within this framework, I have built a collection of utilities to facilitate students' evaluation of their own strategies and to decrease strategy iteration overhead.

Through this work, I aim to improve the quality of computer science theory education, from the perspectives of both the students and faculty associated with COS 445. Quantitative metrics, including software failure rates and instructor time usage, and qualitiative metrics, including student and instructor satisfaction, both illustrate the improvements to the course and emphasize the importance of quality tooling to quality programming-based assignments.

Related Work

This work was developed to replace previous software design exercises assigned to COS 445 students[6] in Spring 2017.[7] In one problem set, students developed and implemented strategies for three different strategy design exercises, related to three different classical games: Prisoner's Dilemma, Centipede, and Ultimatum. In a later problem set, an exercise placed students in the role of bidders in a generalized second-price auction.[8]

[6]including the author

[7]https://web.archive.org/web/20170916042726/https://www.cs.princeton.edu/ smattw/Teaching/cos445sp17.htm

[8]an auction of multiple items, one after another, where the winning bid pays the second highest bid. This is the auction format used by Google to sell sponsored search auctions.

Both problem sets were implemented and graded by Cyril Zhang[9], a graduate student TA for the course. Cyril developed interfaces, dummy implementations, and grader software for each strategy design exercise.[10] However, this existing implementation of grader software for strategy design assignments is not modular and has to be completely rewritten for each assignment, and was designed for grader use only. The student interfaces were documented, but the grader software was not documented for future re-use. Additionally, some of the scripts and notebooks used by Cyril were not preserved and were not accessible as resources. Thus, while these resources served as inspiration for this work and provided some assignment-specific code to my work developing specific assignments, I did not build directly on Cyril's code.

Previous to Spring 2017, COS 445 was taught by Professor Mark Braverman in Fall 2012 and Spring 2014[11]. Though no documentation or archives of the course homepage from these semesters is available online or in the Internet Archive, I recovered some resources from the course account on the "Cycles" cluster of the Princeton Computer Science department and from the source code respositories used by the course staff. In both semesters, Yonatan Naamad[12] was responsible for software development.

The only resources available from Fall 2012 were collected from the course Google Code repository.[13] This repository contains incomplete Python skeleton code for a basic tournament. This was uploaded to the current course repository,[14] but was not reused.

The Spring 2014 iteration of the course was developed on a private GitHub repository.[15] Through a clone of this repository stored on Cycles, I discovered that the course had PHP and Java implementations of strategy design exercises for Prisoner's Dilemma and another auction setting. The Java code consisted of student handouts of interfaces and some testing code, but the testing code was not designed to be automated for grading and was not modular: it tested manually-enterred

[9] https://github.com/cyrilzhang

[10] https://github.com/PrincetonUniversity/cos445-hw/tree/s17 requires invite: contact author or department

[11] https://precourser.io/course/1184012095 requires Princeton CAS authentication

[12] https://github.com/ynaamad/

[13] https://code.google.com/archive/p/cos445-scripts/

[14] https://github.com/PrincetonUniversity/cos445-hw/tree/f12 requires invite: contact author or department

[15] https://github.com/ynaamad/cos445

strategies against each other and assignment-specific details (e.g. game structure and payoffs) were integrated within the simulation code. As a design decision I dismissed the available PHP code, which was used for course grading, due to language unfamiliarity. These resources were uploaded to the current course repository,[16] but were not reused, in part because they were located and examined only when I was given access to the course site in order to deploy my project.

Beyond COS 445 at Princeton, many universities have taught computer science courses in algorithmic game theory. However, a survey of several universities' courses, including courses at Harvard[17], Stanford[18], Cornell[19], U. Washington[20], Yale[21], U. Penn[22], Georgia Tech[23], Carnegie Mellon[24], and Duke[25] identified only one other course with similarly structured assignments.

Harvard's Computer Science 136, "Economics and Computation",[26] assigns two programming problem sets. The first requires implementation of a peer-to-peer filesharing system similar to Bittorrent, a protocol in which incentives parallel Prisoner's Dilemma. However, the assignment is not a competition and student solutions are not differentiated based on strategic properties. The second assignment is a strategy design exercise built on the generalized second-price auction, in which student strategies vary and are graded based on their performance in a competitive environment. This is very similar to both the Spring 2014 and Spring 2017 auction exercises of COS 445 at Princeton. However, the grading structure of this assignment is quite different from the grading structure of the COS 445 strategy design exercises, and the implementation is in Python. While comparison between COS 445 assignments and CS 136 assignments in a future project might provide some pedagogical value, such work would be out of scope for this project. Given that these assignments are implemented in Python, only handout (and not grading) code is available, and the

[16]https://github.com/PrincetonUniversity/cos445-hw/tree/s14 requires invite: contact author or department
[17]https://beta.blogs.harvard.edu/k108875/
[18]https://theory.stanford.edu/ tim/364a.html and https://theory.stanford.edu/ tim/f16/f16.html
[19]https://www.cs.cornell.edu/courses/cs6840/2014sp/
[20]https://courses.cs.washington.edu/courses/cse522/05sp/
[21]https://zoo.cs.yale.edu/classes/cs455/fall11/
[22]https://www.cis.upenn.edu/ aaroth/courses/agtS15.html
[23]https://web.archive.org/web/20131002121909/https://www.cc.gatech.edu/%7Eninamf/LGO10/index.html
[24]https://www.cs.cmu.edu/ arielpro/15896s16/index.html
[25]https://www2.cs.duke.edu/courses/spring16/compsci590.4/
[26]https://beta.blogs.harvard.edu/k108875/assignments/

assignments are graded differently, these resources were dismissed as a base for development of new resources for COS 445 at Princeton.

Many universities, including Princeton, maintain extensive systems for the automatic submission and processing of student assignments. Such resources are useful in the general case of developing new programming-based assignments, but no such resources replicate the behavior implemented in this project. Most autograding systems are designed with an assumption that student submission are graded in a sandbox, isolated from other student submissions. Such systems are not applicable to this project, since the strategy design exercises involve evaluation of student strategy performance in a common environment. However, existing resources for assignment submission and student feedback in use by Princeton are used in this assignment. I integrated some CS DropBox[27] functionality where relevant, and modelled the leaderboard structure after review of the leaderboard implementation used in other Princeton CS classes. Generally, however, no such tools would leverage the high degree of structural similarity between strategy design exercises, and a dedicated project for these exercises will allow for more efficient development and evaluation.

Approach

Cyril Zhang completed the TA requirement for the PhD program last fall, so at the start of this semester it was unclear who among the graduate and undergraduate course staff would be responsible for maintaining these assignments. Thus, the course needed not just long-term work to improve student and instructor resources, but also short-term work to maintain and debug the same resources. These harmonizing requirement suggest that the product should consist of one extensible implementation of all the reusable components for software design exercises, developed iteratively in conjunction with its usage in the course. This software would naturally grow throughout development as demanded by the needs of the course, and the final product will naturally lower-bound the maximum degree of automation and modularity possible.

Before the first problem set of the course was assigned in this semester, Matt and I designed and

[27] https://csguide.cs.princeton.edu/academic/csdropbox

I implemented a handout interface and tools to allow students to test their strategies against each other. The game simulates college admissions using stable matchings. While I implemented this handout with the explicit goal of producing a tool to test out multiple different student strategies in the specified application environment, I followed the principles of modular software design covered in previous courses.[28] I factored out the code which implemented the Gale-Shapley deferred acceptance algorithm for stable matchings from the code which amortized results from multiple randomized trials, and from agglomerated these results to output performance statistics for students. Thus, when it came time to implement the second strategy design exercise, I was able to reuse code which processed results of trials, even though the nature of the trails changed from college admissions to elections. Additionally, when it came time to grade the problem set, I was able to modify only the post-processing step to compute grades. Thus, creating a new assignment requires constant work in the number of framework extensions, and creating a new framework extension requires constant work in the number of assignments.

The result of this process duplicates significant functionality from last year's assignments. However, because of conscious design decisions to factor our repeated code from assignment implementations, it accomplishes what previous implementations had not. It will be modular, as described above, so that new extensions such as additional tools or interfaces can be written for strategy design exercises. Throughout the semester, I can leverage my role as an undergraduate course assistant for COS 445 during the current semester in order to identify and produce such extensions. And, it will be reusable, both for the same assignments in future years, and for new strategy design exercises implemented by future course staff with lower overhead.

In order to increase the reusability, I elected to store it in a University-owned private GitHub repository. This ensures that, even after I have graduated, future course staff will have access to the project. Additionally, I documented the repository structure and each program's role in the repository, as documented later in this document.

[28] Kernighan and Pike chapter 4

Implementation

The first key decision in the implementation of these programming assignments was language selection. Many students suggested the assignments be implemented in Python,[29], and this would be preferred by the course staff as well. Members of the Mechanism Design group are not expected to have working proficiency with Java, but many have enough experience with Python to support students. Additionally, Python's conciseness and ability to load classes from user input would be useful when implementing tools which are parametrized on user-input strategy names. Unfortunately, COS 445 does not have Python experience as a course prerequisite, only Java experience. Additionally, survey data from other course surveys suggests that 90% of students report themselves fluent in Java after the Princeton introductory sequence, compared to 30% for python.[30]

Thus, the whole system under consideration is constrained: it must support student strategy implementations in Java. Given this constraint, and lacking any other language selection pressures, I elected to implement as much of the assignments as possible in Java, with limited usage of scripting languages such as make, Python, and bash. These scripting languages were used where Java would not suffice and only for tasks for which implementation details were irrelevant to student assignment comprehension.

* how to modularize that * what is reusable * what isnt reusable how to grade people not against peers

* what extensions

Results

As the goals of a project naturally define evaluation criteria, I have evaluated my project on the basis of those goals.

- discover modularity and decrease assignment to assignment rewriting

[29]per survey 1 results

[30]COS 333 survey data

  - code rewriting measurements
  - instructor time usage
- improve student resources
  - list resources created
  - polling data about student usage
  - empirical results about student error rates
- create institutional memory and decrease year-to-year rewriting
  - cant measure until next year, but set up for it
- increase student and instructor satisfaction
  - polling data about student usage

Conclusions

Through this work, I aim to improve the quality of computer science theory education, from the perspectives of both the students and faculty associated with COS 445.

prisoner's dilemma + search auctions

Future Work

Some of the implementations of my infrastructure could be simplified using ClassLoaders, which would replace some of the Python code. This improvement was inspired by the implementation of ClassLoaders in the Spring 2014 student handout code.

Earlier assignments were built off of earlier versions of the infrastructure and will need to be ported to newer versions when they are reused: do next year

will be reused in two years without my oversight

other universities? matt through academic connections

Acknowledgements

* matt for assignment design lead * course staff, especially Andreea, Ariel, Evan, Heesu, Leila, Divya, Matheus, Wei, Dylan, for grading student writeups, providing me feedback and passing along student feedback * david walker for iw sem (helping me stay organized and on track, helping me identify my topic) * iw sem peers Greg, Alex, Mel, Noah for feedback * jeremie for creating github repo * kate for providing writing help and moral support

Ethics

* assignments used in the course were used by the executive decision of Professor Weinberg, * potential conflicts of interest * honor code