Fair Allocation of Columbia Computer Science Course Seats

Jessica Shi
jjs2295@columbia.edu

Sarah Gu
swg2121@columbia.edu

Bailee Hanson
bh2832@columbia.edu

PJ Yoo
pgy2101@columbia.edu

Abstract

Approximate Competitive Equilibrium from Equal Incomes (A-CEEI) is an equilibrium-based solution concept for fair division of indivisible items to agents with combinatorial demands. One setting in which A-CEEI has seen practical application is in the allocation of courses to students. In this project, we explore the course allocation problem experimentally using real course registration data from Columbia's Department of Computer Science. From this data, we model student preferences over schedules of Computer Science classes, and examine the performance of the current state-of-the art implementation for computing A-CEEI. On a problem instance with 50 students and 11 courses, we are able to compute an A-CEEI that achieves exactly zero clearing error in $\sim$20 minutes.

1 Introduction

1.1 Motivation

With limited capacity in highly-demanded classes, assigning course seats to students can be a difficult challenge for administrators in higher education. This is certainly true for students in the Computer Science Department at Columbia University, where hundreds of students must choose from limited courses on machine learning and other high-demand topics. After meeting with the advising/enrollment team for the Computer Science department, we found that there is no organized process for allocating students to Computer Science courses. The current process is as follows: the department allows students to sign up for their desired courses, and after selecting a course the student is automatically moved onto the waitlist, regardless of course capacity. Since Columbia students are allowed to join at most four waitlists, students can attempt to register for at most four COMS courses (typically less, since they must register for other non-COMS courses simultaneously). Periodically throughout the registration period, faculty members within the Computer Science department will manually, and sometimes even randomly, move students from the waitlist into each course.

The decision process is unsystematic but usually considers: 1) what degree program students are in (department and degree type); 2) how many semesters left until their graduation 3) scheduling considerations and other one-off scenarios. As students, we have all experienced the confusion and frustration we face when we are unable to enroll in required courses or courses that are pertinent for our academic and/or professional careers. In this paper we look to use solutions from existing literature; more specifically, we implement the algorithm developed in [Budish et al., 2023] for course seat allocation in the Computer Science Department at Columbia University.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

1.2 Existing solutions

In a combinatorial assignment problem, a set of indivisible objects is to be allocated to a set of agents with different preferences. Some motivating examples include the assignment of shifts/tasks to employees, players to sports teams, and seats in courses to students. In this project, we focus on the course allocation setting - how should seats in overdemanded courses be allocated?

One common solution concept for fair division problems is the competitive equilibrium from equal incomes (CEEI). CEEI is a solution concept that satisfies Pareto optimality, envy-freeness, and proportionality. For complex preferences and indivisibilities, however, CEEI is not guaranteed to exist. To fix the existence problem, Budish [2011] introduced the approximate CEEI (A-CEEI) mechanism, and proved that an approximate equilibrium always exists when small perturbations to agents' incomes are allowed. In addition, this solution concept is approximately strategyproof, fair, and efficient.

Although A-CEEI is PPAD-complete, there exist implementations that can compute solutions when problem instances are not too large. A commercial implementation of A-CEEI called Course Match is successfully used in the course allocation setting at many professional schools including Wharton, CBS, Toronto Rotman, and Dartmouth Tuck. Recently, Budish et al. [2023] developed a new implementation for computing A-CEEI which outperforms Course Match by several orders of magnitude.

1.2.1 Comparison with previous state-of-the-art

The previous state-of-the-art algorithm (henceforth *Course Match*) for computing A-CEEI is summarized in [Budish et al., 2017]. At a high-level, the algorithm proceeds through three steps: 1) finds a price vector to minimize clearing error by local search, 2) raises individual prices to eliminate oversubscription, and 3) reduces undersubscription by augmenting student budgets. The main part of the algorithm is the first step, and one of the key techniques of Course Match was to leverage individual price adjustments. Briefly, the price vector search from Course Match works as follows. Prices and budgets are initially randomized. On each iteration, Course Match begins with a price vector and checks if such a price vector results in allocations with zero clearing error. If not, a "neighborhood" of price vectors is generated via both individual price adjustments and full tatonnement updates. First, Course Match eliminates prices from this neighborhood that are effectively equivalent, i.e. if such a price vector produces equivalent student demands, it is deemed equivalent. After eliminating redundant prices, Course Match searches among the neighborhood for the price vector that minimizes clearing error, and repeats the local search method.

Our implementation will focus on the algorithm from Budish et al. [2023] (henceforth *fast*). One key difference of the fast algorithm is that it disregards perturbing prices and mainly relies on budget perturbations. Indeed, the novelty of the algorithm is that it can find "optimal" budget perturbations for each student. Here, budgets are not perturbed at random; they are perturbed one time to guarantee an approximate equilibrium, and a second time to account for tie-breaking. A clever rule for updating prices allows the algorithm to quickly enumerate students' demand bundles; then, it solves for a combination of demand bundles that minimizes clearing error through an integer program.

The next key difference of the fast algorithm is that it no longer uses individual price updates. After solving the integer program for minimal clearing error, the algorithm updates prices through a full tatonnement. In particular, it uses the excess demand as a gradient, and updates all of the prices according to a specified step size. Using optimal budget perturbations and full tatonnement price updates, Budish et al. [2023] reports that an A-CEEI can be computed very efficiently.

1.3 Overview

We generate an artificial dataset that accurately reflects the population of students who want to enroll in computer science courses at Columbia University. This dataset will contain information on the students' utility functions u, course capacities c, and initial budgets b_0.

2 Data Generation

We have a dataset containing information on the number of students that are waitlisted for every course in the Columbia University Computer Science Department grouped into types of students for the upcoming Spring 2024 semester. These "types" of students describe what sort of degree program they're in and how long they've been enrolled. There are 49 different groups. As it relates to course selection, we expect students in the same group to behave in a similar way, such that they will maximize their expected utility given the courses available to them. Using this as a baseline for student course preferences, we assume the courses that a student waitlisted are their top choices for courses.

2.1 Full dataset

As our first approach, we artificially generate demand sets for each type group. Since we don't have the actual information on how computer science students would rank the courses, we use some estimates from the data provided. There are a few stages of the sampling.

1. We sample values of f_{ik}, the number of classes student i in group k would rank.
2. A ranking of individual classes. This will be the basis of building out the utility of bundles.
3. Computing the ranking of bundles.

For each group g_k, there are n_k student total. For each course c_j, where there are m courses total, there are w_{jk} students waitlisted and e_{jk} students enrolled from the group k. The total demand for a course from group k is $d_{jk} = w_{jk} + e_{jk}$. The total enrollment and waitlist count for group k is $D_k = \sum_{j=1}^{m} d_{jk}$.

Although we believe this generates a somewhat accurate artificial dataset for all students, we eventually reduce the size of the problem as detailed in the next section.

2.1.1 Stage 1

For each group k, we compute $R_k = \dfrac{D_k}{n_k}$ as the average enrollment or waitlist total for students in group k. Using this we can sample f_{ik}, as a Poisson distribution where $\lambda = R_k$. This gives the desired nonnegative integer values as the number of classes they rank. This does not need to have an upper bound, since some students may want to rank even more courses than the 4 waitlists they could possibly join.

2.1.2 Stage 2

Let $p_{jk} = \dfrac{d_{jk}}{D_j}, \forall (j,k)$ be the probability that a student in group k will choose course j given that they are enrolling or waitlisting a course.

For each student i in group k and for each rank spot from 1 to f_{ik}, we can sample the probability distribution $P_k = [p_{1k}, \cdots, p_{mk}]$ to get a list of indices of the courses that they rank. We take the 1st to be first choice and sample without replacement to ensure no repeats. This is a ranking of individual courses, lets call this set of indices to be I.

2.1.3 Stage 3

For simplicity, the highest ranked course has utility score f_{ik}, and the second-highest ranked course has utility $f_{ik} - 1$, and the last ranked course has utility 1. Let this utility of individual courses $\in [m]$ be

$$u_{ijk} = \begin{cases} f_{ik} - g & \text{if course j is ranked at place g} \\ 0 & \text{otherwise} \end{cases}$$

A bundle is defined by x_{ikq} where

$$x_{ijq} = \begin{cases} 1 & \text{if course j is in the bundle} \\ 0 & \text{otherwise} \end{cases}$$

The total utility for a given bundle x_{ikq} is $U(x_{ijkq}) = \sum_{j=0}^{m} x_{ijkq} u_{ijk}$. Each bundle can contain at most M courses. We can tiebreak between bundles with equal utility by the following two rules: 1) More courses is better. 2) The bundle containing the higher individually ranked course is ranked higher.

With these tie-breaking rules and an ordering from highest to lowest utility, we have the resulting demand set for student i in group k: $X_{ik} = [x_{ik1}, \cdots, x_{ikQ}]$ where Q is the total number of possible combinations.

2.2 Reduced Dataset

After examining the waitlist and enrollment data, we found that only a small number of classes were overenrolled, and reformulated the problem over these classes (we owe this idea to a remark from [Budish, 2011], where "if some courses are known to be in substantial excess supply, we can reformulate the problem as one of allocating only the potentially scarce courses").

There were only 11 courses that had less capacity than the number of enrollments and waitlists. Further simplifications were made with the reduced number of courses. The original dataset had 2405 students, however, we reduced the size to 50 to be able to reasonably run the algorithm on our laptops. The notion of type groups of students is dropped, and all students sample 3 courses only from the 11 courses. Table 1 provides information on the these 11 courses.

Table 1: Course data

Class	Enrollments & Waitlists	Capacity	Ratio	Probability	Reduced Capacity
CSEEW3827	300	280	1.071429	0.115252	15
CSEEW4119	122	100	1.220000	0.046869	5
COMSW4170	492	398	1.236181	0.189013	22
COMSE6424	38	30	1.266667	0.014599	1
COMSW3203	516	360	1.433333	0.198233	20
COMSW3132	88	60	1.466667	0.033807	3
COMSW3261	457	300	1.523333	0.175567	16
COMSW4460	66	40	1.650000	0.025355	2
COMSE6111	173	75	2.306667	0.066462	4
COMSE6178	71	30	2.366667	0.027276	1
COMSW4732	280	100	2.800000	0.107568	5

As a result, our problem instance contains 11 courses and 50 students. Students had a maximum bundle size of 3. Course capacities were scaled appropriately to make overenrollment a legitimate issue shown in the "Reduced Capacity" column of Table 1. The process of generating bundles follows 2.1.2 and 2.1.3, without the notion of type groups, sampling from the "Probability" column of Table 1 to get a ranking of three individual courses. Hence, our result is a demand set of 7 bundles for each of the 50 students. The first bundle having all 3 of their ranked courses, the following 3 having a combination of 2 courses, and the last 3 having 1 course in each bundle.

3 Course Allocation Algorithm Implementation

3.1 Preliminaries

Here we drop the notion of the groups of students so i is indexing over all students in the problem.

We use the market described in [Budish et al., 2023]. The course allocation market $(u = (u_i)_{i=1}^n, c = (c_j)_{j=1}^m)$ consists of m courses where each course $j \in [m]$ has capacity c_j and n students where each student $i \in [n]$ has a utility function u_i over each course bundle.

Allocation: For a market (u, c) with prices $p = (p_j)_{j=1}^n$, budgets $b = (b_i)_{i=1}^n$, the allocation function $a = (a_i)_{i=1}^n$ is defined as

$$a_i(u, p, b) = \arg \max_{x \in 2^{[m]}, \langle p, x \rangle \leq b_i} u_i(x)$$

In other words, a student will choose the bundle that has the highest utility and is within their budget range.

Excess demand: The excess demand $z = (z_j)_{j=1}^m$ is

$$z_j(u, c, p, b) = \sum_{i=1}^{n} a_{ij}(u, p, b) - c_j$$

which is essentially the difference between the total students enrolled in a course and a courses capacity. If $z_j < 0$, it is under-enrolled, $z_j > 0$ it is over-enrolled.

Clipped excess demand: $\tilde{z} = (\tilde{z}_j)_{j=1}^m$ as

$$\tilde{z}_j(u, c, p, b) = \begin{cases} z_j(u, c, p, b) & \text{if } p_j > 0 \\ \max\{0, z_j(u, c, p, b)\} & \text{if } p_j = 0 \end{cases}$$

Since we don't need to consider under-enrollment, we can ignore these in the total error.

Market clearing error: $||\tilde{z_j}(u, c, p, b)||_2$

3.2 Basic Algorithm

The input consists of the student's preferences u, course capacities c, and initial budgets b_0. There are also a few input parameters: the step size for price changes δ, the maximum budget perturbation ϵ, and type t of the EF-TB constraint (0 for no EF-TB constraint, 1 for the EF-TB constraint, and 2 for contested EF-TB. At each iteration, we optimize the budget perturbation given the prices in terms of finding the optimal market clearing error. Prices are updated using the tatonnement rule– if the course is overallocated increase the price, under-allocated lower the price. The algorithm terminates when market clearing error is zero.

3.2.1 Pseudocode

```
ALGORITHM:
set p = 0
repeat
    budget perturbation, find budgets b such that market clearing error
    is minimized under the constraints
    if market clearing error = 0
        terminate with p*=p, b*=b
    else
        update p by tatonnement rule
end repeat
```

The constraints for the ϵ-budget perturbation are as follows:

- The maximum perturbation $||b - b_0||_\infty \leq \epsilon$
- The allocation is EF-TB or contested EF-TB
- $||b||_1$ is the tiebreaker, it is minimized amongst multiple optimal solution

The tatonnement rule is setting p to $p + \delta\tilde{z}(u, c, p, b)$, this increases the prices for courses that are over-allocated, since we are initializing with $p = 0$, there's no need to decrease price for the under-allocated because it will stay as 0.

3.2.2 How to compute budget perturbation

For a fixed price vector, the demand of the student will only change when the budget is the sum of prices or in other words, the sum of prices is a multiple of δ the step size to change prices. Therefore, we can partition the interval of Student i's possible budgets $[b_{0i} \pm \epsilon]$ into $k_i \leq \frac{2\epsilon}{\delta} + 1$ sub-intervals $(\underline{b_{il}}, \overline{b_{il}}]$ such that i's demand bundle a_{il} is constant on each sub-interval. So we can compute the set,

$$\{(a_{il}, \underline{b_{il}}, \overline{b_{il}}) : \text{student } i\text{'s demand is } a_{il} \text{ for every budget in } [\underline{b_{il}}, \overline{b_{il}}]\}_{l=1}^{k_i}$$

where $\underline{b_{il}}, \overline{b_{il}}$ are multiples of δ in $b_{0i} \pm \epsilon$

3.2.3 Integer Program

The following integer-linear program (ILP) solves over student demand bundles to find the optimal ϵ-budget perturbation, originally found in [Budish et al., 2023]:

$$\begin{aligned}
& \min ||z||_1 && && \text{(Minimize clearing error)} \\
\text{s.t.} & \sum_{i\in[n]} \sum_{l\in[k_i]} x_{il} \cdot a_{ilj} = c_j + z_j && \forall j \in [m], p_j > 0 && \text{(Clearing error: } p_j > 0\text{)} \\
& \sum_{i\in[n]} \sum_{l\in[k_i]} x_{il} \cdot a_{ilj} \leq c_j + z_j && \forall j \in [m], p_j = 0 && \text{(Clearing error: } p_j = 0\text{)} \\
& \sum_{l\in[k_i]} x_{il} = 1 && \forall i \in [n] && \text{(1 schedule per student)} \\
& x_{il} \in \{0, 1\} && \forall i \in [n], l \in [k_i] && \text{(Integral allocations)}
\end{aligned}$$

Note that the equality constraint for classes with $p_j > 0$ penalizes both over and underenrollment (i.e. "good" courses should not leave any seats open). In practice, these equality constraints often led to ILP infeasibility issues. To resolve these issues, we relaxed these inequality constraints. In our implementation, all of the clearing-error related constraints are formulated in the same way as classes with $p_j = 0$. Thus, underenrollment is no longer penalized in our formulation.

To control for EF-TB and contested EF-TB, we would add the following constraints.

$$x_{il} + x_{i'l'} \leq 1 \quad \text{if} \begin{cases} \exists S \subseteq a_{i'l'}, u_i(a_{il}) < u_i(S) & \text{(classic EF-TB)} \\ \exists S \subseteq a_{i'l'} \cup \{k : p_k = 0\}, u_i(a_{il}) < u_i(S) & \text{(contested EF-TB)} \end{cases}$$

4 Results

In this section, we discuss the details of our problem instance and results of the algorithm's performance. We show results from running the algorithm on our reduced dataset described in Section 2.2.

4.1 Parameters

A challenge that we faced was calibrating the parameters ϵ, δ to fit our problem. Although Budish cites several examples of suitable parameters for running the algorithm for different school sizes and numbers of classes, we found that our problem did not match any of those inputs well. If delta is too large, an infeasibility issue arises if the step increase in price causes a previously overallocated class to become an underallocated class. However, if delta is too small, the algorithm takes a lot longer time to reach the proper prices. After experimenting with this tradeoff, we found that $\delta = 0.00005$ would not cause these infeasibility issues. In addition, we implemented a warm start for the first 100 iterations with $\delta = 0.0001$. As recommended by Budish, we settled on using $\epsilon = 0.01$ for the budget range.

Additionally, we introduce different initial budgets for different grades of students to give more senior students a higher priority to achieving a higher utility. This reflects how course assignment is currently done in the department. We randomly assign the 50 students into 4 equal groups, freshmen, sophomores, juniors, and seniors given budgets $0.97, 0.98, 0.99, 1.0$ respectively.

4.2 Computational Performance

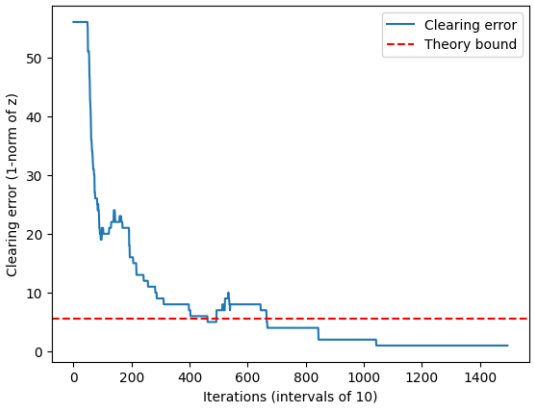

Figure 1: Plot of clearing error at iteration t for the fast algorithm.

One notable claim of [Budish et al., 2023] is that the algorithm can find solutions with exactly zero clearing error, even though [Budish, 2011] only guarantees a small clearing error. For a problem instance with m courses and a maximum bundle size of k, [Budish, 2011] guarantees a clearing error less than $\frac{\sqrt{\sigma \cdot m}}{2}$, where $\sigma = \min(2k, m)$. This clearing error guarantee is the theory bound displayed in Figure 1.

The algorithm achieves zero clearing error on this problem instance in roughly 20 minutes. The tatonnement price updates occasionally result in to local increases in clearing error, but the error continues decreasing, reaching well below the theoretical bound from [Budish, 2011] and eventually achieving zero, as shown in Figure 1.

4.3 Final allocations & prices

Every student received one of their 7 bundles, and every class was filled to its capacity. Additional information on the final allocations to students can be found in the Appendix. We compute which ranked bundle each student received and show the results in Figure 2. Only one student received their top ranked bundle, whereas 44% received their 5th ranked bundle, which is the bundle containing only the student's first choice class. There were 94 seats for 50 students, so on average students got allocated 1.88 courses, so it is logical that many students only get allocated one course. 50% of students got a bundle of 2 courses. No students received their last choice course.

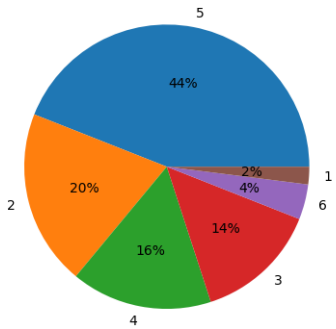

Figure 2: The rank of the bundle that students received

By the tatonnement rule, over time the price of each class converges to a value when it reaches zero excess demand. We can observe when each course hits this threshold in Figure 3. About half of the courses hit their zero clearing error early on, whereas others took a much longer time to converge.

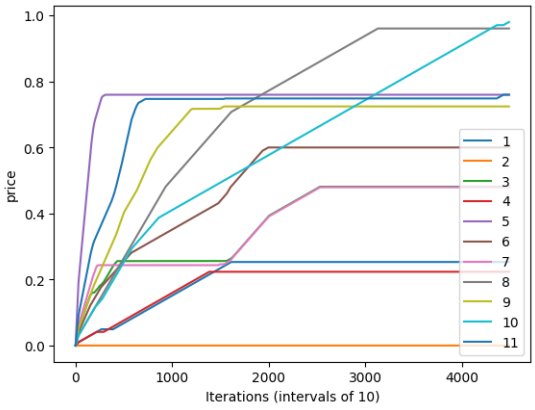

Figure 3: Prices over time

The final course prices are shown in Table 2. We hypothesize that the final prices may be correlated with the demand to capacity ratio. The demand to capacity ratio encapsulates both supply and demand of a particular course, so a more overdemanded course may have a higher price. The variables are plotted in Figure 4, where the x-axis is the demand to capacity ratio and the y-axis is the final price. We observe a slight positive correlation between the two. Other factors may also influence the final price of the course.

Table 2: Course prices

Class	Prices	Ratio
CSEEW3827	0.25295	1.071429
CSEEW4119	0.00000	1.220000
COMSW4170	0.48095	1.236181
COMSE6424	0.22330	1.266667
COMSW3203	0.75940	1.433333
COMSW3132	0.59970	1.466667
COMSW3261	0.47915	1.523333
COMSW4460	0.96005	1.650000
COMSE6111	0.72405	2.306667
COMSE6178	0.98005	2.366667
COMSW4732	0.75945	2.800000

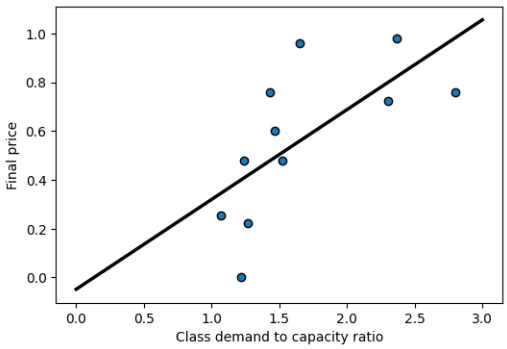

Figure 4: Relationship between class demand to capacity ratio to prices

Lastly, we can evaluate the resulting budget perturbations. Only 3 students had a unique budget from the rest of their class, which is good since that means very few students got a advantage over their

peers. The rest of the students were able to afford their bundles with their minimum budget in their range ($\beta_0 - \epsilon$).

5 Conclusion

We were able to implement the A-CEEI course allocation described in Budish et al. [2023] and ran it on an artificial dataset generated from the Columbia Computer Science Department registration data. We cound that it is possible to reach an optimal solution with zero clearing error in $\sim$20 minutes. Prices that had a higher demand to capacity ratio tended to have higher prices.

5.1 Challenges

We initially started the project off by developing a survey that was sent to undergraduate students in the Computer Science department. As discussed earlier in the paper the algorithm uses ranked demands per student. The goal of our survey was to collect data that would provide rankings for courses. Unfortunately we did not receive much participation in our survey and decided to work with the Computer Science department to access an alternative data source.

The data set provided to us by the Computer Science department reflects a snapshot at the start of the registration period for the Spring 2024 semester. Given that not all students had signed up for courses at that time we found that the numbers for the waitlists and enrollment and did not accurately reflect reality. After simulating the data we found that the capacities were not exceeded for most courses. This issue caused challenges when allocating student to all 43 courses. Either our algorithm was able to reach an efficient allocation (i.e. zero clearing error) in the first iteration, or our data was too sensitive to the delta, epsilon values thus resulting in an infeasible linear program.

Additionally, given our limited compute power we had to adjust our approach on a few aspects of the algorithm. We initially intended to implement an additional constraint for Envy-Free but for Tie-Breaking (EF-TB) into the integer program. This was not implemented as it would require comparing every student's allocation against every other student's allocation. This was not manageable given the size of our data set.

5.2 Future Work

In the future, there are other aspects in Budish et al. [2023] that we have not yet implemented, such as the EF-TB constraint and other efficiency improvements. With those improvements, we believe that the algorithm would be more efficient to run over a larger dataset. Additionally, it would be interesting to explore running the code on a different system or using a more low-level language, since many of the challenges we ran into were caused by our Python Jupyter Notebooks freezing up and crashing on our machines. It would be interesting to run the algorithm more extensively across a wider range of parameters adn inputs.

In terms of implementation in a department like Columbia's Computer Science Department, there's still a long ways to go. There are more considerations such as prerequisites and required core classes that cannot be taken into account by the current model. We believe that a useful implementation would be to run this sort of algorithm on a high-level elective classes with high demand, since those complexities wouldn't exist, and the set of students and courses would be much smaller.

We believe in order to further our analysis on course seat allocation in the Computer Science department at Columbia University, it would be important to collect data from students that includes their course rankings, similar to how ranking is currently done at Columbia Business School. With accurate data, we believe we could properly build utility functions and assess accurate metrics in terms of efficiency, fairness and potential areas for manipulation by the students.

6 Supplementary Material

Code can be found at: `https://github.com/sarah-gu/course-allocation-aceei`. Running the A-CEEI algorithm with the generated data in `demand\_set.csv` can reproduce the results described above.

References

E. Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011. ISSN 00223808, 1537534X. URL http://www.jstor.org/stable/10.1086/664613.

E. Budish, G. P. Cachon, J. B. Kessler, and A. Othman. Course match: A large-scale implementation of approximate competitive equilibrium from equal incomes for combinatorial allocation. *Oper. Res.*, 65(2):314–336, apr 2017. ISSN 0030-364X. doi: 10.1287/opre.2016.1544. URL https://doi.org/10.1287/opre.2016.1544.

E. Budish, R. Gao, A. Othman, A. Rubinstein, and Q. Zhang. Practical algorithms and experimentally validated incentives for equilibrium-based fair division (a-ceei). In *Proceedings of the 24th ACM Conference on Economics and Computation*, EC '23, page 337–368, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701047. doi: 10.1145/3580507.3597809. URL https://doi.org/10.1145/3580507.3597809.

A Allocation Results

Table 3: A-CEEI Results, Initial Ranking provides information on the first ranking of course numbers

Student	Initial Ranking	A-CEEI Allocation	Allocation Preference	Budget
0	[10, 6, 2]	[2, 6]	3	0.99000
1	[4, 10, 2]	[4]	4	0.99000
2	[4, 0, 6]	[0, 6]	3	0.99000
3	[10, 0, 2]	[0, 2]	3	0.99000
4	[10, 4, 2]	[10]	4	0.99000
5	[2, 5, 4]	[2]	4	0.99000
6	[0, 6, 5]	[0, 6]	1	0.99000
7	[10, 1, 8]	[1, 10]	1	0.99000
8	[4, 6, 5]	[4]	4	0.99000
9	[2, 10, 4]	[2]	4	0.99000
10	[2, 4, 8]	[2]	4	0.99000
11	[4, 2, 6]	[2, 6]	3	0.99000
12	[8, 4, 9]	[8]	4	0.99000
13	[9, 8, 4]	[8]	5	0.98000
14	[4, 9, 10]	[4]	4	0.98000
15	[0, 4, 6]	[0, 6]	2	0.98000
16	[6, 2, 4]	[2, 6]	1	0.98000
17	[5, 0, 4]	[0, 5]	1	0.98000
18	[0, 4, 10]	[0]	4	0.98000
19	[6, 2, 7]	[2, 6]	1	0.98000
20	[0, 4, 6]	[0, 6]	2	0.98000
21	[8, 4, 2]	[8]	4	0.98000
22	[4, 6, 3]	[3, 4]	2	0.98270
23	[0, 2, 4]	[0, 2]	1	0.98000
24	[2, 4, 0]	[0, 2]	2	0.98000
25	[4, 6, 2]	[2, 6]	3	0.98000
26	[4, 0, 5]	[0, 5]	3	0.97000
27	[1, 0, 4]	[0, 1]	1	0.97000
28	[10, 3, 4]	[10]	4	0.97000
29	[7, 6, 4]	[7]	4	0.97000
30	[2, 4, 10]	[2]	4	0.97000
31	[0, 4, 6]	[0, 6]	2	0.97000
32	[9, 10, 2]	[9]	4	0.98005
33	[2, 8, 6]	[2, 6]	2	0.97000
34	[10, 8, 6]	[10]	4	0.97000
35	[4, 0, 2]	[0, 2]	3	0.97000
36	[2, 1, 6]	[1, 2, 6]	0	0.97000
37	[4, 1, 6]	[1, 4]	1	0.97000
38	[6, 2, 4]	[2, 6]	1	0.97000
39	[7, 8, 4]	[8]	5	0.96000
40	[2, 5, 4]	[2]	4	0.96000
41	[1, 10, 4]	[1, 10]	1	0.96000
42	[4, 0, 2]	[0, 2]	3	0.96000
43	[4, 10, 5]	[4]	4	0.96000
44	[0, 7, 2]	[0, 2]	2	0.96000
45	[6, 8, 2]	[6]	4	0.96000
46	[2, 6, 4]	[2]	4	0.96000
47	[2, 6, 4]	[2]	4	0.96000
48	[7, 0, 4]	[7]	4	0.96005
49	[4, 6, 2]	[4]	4	0.96000

B Course Name to Index

Table 4: Course prices

Class	Index
CSEEW3827	0
CSEEW4119	1
COMSW4170	2
COMSE6424	3
COMSW3203	4
COMSW3132	5
COMSW3261	6
COMSW4460	7
COMSE6111	8
COMSE6178	9
COMSW4732	10