# Scaling Fair and Efficient Course Allocation: From Theory to Practice

Paula Navarrete Díaz, Cyrus Cousins, George Bissias and Yair Zick

This is an abbreviated version of our working paper [11]. Further details can be found there.

**Abstract**

Universities regularly face the challenging task of assigning classes to thousands of students while considering their preferences, along with course schedules and capacities. Ensuring the effectiveness and fairness of course allocation mechanisms is crucial to guaranteeing student satisfaction and optimizing resource utilization. We approach this problem from an economic perspective, using formal justice criteria to evaluate different algorithmic frameworks. To evaluate our frameworks, we conduct a large scale survey of university students at University of Massachusetts Amherst, collecting over 1000 student preferences. This is, to our knowledge, the largest publicly available dataset of student preferences. We develop software for generating synthetic student preferences over courses, and implement four allocation algorithms: the serial dictatorship algorithm used by University of Massachusetts Amherst; Round Robin; an Integer Linear Program; and the Yankee Swap algorithm. We propose improvements to the Yankee Swap framework to handle scenarios with item multiplicities. Through experimentation with the Fall 2024 Computer Science course schedule at University of Massachusetts Amherst, we evaluate each algorithm's performance relative to standard justice criteria, providing insights into fair course allocation in large university settings.

## 1 Introduction

Public universities regularly run large-scale matching markets: enrolling students to classes. There are over 15 million undergraduate students in the United States alone [32], with some universities boasting cohorts numbering in the tens of thousands [32, Table 312.10]. The sheer scale of the course allocation problem requires the use of automated assignment mechanisms, which typically (a) collect student preferences and (b) assign students to classes based on their eligibility/preferences/priority. Ideally, course allocation mechanisms should be fast, effective and satisfy certain objectives. One reasonable objective is *efficiency*: ensuring that classes are assigned to students who actually want and are able to take them; another is *fairness*: ensuring that class seats are fairly distributed among students. Other constraints must be accounted for as well: student schedules should be *feasible*, i.e., the mechanism does not cause a scheduling conflict; another is *student priority*: course allocation mechanisms should prioritize students with greater needs. Properly designed, course allocation mechanisms guarantee student satisfaction, facilitate timely graduation, and optimize the utilization of public resources.

The course allocation problem can be naturally modeled as a problem of *fair allocation of indivisible goods* [14]: students are *agents* who exhibit preferences over *items* — seats in classes. Our goal is to design a *mechanism* that assigns items to agents while satisfying certain design criteria. *Prima facie*, the course allocation problem is well-positioned to be central in the fair allocation community: it is a regularly occurring market involving thousands of agents with preferences over thousands of items (unlike smaller domains, e.g., those offered by the Spliddit platform [24]); moreover, it offers a rich landscape of constraints and modeling challenges: modeling student preferences, accounting for scheduling clashes, and determining acceptable justice criteria. *The lack of public datasets* is one major obstacle to the proper scientific analysis of course allocation mechanisms. Recent works, e.g., the Course Match mechanism, [15, 17, 18, 38] offer empirical analysis of course allocation instances; however, these works do not publicly release their datasets, nor are their proposed allocation mechanisms publicly available. This makes it difficult to independently verify the efficacy of their proposed approach. In this work, we take a first step towards creating a *publicly available*, *large-scale* ecosystem for the analysis of the course allocation problem.

### 1.1 Our Contribution

We implement several algorithmic frameworks for large-scale fair allocation, and test them on preference data collected from students at University of Massachusetts Amherst. The data [*] and algorithmic frameworks [†] are all in public repositories. In more detail, our contributions include:

**Data collection:** we collect preference data from 1061 students in the computer science department of University of Massachusetts Amherst. In the fair allocation domain, this dataset represents one of the largest sources of publicly available preference data.

**Software for Large-Scale Fair Allocation:** we implement four allocation mechanisms: Round Robin (a classical scheduling algorithm, also known as the draft mechanism [16]), a version of Serial Dictatorship (the algorithm used at University of Massachusetts Amherst), an Integer Linear Program that finds the solution maximizing utilitarian social welfare, and Yankee Swap [39], a recently introduced fair allocation mechanism. We discuss practical considerations in the development of the mechanisms, as well as methods to utilize structural properties of the course allocation problem which significantly improve the runtime of the Yankee Swap framework. We test each algorithm against the Fall 2024 University of Massachusetts Amherst Computer Science course schedule, and measure its performance relative to a number of standard *justice criteria*.

[*] https://github.com/Fair-and-Explainable-Decision-Making/course-allocation-data
[†] https://github.com/Fair-and-Explainable-Decision-Making/yankee-swap-allocation-framework

**Synthetic Student Preference Generation:** to test our algorithmic frameworks under different regimes of course supply and demand, we introduce a synthetic student generator which creates new students based on the collected preference data. The student preference generator creates student profiles that, while distinct from the original data, follow a similar preference distribution.

### 1.2 Related work

The course allocation problem is well-studied in combinatorial optimization, with various mechanisms proposed to balance efficiency, fairness, and strategic considerations. Hatfield [26] shows that serial dictatorships are the only Pareto efficient, strategyproof and non-bossy mechanisms; Hatfield [26] focuses on additive preferences, whereas we study more general submodular utilities. Budish and Cantillon [16] examine the Draft (Round-Robin) mechanism for course allocation, and show that the Draft mechanism retains high efficiency guarantees in practice, despite students misreporting their preferences. Budish and Cantillon [16] also collect data on a similar scale to ours (900 students at Harvard Business School), with students ranking classes on a Likert scale. The bidding point mechanism is widely used for course allocation [37]: students distribute a fixed number of points among their preferred courses. Bids are then processed in decreasing order and honored if the course and the student's schedule are not at capacity. This approach focuses solely on efficiency, neglecting equity, and is highly susceptible to strategic manipulation, as students may misreport their preferences to get better outcomes [2]. Furthermore, since bids serve as proxies for preferences, the inferred preferences may not accurately reflect true student demand, leading to market distortions [37, 28].

Sönmez and Ünver [37] propose a Gale-Shapley Pareto-dominant market mechanism that separates the dual role of bids by assigning courses through a matching mechanism considering students preferences, while using bids solely to break ties. In any case, inefficiencies arise from students overbidding on courses they could have gotten with fewer points. Atef Yekta and Day [2] introduce multi-round algorithms, based on matching and second-price concepts, which addresses these inefficiencies and show promising empirical results.

Diebold et al. [20] study course allocation as a two sided matching problem, where courses have preferences over students, induced by student priority. They analyze the Gale-Shapley Student-Optimal Stable mechanism — a modified version of the Gale-Shapley deferred acceptance algorithm [22] — and the Efficiency Adjusted Deferred Acceptance mechanism [27], which reduces the welfare losses produced by the deferred acceptance algorithm, but is not strategyproof. Similar to other approaches [33, 36], these fail to incorporate conflicts between classes. Biswas et al. [12] analyze the complexity of finding fair and efficient course allocations under course conflicts (see also [13]). They establish that the problem is computationally intractable, unless the number of agents is small. At University of Massachusetts Amherst, course schedules are very standardized, which makes course schedule conflicts well-structured and tractable.

Budish [15] introduces the Approximate Competitive Equilibrium from Equal Incomes (A-CEEI) mechanism, which is strategy-proof at large, and approximates envy-freeness and maximin share fairness, both in theory and empirically (see also [34]). In the context of course allocation, this mechanism faces some challenges: approximation errors can result in capacity constraint violations, and, early implementations faced scalability issues [17, 2].

Course Match [17], an A-CEEI-based approach, addresses these issues by introducing two additional stages to ensure feasibility, together with an interface for students to report utilities, and a software system capable of handling reasonably sized instances. Course Match relies on students reporting their complete ordinal preferences over all possible course bundles — an impractical requirement. Instead, students only rate individual courses and course pairs. At the same time, Course Match does not guarantee optimal outcomes if students misreport or fail to articulate their true preferences. To mitigate this issue, Soumalias et al. [38] propose a machine-learning-powered version of Course Match to reduce

reporting errors by iteratively asking student to rate personalized pairwise comparison queries.

Despite these efforts, Course Match is not guaranteed to find a price vector and corresponding allocation that respects course capacities, and may not run in reasonable time for large instances [17]. Furthermore, Course Match is a commercial product. To our knowledge, there is currently no publicly available implementation of Course Match, nor is there large-scale empirical data to evaluate its effectiveness in real-world settings. Lastly, Course Match places a significant elicitation burden on students. Students are encouraged to study a 12-page manual in order to understand the mechanism and enroll in courses [42]. While this may be reasonable for MBA students or those in advanced economics-related programs, it may not be suitable for undergraduates from diverse majors and academic backgrounds at a large public university.

Fair allocation algorithms and solution concepts have been extensively studied in recent years, with a particular focus on leveraging the structure of agent preferences (see survey by Aziz et al. [3]). Recent works focus on settings where agents have *binary submodular* utilities, i.e., items have a marginal utility of either 0 or 1, and agents exhibit diminishing marginal returns on items as their bundles grow [4, 6, 23, 10, 40, 39]. Following these works, we implement the Yankee Swap algorithm [40] as one of our benchmarks. The Yankee Swap algorithm outputs a *Lorenz dominating* allocation, which is guaranteed to satisfy several theoretical guarantees [4, 10]. In addition to maximizing the minimal utility of any agent (implied by the leximin property), it is approximately envy-free, maximizes utilitarian welfare, and offers each agent at least half of their maximin share guarantee. Furthermore, Yankee Swap is a strategyproof mechanism, and can encode student priorities [39]. Our empirical evaluation confirms the effectiveness of the Yankee Swap framework, providing evidence for its potential efficacy as a practical course allocation mechanism.

## 2 Course Preference Data

We conducted a survey to gather student preferences for courses offered by the Computer Science department at University of Massachusetts Amherst during the Fall 2024 semester. We invited all undergraduate and graduate students in the department to participate in the survey. We reached out in May 2024, shortly after the final examination period but before grades were released. The survey timing ensured that most students were available — they were not overly busy with classwork or studying for exams, nor were they off-campus — and had time to consider their preferences before answering our survey — they had recently indicated their course preferences for the Fall 2024 on the University of Massachusetts Amherst portal. The survey achieved a significant effective response rate of 30.33%, with a relatively balanced response rate across students of different academic status (see Table 1).

The Fall 2024 course schedule for the Computer Science department includes 96 distinct course sections. Each section is identified by a combination of attributes such as catalog number, section number, course name, instructor, capacity, and time of day. The same catalog number may offer multiple sections. However, the combination of catalog number and section number is unique, and is referred to as a *course* from now on.

Participation was voluntary, and participants received a $10 Amazon gift card. The survey was hosted on Qualtrics, an online survey platform, and consisted mainly of the following questions:

**Current Status:** current academic status refers to the student's current year in their program, whether they are an undergraduate (freshman, sophomore, junior, or senior) or a graduate student (enrolled in a master's or Ph.D. program). We used this information to tailor the courses presented in surveys to reduce cognitive load —we did not collect student preferences on any courses that were inappropriate for their academic status— and to determine student's priority when enrolling in classes: PhD students are given priority over MS students, who are given priority over seniors, and so on.

| Academic Status | Number of Students | Number of Responses | Response Rate (%) | Effective responses | Effective Response Rate (%) |
|---|---|---|---|---|---|
| Freshmen | 239 | 156 | 65.27 | 125 | 53.30 |
| Sophomore | 327 | 134 | 40.98 | 113 | 34.56 |
| Junior | 408 | 143 | 35.05 | 126 | 30.88 |
| Senior | 573 | 135 | 23.56 | 117 | 20.42 |
| MS | 613 | 190 | 31.00 | 172 | 28.06 |
| PhD | 148 | 51 | 34.46 | 47 | 31.76 |
| Unspecified | N/A | 256 | N/A | 0 | N/A |
| Total | 2308 | 1065 | 46.14 | 700 | 30.33 |

**Table 1:** Number of students, number of responses, and response rates, both overall and broken down by academic status. Effective response rates reflect responses with non-empty preferences and specified academic status.

**Desired course load:** we asked students how many courses they wish to enroll in before and after the add/drop period. Since students are often uncertain about their final preferences, many over-enroll to explore their options before finalizing their schedules. Thus, these quantities might differ.

**Preference Over Courses:** students were shown a tailored list of courses with the following key details — course name, catalog number, section, instructor, day and time — in order to help students make informed decisions about their preferences. For each course listed, students rated their interest on a scale from 1 (not interested) to 7 (very interested), with an additional option to select 8 (required) if the course was a requirement for their program. All courses were rated 1 by default.

The total number of survey responses, the actual number of students (provided by the department), and the corresponding response rates — both overall and broken down by academic status — are summarized in Table 1. 256 respondents did not specify their academic status, and as a result, we did not use their recorded rankings in our analysis. 109 respondents indicated their status but submitted empty preferences (those students were not paid). We utilized 700 effective responses with both academic status and non-empty preferences, achieving an effective response rate of 30.33%. Table 1 provides a detailed breakdown of these values by academic status.

We did not allow students to express preferences over bundles of courses. This is a departure from other preference elicitation models in the course allocation domain, e.g., [17, 38]. We elicited information on individual courses since it allows students to express preferences over a large number of courses while avoiding significant cognitive overload. In addition, our elicitation protocol maps well to constraint based utility models: we can naturally map the results of our survey to *submodular* (or nearly submodular) student preference functions. Having a complete picture of student preferences over individual courses also allows us to naturally simulate additional student profiles (as described in Appendix A).

## 3 Course Assignment as a Fair Allocation Problem

We start by presenting the fair allocation of indivisible goods framework. Our model offers a slight departure from the standard problem formulation [14], since we allow item *types*. Let $\mathbb{N}_0$ be the set of non-negative integers. Let $N = \{1, 2, ..., n\}$ be a set of *agents* and $G = \{g_1, g_2, ..., g_m\}$ be a set of *item types*. Each item type $g \in G$ has a limited number of identical copies $q(g)$; $\vec{q} = (q(g_1), ..., q(g_m))$ is the vector describing the quantities of each item. In the context of course allocation, agents are students, item types are courses, and $q(g)$ is the number of available seats in course $g$.

The classic fair allocation literature assigns agents *bundles* of items, i.e., subsets of the set $G$. Since our framework allows items to have multiple copies, a bundle $S$ is a vector in $\mathbb{N}_0^m$, where $S_g$ is the number of copies of item $g$ that is in the bundle $S$. One can simply treat multiple copies of an item as distinct items and do away with the duplicate items model. However, the notion of item copies is useful for proving better upper bounds on the runtime of our proposed allocation mechanisms. From a modeling perspective, treating individual course seats as distinct items makes running even somewhat simple allocation mechanisms like Round Robin impractical: instead of having $\sim 100$ item types, we need to store information about $\sim 10k$ individual items. An *allocation* $X = (X_0, X_1, \ldots, X_n)$ is a partition of item copies; each agent $i \in N$ is assigned a bundle $X_i$, and $X_0$ denotes the set of unassigned copies. The allocation $X$ is represented as a matrix, where $X_i$ is a vector in $\mathbb{N}_0^m$ that represents agent $i$'s bundle under allocation $X$. Since agents may own multiple copies of an item, $X_{i,g}$ is the number of copies of item $g$ in agent $i$'s bundle; however, in the course allocation domain students do not receive more than one seat in any class. An allocation is *valid* if for any item $g \in G$, $X_{0,g} + \sum_{i=1}^{n} X_{i,g} = q(g)$: the number of unassigned copies of $g$, $X_{0,g}$, plus the number of copies assigned to all agents equals exactly the number of copies $q(g)$. For ease of readability, for an allocation $X$ and an item type $g$, we say that $g \in X_i$ if $X_{i,g} > 0$. Let $\mathbb{1}_g \in \{0,1\}^m$ be the indicator vector of item $g$, i.e., it is equal to 1 in the $g$-th coordinate and is 0 elsewhere. Given a bundle of items $S$, we write $S + g$ and $S - g$ to refer to $S + \mathbb{1}_g$ and $S - \mathbb{1}_g$, i.e., $S$ with an additional copy of the item $g$, or with one copy of $g$ removed.

Each agent $i \in N$ has a *valuation function* $v_i : \mathbb{N}_0^m \to \mathbb{N}_0$ which depends only on the bundle $X_i$ allocated to $i$. We define the *marginal utility* of agent $i$ from receiving an additional copy of the item type $g$ as

$$\Delta_i(X_i, g) = v_i(X_i + g) - v_i(X_i).$$

We say that $v_i$ is a *binary* valuation if for any bundle $S \in \mathbb{N}_0^m$ and any item type $g$, $\Delta_i(S, g) \in \{0, 1\}$.

Given two bundles $S, T \in \mathbb{N}_0^m$, we write $S \preceq T$ if for all $g \in G$, $S_g \leq T_g$; this is equivalent to stating that the bundle $S$ is a subset of $T$ in the standard fair allocation model; we write $S \prec T$ if any of these inequalities is strict. We say that $v_i$ is a *submodular* function if for any two bundles $S, T \in \mathbb{N}_0^m$ such that $S \preceq T$, and any item type $g$, $\Delta_i(S, g) \geq \Delta_i(T, g)$. Intuitively, the more items an agent $i$ owns, the less marginal benefit they receive from additional items.

Finally, a bundle $S \in \mathbb{N}_0^m$ is *clean* [10] with respect to agent $i \in N$ if for any bundle $T \prec S$ we have $v_i(S) > v_i(T)$, i.e., removing any item from $i$'s bundle strictly reduces their utility.

### 3.1 Encoding Student Valuations

While our framework applies to general valuation settings, in our empirical evaluations we model students as having binary preferences. Binary preferences are extensively studied in the fair allocation literature, see e.g. [4, 7, 6, 10, 9, 25, 40, 39]. Thus, we have a good understanding of their properties and the types of guarantees we can offer; furthermore, there exist algorithmic frameworks that compute fair and efficient allocations for agents with binary submodular valuations. We chose to elicit numerical preferences from our survey participants in order to offer a more refined view of student preferences, and to support future empirical analysis of algorithmic frameworks beyond those that handle binary preferences. In other words, we see the *data collection* and the *algorithmic frameworks/preference encoding* portions of our work as complementary yet distinct contributions to the empirical evaluation of fair allocation mechanisms.

Allocation mechanisms in the literature typically rely on oracle access to agent valuations. However, pre-computing and storing agent valuations in a fixed data table is intractable when dealing with thousands of students and hundreds of classes; what's more, the allocation mechanisms we utilize do not need to know agents' valuations for all possible bundles. To address this challenge, we encode student preferences via *linear inequality constraints*. We can encode several types of natural course

constraints via linear inequalities, such as course scheduling conflicts, maximum enrollment caps for a student, substitutions (wanting to take course A or course B but not both), and taking only classes the students approve.

We represent the preferences of a student $i$ by $r_i$ linear constraints through a $r_i \times m$ matrix $Z^i$ and a limit vector $\vec{b}^i$. A bundle $X_i$ is *feasible*, i.e., it is clean with respect to agent $i$, if and only if

$$Z^i X_i \leq \vec{b}^i. \tag{1}$$

These constraints can be computed as needed at a relatively low cost, offering a more manageable approach to modeling student preferences.

Finally, the utility an agent obtains from a bundle $X_i$ is the size of the largest feasible sub-bundle of $X_i$. In other words, let $|S|$ denotes the 1-norm of $S \in \mathbb{N}_0^m$, then

$$v_i(X_i) = \max\{|S| : Z^i S \leq \vec{b}^i, S \preceq X_i\}, \tag{2}$$

It is relatively straightforward to show that student preferences encoded by Equation (2) are binary: adding a single item copy can increase the cardinality of the feasible set by at most one. While our encoding ensures that student valuations are binary, they are not necessarily submodular. For example, assume that a student $i$ likes classes $A$, $B$ and $C$. If class $A$ conflicts with $B$, $B$ conflicts with $C$, and $A$ does not conflict with $C$, then the marginal gain of giving $C$ to $i$ when they have $B$ is 0; however, if they have both $A$ and $B$, then the marginal gain of $C$ is 1. In this scenario, items do not exhibit *substitutability* [35], a key property of submodular preferences. However, in our data, student scheduling constraints largely retain submodularity. This is because the vast majority of classes are scheduled at a set of predetermined times, e.g., Tues/Thu at 1pm-2:15pm. Thus, if class $A$ conflicts with $B$ and $B$ conflicts with $C$, then $A$ conflicts with $C$, avoiding issues similar to those described above.

### 3.2 Allocation Mechanisms

An allocation algorithm takes as input a set $G$ of $m$ courses with $\vec{q} = (q(g_1), ..., q(g_m))$ seats per course, and a set $N$ of $n$ students with valuations $(v_1, \ldots, v_n)$. Its output is some feasible allocation $X$ of the course seats. We consider four different allocation algorithms: Serial Dictatorship, Round Robin, an Integer Linear Program, and Yankee Swap.

**Serial Dictatorship:** the course allocation algorithm used by University of Massachusetts Amherst for student enrollment uses a variant of Serial Dictatorship (SD) [26]. The allocation system opens to students in order of seniority. PhD students enter first, followed by MS students; next, the system opens for undergraduate enrollment in decreasing order of seniority. We emulate this process by letting students pick in order of their academic status, and in random order within members of the same academic cohort. The first student to access the platform enrolls in all their desired courses. Subsequently, the following students enroll in all desired courses that have available seats left. Students can enroll in classes they do not want to take; however, since SD is strategyproof, students have no incentive to do so. However, students are often uncertain about what classes they actually want to take, and tend to enroll in more classes than they intend on keeping (subsequently dropping out during the grace period early in the semester). Students who access the platform early get a wide range of available classes and 'hoard' classes; those who log in later face a considerable disadvantage in securing desired classes. These issues with SD are known from prior works on course allocation (e.g. [15, 16]).

**Round Robin:** the *Round Robin algorithm* (RR) [14] (also known as the draft mechanism [16]), operates in rounds. In each round, students are assigned one available seat that they like, i.e., one that offers them a marginal gain of 1. This continues until no such seats are available. The order in which students select their seats is fixed. We again prioritize students according to academic status.

**Integer Linear Program:** allocation by Integer Linear Program (ILP) finds an allocation $X$ that maximizes the utilitarian social welfare subject to course capacity constraints, and linear inequality constraints that encode student preferences (see Section 3.1). Since both the objective and constraints are linear, the ILP returns an optimal integer allocation with respect to USW.

**Yankee Swap:** in the *Yankee Swap algorithm* (YS) [40, 39] agents sequentially pick items. In each round, YS selects the lowest utility student, breaking ties in favor of higher academic status, and lets them pick a seat that offers them a marginal benefit of 1. If no such seat is available, they may steal such a seat from another student, as long as that student can recover their utility by either taking an unassigned seat or stealing a seat from yet another student. This goes on until the last student takes an unassigned seat. If no such *transfer path* exists, the student is not elected again. The algorithm terminates when no student can further benefit from enrolling in courses with available seats, or when students are solely interested in stealing seats from students who cannot recover their utility. When agents have binary submodular valuations, YS offers several theoretical guarantees: it outputs a leximin, EF-X allocation, which also maximizes USW. In addition, YS is *truthful*: no agent can increase their utility by misreporting their preferences, e.g., falsely state that they want to enroll in an undesirable class, or say that they do not want to enroll in a desirable class.

## 4 Adapting the Yankee Swap Mechanism in the Course Allocation Domain

The main computational overhead of the Yankee Swap algorithm [40] stems from maintaining the *exchange graph* $\mathcal{G}$. Given an allocation $X$, $\mathcal{G}(X)$ is a directed graph over the set of items $G$. There is an edge from an item $g \in X_i$ to another item $g' \in G$ if $v_i(X_i) = v_i(X_i - \mathbb{1}_g + \mathbb{1}_{g'})$. In other words, there is an edge from $g$ to $g'$ if the agent who owns $g$ under $X$ can replace it with $g'$ without reducing their utility. If no agent owns item $g \in G$ (i.e., $g \in X_0$), then $g$ has no outgoing edges.

The Yankee Swap algorithm works as follows. Initially, all agents are assigned empty bundles and all items are unassigned, i.e., the initial allocation $X$ is such that $X_{0,g} = 1$ and $X_{i,g} = 0$ for all item types $g \in G$ and agents $i \in N$. At each round, we pick an agent $i$ with the lowest utility, breaking ties arbitrarily. Let $F_i(X) = \{g \in G : \Delta_i(X_i, g) = 1\}$ be the set of items that offer agent $i$ a marginal benefit of 1 given their current bundle. We search for a shortest path from $F_i(X)$ to items in $X_0$ in the resulting graph and execute it. Note the execution of the path $(g_0, g_1, \ldots, g_p)$ corresponds to agent $i$ taking item $g_0$, and the agent who owns item $g_0$ replaces it with item $g_1$, and so on, until we reach the unassigned item $g_p$, which is now assigned to the last agent in the path. This is referred to as *path augmentation*. If no such path exists, agent $i$ is kicked out of the game. We repeat this process until there are no agents left playing or all items have been allocated.

The original problem formulation assumes no item multiplicity. When items have multiple identical copies, considering each item copy individually becomes highly inefficient, as the problem size scales by the number of course *seats* rather than the number of courses, which is typically significantly smaller.

We propose a modified version of the Yankee Swap algorithm that allows items to have multiple identical copies, i.e., $q(g) > 1$. Items now might have multiple owners, which requires a redefinition of the exchange graph. The exchange graph $\mathcal{G}$ is now a directed graph over the set of item types $G$, in which there is an edge of the form $(g, g')$ if there *exists* some agent $i$ who owns a copy of $g$, and is willing to exchange it for a copy of $g'$. Thus, for each edge $(g, g')$ we need to maintain a list of agents willing to make the exchange.

We define a *responsible agents tuple* $R$ consisting of $m^2$ lists of agents that keep track of the agents responsible for each of the edges. Here, $R_{g \to g'}$ is the set of agents who own a copy of $g \in G$ and are willing to exchange it for a copy of $g' \in G$:

$$R_{g \to g'} = \{i \in N : X_{i,g} > 0 \wedge v_i(X_i) = v_i(X_i - \mathbb{1}_g + \mathbb{1}_{g'})\}.$$

$R_{g\to g'} = \emptyset$ if and only if there is no edge $(g, g')$ in $\mathcal{G}$: no agent who owns a copy of $g$ wants to exchange it for a copy of $g'$. Initially, $R_{g\to g'} = \emptyset$ for every $g, g' \in G$ since no items have been allocated. Once a path transfer is executed, every agent who had their bundle changed in the process might change the $R_{g\to g'}$ values. If done naively, these checks can be time-consuming: theoretically one needs to check whether any item in agent $i$'s bundle can be exchanged for any other item. To avoid this overhead, we observe that the set of classes that agent $i$ can potentially want to exchange their classes for is a subset of the set of classes they approve. Let $D_i = \{g \in G | v_i(\mathbb{1}_g) = 1\}$ be the set of agent $i$'s approved classes. If agents have binary submodular valuations over the items, then for any allocation $X$, $F_i(X) \subseteq D_i$. In particular, $F_i(X) = D_i$ when $X$ is the initial empty allocation.

Our implementation of YS (see Algorithm 1) incorporates key adjustments to account for item multiplicity and efficiency. Initially all items copies are unassigned, i.e., $X_{0,g} = q(g)$ for all $g \in G$. Second, and more importantly, we run path augmentations over item copies while keeping track of item owners. Third, instead of reconstructing the exchange graph after each iteration, we incrementally update it, resulting in a more efficient execution.

**Algorithm 1** Yankee Swap with Item Multiplicity

**Require:** A set of item types $G$ with $\vec{q} = (q(g_1), ..., q(g_m))$ copies each, and a set of agents $N$ with binary submodular valuations $\{v_i\}_{i\in N}$ over the items
**Ensure:** A clean leximin allocation

$X = (X_0, X_1, ..., X_n) \leftarrow (\vec{q}, \vec{0}, ..., \vec{0})$, $\mathbb{R}_{g\to g'} \leftarrow \emptyset$ for all $g \in G, g' \in G$, $U \leftarrow N$
Build graph $\mathcal{G}$ with one node per item type $g \in G$
**while** $U \neq \emptyset$ **do**
    Let $i \in \arg\min_{j\in U} v_j(X_j)$
    Check if there is a path $P = (g_0, ..., g_p)$ where $g_0 \in F_i(X)$ and $g_p \in X_0$
    **if** a path exists **then**
        $I \leftarrow \{i\}$, $X_i \leftarrow X_i + \mathbb{1}_{g_0}$
        **for** each pair $(g, g')$ in $P$ **do** `//Path Augmentation Phase 1`
            Let $j \in R_{g\to g'}$
            $I \leftarrow I \cup \{j\}$, $X_j \leftarrow X_j - \mathbb{1}_g + \mathbb{1}_{g'}$
            **for** each item type $h \in D_j$ **do**
                **if** $j \in R_{g\to h}$ and $X_{j,g} = 0$ **then**
                    $R_{g\to h} \leftarrow R_{g\to h} \setminus \{j\}$
                    **if** $R_{g\to h} = \emptyset$ **then**
                        Remove edge $(g, h)$ from $\mathcal{G}$
        $X_0 \leftarrow X_0 - \mathbb{1}_{g_p}$
        **for** each agent $j \in I$ **do** `//Path Augmentation Phase 2`
            **for** each item type $h \in X_j$ and item type $h' \in D_j$ **do**
                **if** $j \in R_{h\to h'}$ **then**
                    **if** $v_j(X_j) > v_j(X_j - \mathbb{1}_h + \mathbb{1}_{h'})$ **then**
                        $R_{h\to h'} \leftarrow R_{h\to h'} \setminus \{j\}$
                        **if** $R_{h\to h'} = \emptyset$ **then**
                            Remove edge $(h, h')$ from $\mathcal{G}$
                **else**
                    **if** $v_i(X_i) \leq v_i(X_i - \mathbb{1}_h + \mathbb{1}_{h'})$ **then**
                        $R_{h\to h'} \leftarrow R_{h\to h'} \cup \{j\}$
                        **if** $|R_{h\to h'}| = 1$ **then**
                            Add edge $(h, h')$ to $\mathcal{G}$
    **else**
        $U \leftarrow U \setminus \{i\}$
**return** $X$

Path augmentation occurs in two distinct phases. In the first phase, we identify the agents involved in the transfer path and update the allocation matrix $X$. For instance, if agent $j$ exchanges item $g$ for $g'$, we update the matrix to reflect that the agent now possesses $g'$ and no longer has $g$. Next, we update the exchange graph $\mathcal{G}(X)$ and the tuple $R$ to account for items that agents along the path *lose*. Specifically, if agent $j$ loses the item $g$ then for any item $h$ approved by agent $j$, $h \in D_j$, we remove agent $j$ from $R_{g \to h}$. If $R_{g \to h}$ becomes empty, we remove the edge $(g, h)$ from $\mathcal{G}$. In the second phase, we update the exchange graph and the matrix $R$ based on items that agents along the path *receive*. For each agent $j$ identified in the previous phase, we check their updated bundle and determine whether they are willing to exchange any item in their bundle $X_j$ for items in their desired set $D_j$. We then adjust the tuple $R$ and update the edges in the exchange graph accordingly. Since $D_j$ is the set of classes that each student intrinsically approves, it remains invariant throughout the run of Yankee Swap, and is relatively easy to maintain.

A naive implementation of Yankee Swap searches for a shortest path in the exchange graph in time quadratic in $q_{\text{total}} = \sum_{g \in G} q(g)$, which is significantly larger than the number of item types $m$. By redefining the exchange graph and adapting the algorithm accordingly, we reduce the runtime dependency of the shortest path to be quadratic on $m$ instead. Second, it turns out that calls to student valuations can be expensive; separating the exchange graph representation from the agents through the tuple $R$ considerably reduces the number of student valuation function calls (see Appendix B).

## 5 Justice Criteria

We evaluate the performance of course allocation mechanisms according to *fairness* and *social welfare* criteria. We consider three efficiency criteria. The *Utilitarian Social Welfare* (USW) sums the total welfare of agents: $\texttt{USW}(X) = \frac{1}{n} \sum_{i \in N} v_i(X_i)$, which is equivalent to the percentage of seats assigned. The USW criterion does not account for potential welfare disparities. For example, from the USW perspective, a course assignment that offers six courses to one student and none to another is equivalent to one that assigns three courses each.

The *Nash welfare* [31, Chapter 3] strikes a balance between utilitarian and egalitarian approaches. An allocation that maximizes the Nash welfare first minimizes the number of agents with zero utility. Subject to that, it maximizes the *product* of agent utilities, i.e. the value $\texttt{NSW}(X) = \sqrt[n]{\prod_{i \in N_{>0}(X)} v_i(X_i)}$, where $N_{>0}(X)$ is the set of agents with positive utility under $X$. Since the Nash welfare takes the geometric average of the positive student utilities, we complete the picture by reporting the number of students who receive an empty bundle, i.e., students who ended up receiving no classes that they like.

We also evaluate allocations based on *fairness* metrics. Given an allocation $X$, we say that agent $i$ *envies* agent $j$ [21] if $v_i(X_i) < v_i(X_j)$. An allocation is *envy-free* (EF) if no agent envies another. We count the number of times an agent envies another, i.e.,

$$\texttt{Envy}(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{1}[v_i(X_i) < v_i(X_j)].$$

Envy-free allocations are not guaranteed to exist (consider a setting with two agents and one single item); this gave rise to a the canonical notion of *envy-freeness up to one item* (EF-1) [15, 30]. An allocation $X$ is EF-1 if for any two agents $i$ and $j$, if $i$ envies $j$, then there exists some item $g \in X_j$ such that $v_i(X_i) \geq v_i(X_j - \mathbb{1}_g)$: removing some item from agent $j$'s bundle eliminates agent $i$'s envy. We count the number of times an EF-1 violation occurs:

$$\texttt{Envy-1}(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{1}[\forall g \in X_j : v_i(X_i) < v_i(X_j - \mathbb{1}_g)].$$

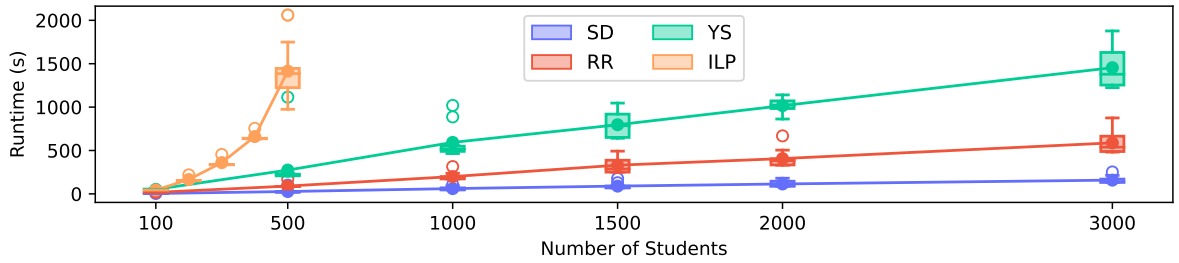



**Figure 1:** Runtime of the four allocation algorithms as a function of the number of students. Boxplots represent the distribution of runtime values across 10 runs with different random seeds for each instance.

Lastly, we wish to examine *share-based* fairness concepts, e.g., the maximin share [15]. The maximin share guarantee of agent $i$ is computed by letting agent $i$ partition the items into $n$ bundles, and receiving the worst one. More formally, $\texttt{MMS}_i = \max_X \min_j v_i(X_j)$. While the maximin share is a well-established and well-studied share-based concept (see, e.g.,[1, 8, 5, 15, 19, 23, 29]), it is inappropriate in our setting. The reason is simple: most students rate the vast majority of classes at 1, i.e., they do not want to take them. Thus, any partition of item copies to $n$ bundles will result in at least one bundle whose value is 0. This implies that the maximin share of the vast majority of students is 0. Since we are interested in analyzing some share-based fairness criterion, we use the *pairwise maximin share* [19]. An allocation $X$ is *Pairwise Maximin Share Fair* (PMMS-fair) if for any pair of agents $i$ and $j$, agent $i$'s valuation of their own bundle is at least the valuation of the worst bundle they could get in a two way division of $X_i \cup X_j$. Finally, in our results, we count the number of times a PMMS violation occurs:

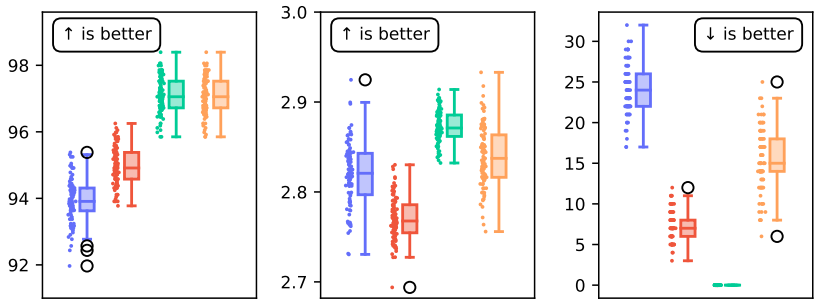
$$\texttt{PMMS}(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{1}[v_i(X_i) < \max_{T \preceq X_i + X_j} \min\{v_i(T), v_i(X_i + X_j - T)]$$

## 6 Implementation, Experiments and Results

We implement a general suite of tools for the fair allocation of indivisible items. We allow for item multiplicities and allow general constraint-based agent valuations. Our implementation includes algorithms for the mechanisms discussed in Section 3.2, and the justice criteria described in Section 5 to evaluate the allocations. In the context of course allocation, we model items as courses derived from the Fall 2024 course offerings of the Computer Science department at University of Massachusetts Amherst referred to as the *schedule*— 96 courses, with real section details, enrollment capacities, and meeting times— and agents as students with binary valuation functions.

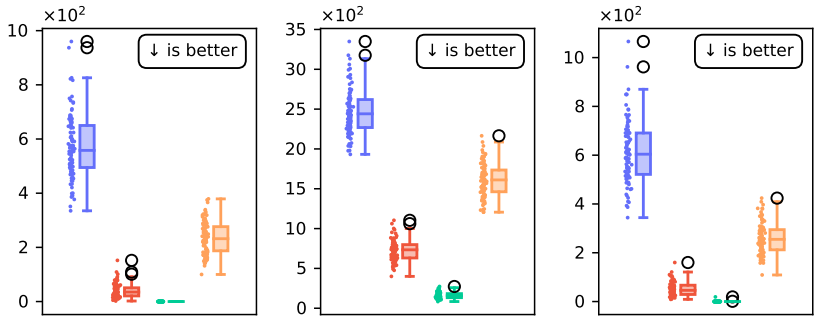
We model *real students* based on survey responses and generate *synthetic students* using the method outlined in Appendix A. From the survey responses regarding the maximum number of courses students wish to enroll in, along with corresponding simulated values, we define the maximum enrollment capacity for each student. We take the minimum between this value and the capacity allowed by the department, which is 6 for undergraduate students and 4 for graduate students.

Both real and synthetic students express preferences for each course as numerical values ranging from 1 to 8. However, since the allocation algorithms require binary preferences, we map these cardinal preferences to approval preferences by constructing a set of preferred courses using a top-$k$ approach: for each student, we determine a preference threshold based on their $k$-th preference and the corresponding set of preferred courses. This approach eliminates any inherent bias that students might exhibit in reporting their scores: being more enthusiastic about classes does not offer an advantage. In addition, it avoids breaking ties in an arbitrary fashion: all classes that receive the same rating receive the same approval score. For our experiments, we consider $k = 10$.




**(a)** % of seats assigned **(b)** Nash welfare **(c)** Num. of 0 util. agents

**Figure 2:** Welfare metrics for the reduced student instance. Figure 2a compares % of allocated seats (USW), Figure 2b compares the Nash welfare, and Figure 2c presents the number of empty bundles. We compare SD (in blue), RR (in red), YS (in green) and ILP (in orange).




**(a)** PMMS violations **(b)** Envy violations **(c)** EF-1 violations

**Figure 3:** Fairness metrics for the reduced student instance. Figure 3a shows the number of PMMS violations, Figure 3b shows the number of envious agents, and Figure 3c shows the number of EF-1 violations. We compare SD (in blue), RR (in red), YS (in green) and ILP (in orange).

We first examine the scalability of each of our benchmarks. For each cohort size, we use either real students responses, or a mixture of real and synthetic students to reach the target cohort size. We adjust the student composition to match the proportional distribution of each academic status as shown in Table 1, and we scale course capacities accordingly. All sequential algorithms use the same student tie-breaking scheme: students with a higher academic status are given higher priority, and are ordered uniformly at random within each group. Figure 1 shows that sequential approaches such as SD, YS and RR scale well on the number of students, whereas the ILP suffers from exponential blowup in its runtime, making it infeasible to evaluate for instances of more than 500 students.

To compare all four benchmarks, we use a reduced instance only of students generated from a subset of real survey responses. According to Table 1, seniors have the lowest response rate of 20.42%. We scale each course capacity to this value, and sample a subset of real responses from each other status to match this rate, obtaining an instance of 471 students. We subsample 100 different instances by varying which students are included in the sample and or the random seed given for the student generation.

Figure 2 summarizes the relative welfare performance of the four algorithmic benchmarks for reduced student instances. Since YS and ILP maximize welfare, they assign the maximal number of seats, whereas RR and SD fail to maximize welfare. YS is theoretically guaranteed to outperform all other benchmarks on Nash welfare (Figure 2b); the fact that other benchmarks sometimes outperform it is explained by Figure 2c: YS first minimizes the number of zero utility agents (all students were assigned at least one desirable class in every run of Yankee Swap), and then maximizes the Nash welfare. Every other benchmark leaves at least some students with zero utility; SD exhibits the worst performance, leaving 23 students (roughly 5% of the cohort) with zero utility on average.

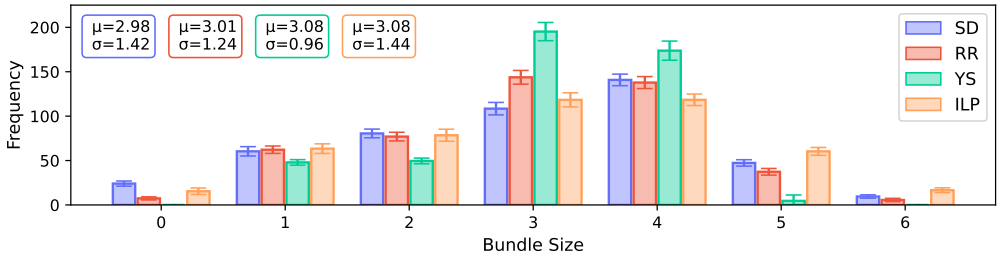



**Figure 4:** Histogram of the number of seats assigned to students under the four algorithmic benchmarks, for reduced student instances. We also report the mean $\mu$ number of seats assigned, as well as the standard deviation $\sigma$ of the assignment distribution for each allocation mechanism.

Figure 3 summarizes how the four benchmarks perform in terms of fairness. Since YS is guaranteed to be EF-1 and PMMS-fair, it significantly outperforms the other methods in terms of both PMMS and EF-1 violations. In addition, it exhibits significantly fewer envy violations than the other methods. Perhaps unsurprisingly, the allocations output by the SD mechanism highly favor students who access the system early, resulting in significant envy and PMMS violations. Finally, we note that the distribution of student utilities (Figure 4) shows that the allocations output by YS ensure significantly more equitable distributions: most students receive bundles of 3 or 4 courses, while none receive 0 or 6 classes.

We run a real-scale experiment with the three scalable algorithms — SD, RR, and YS — by augmenting the real instance with synthetically generated students until reaching a full-sized instance of 2,308 students, obtaining similar results (see Appendix C.1).

## 7 Discussion

In this work, we present a publicly available, large-scale framework including detailed student preference data, fair allocation mechanisms, and evaluation metrics. Our evaluation suite is not complete. In particular, we hope to include additional mechanisms in our suite (Course Match is a prime candidate) as well as additional survey data. We also plan to conduct longitudinal studies on the perceived effectiveness of different course allocation mechanisms within University of Massachusetts Amherst. Discussions with decision makers at University of Massachusetts Amherst identify an interesting advantage of serial dictatorship: students do not need to report their full preferences to the university, and have complete certainty regarding their course schedule when they conclude their interaction with the allocation mechanism. Assessing the impact of the inherent uncertainty of sequential allocation on student satisfaction is an important direction for future work.

Assessing the effect of course *supply and demand* is an promising direction for future work. Preliminary analysis (see Appendix C.2) indicates that artificially increasing course demand by introducing additional students results in poorer performance on fairness metrics for all mechanisms; however, since Yankee Swap offers provable fairness guarantees, it results in more equitable outcomes than other benchmarks. For example, as we increase the number of students (while keeping course capacities constant) the number of students who receive no desired classes grows linearly under serial dictatorship. Since Yankee Swap outputs leximin allocations, this effect does not occur.

Finally, we would like to encode more complex elicitation mechanisms and their effects on student satisfaction. In particular, explicitly eliciting students' preferences over course bundles (as is done by Budish et al. [17]), and integrating those preferences in our frameworks is an important direction for future work. The sequential frameworks we utilize need to be adapted in order to account for these changes. The fact that the underlying preferences explicitly encode synergies may call for more complex sequential approaches, or their integration with optimization based methods.

## References


[1] Georgios Amanatidis, Evangelos Markakis, Afshin Nikzad, and Amin Saberi. Approximation algorithms for computing maximin share allocations. *ACM Transactions on Algorithms (TALG)*, 13 (4):1–28, 2017.

[2] Hoda Atef Yekta and Robert Day. Optimization-based mechanisms for the course allocation problem. *INFORMS Journal on Computing*, 32(3):641–660, 2020.

[3] Haris Aziz, Bo Li, Hervé Moulin, and Xiaowei Wu. Algorithmic fair allocation of indivisible items: A survey and new questions. *ACM SIGEcom Exchanges*, 20(1):24–40, July 2022.

[4] Moshe Babaioff, Tomer Ezra, and Uriel Feige. Fair and truthful mechanisms for dichotomous valuations. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 5119–5126, 2021.

[5] Siddharth Barman and Sanath Kumar Krishnamurthy. Approximation algorithms for maximin fair division. *ACM Transactions on Economics and Computation (TEAC)*, 8(1), March 2020.

[6] Siddharth Barman and Paritosh Verma. Existence and computation of maximin fair allocations under matroid-rank valuations. In *Proceedings of the 20th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 169–177, 2021.

[7] Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Greedy algorithms for maximizing nash social welfare. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, page 7–13, 2018.

[8] Siddharth Barman, Ganesh Ghalme, Shweta Jain, Pooja Kulkarni, and Shivika Narang. Fair division of indivisible goods among strategic agents. In *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1811–1813, 2019.

[9] Nawal Benabbou, Mithun Chakraborty, Edith Elkind, and Yair Zick. Fairness towards groups of agents in the allocation of indivisible items. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 95–101, 2019.

[10] Nawal Benabbou, Mithun Chakraborty, Ayumi Igarashi, and Yair Zick. Finding fair and efficient allocations for matroid rank valuations. *ACM Transactions on Economics and Computation*, 9(4), oct 2021.

[11] George Bissias, Cyrus Cousins, Paula Navarrete Diaz, and Yair Zick. Deploying fair and efficient course allocation mechanisms, 2025. URL https://arxiv.org/abs/2502.10592.

[12] Arpita Biswas, Yiduo Ke, Samir Khuller, and Quanquan C Liu. An algorithmic approach to address course enrollment challenges. *arXiv preprint arXiv:2304.07982*, 2023.

[13] Arpita Biswas, Yiduo Ke, Samir Khuller, and Quanquan C Liu. Fair allocation of conflicting courses under additive utilities. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 2162–2164, 2024.

[14] Sylvain Bouveret, Yann Chevaleyre, and Nicolas Maudet. Fair allocation of indivisible goods. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia, editors, *Handbook of computational social choice*, chapter 12, pages 284–309. Cambridge University Press, 2016.

[15] Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119:1061–1061, 12 2011.

[16] Eric Budish and Estelle Cantillon. The multi-unit assignment problem: Theory and evidence from course allocation at Harvard. *American Economic Review*, 102(5):2237–2271, 2012.

[17] Eric Budish, Gérard P. Cachon, Judd B. Kessler, and Abraham Othman. Course match: A large-scale implementation of approximate competitive equilibrium from equal incomes for combinatorial allocation. *Operations Research*, 65(2):314–336, 2017.

[18] Eric Budish, Ruiquan Gao, Abraham Othman, Aviad Rubinstein, and Qianfan 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)*, pages 337–368, 2023.

[19] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Transactions on Economics and*

*Computation (TEAC)*, 7(3):1–32, 2019.

[20] Franz Diebold, Haris Aziz, Martin Bichler, Florian Matthes, and Alexander Schneider. Course allocation via stable matching. *Business & Information Systems Engineering*, 6:97–110, 2014.

[21] Duncan K. Foley. Resource allocation and the public sector. *Yale Economics Essays*, 7:45–98, 1967.

[22] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[23] Jugal Garg and Setareh Taki. An improved approximation algorithm for maximin shares. In *Proceedings of the 21st ACM Conference on Economics and Computation (EC)*, pages 379–380, 2020.

[24] Jonathan Goldman and Ariel D. Procaccia. Spliddit: Unleashing fair division algorithms. *SIGecom Exchanges*, 13(2):41––46, January 2015.

[25] Daniel Halpern, Ariel D. Procaccia, Alexandros Psomas, and Nisarg Shah. Fair division with binary valuations: One rule to rule them all. In *Proceedings of the 16th International Conference on Web and Internet Economics (WINE)*, pages 370–383, 2020.

[26] John W. Hatfield. Strategy-proof, efficient, and nonbossy quota allocations. *Social Choice and Welfare*, 33:505–515, 2009.

[27] Onur Kesten. School Choice with Consent*. *The Quarterly Journal of Economics*, 125(3):1297–1348, 2010.

[28] Aradhna Krishna and M Utku Ünver. Research note—improving the efficiency of course bidding at business schools: Field and laboratory studies. *Marketing Science*, 27(2):262–282, 2008.

[29] David Kurokawa, Ariel D. Procaccia, and Junxing Wang. Fair enough: Guaranteeing approximate maximin shares. *Journal of the ACM*, 65(2), feb 2018.

[30] Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC)*, pages 125–131, 2004.

[31] Hervé Moulin. *Fair Division and Collective Welfare*. MIT Press, 2003.

[32] National Center for Education Statistics. Undergraduate enrollment. condition of education. Technical report, U.S. Department of Education, Institute of Education Sciences, 2023. retrieved January 2025.

[33] Ana-Maria Nogareda and David Camacho. Optimizing satisfaction in a multi-courses allocation problem combined with a timetabling problem. *Soft Computing*, 21(17):4873–4882, 2017.

[34] Abraham Othman, Tuomas Sandholm, and Eric Budish. Finding approximate competitive equilibria: efficient and fair course allocation. In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, page 873–880, 2010.

[35] Renato Paes-Leme. Gross substitutability: An algorithmic survey. *Games and Economic Behavior*, 106:294–316, 11 2017.

[36] Antonio Romero-Medina and Matteo Triossi. Strategic priority-based course allocation. *Journal of Economic Behavior & Organization*, 226:106701, 2024.

[37] Tayfun Sönmez and M Utku Ünver. Course bidding at business schools. *International Economic Review*, 51(1):99–123, 2010.

[38] Ermis Soumalias, Behnoosh Zamanlooy, Jakob Weissteiner, and Sven Seuken. Machine learning-powered course allocation. In *Proceedings of the 25th ACM Conference on Economics and Computation (EC)*, page 1099, 2024.

[39] Vignesh Viswanathan and Yair Zick. A general framework for fair allocation with matroid rank valuations. In *Proceedings of the 24th ACM Conference on Economics and Computation (EC)*, pages 1129–1152, 2023.

[40] Vignesh Viswanathan and Yair Zick. Yankee swap: a fast and simple fair allocation mechanism for matroid rank valuations. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 179–187, 2023.

[41] Max Westphal. Simultaneous Inference for Multiple Proportions: A Multivariate Beta-Binomial Model. *arXiv preprint arXiv:1911.00098*, 2019.

[42] Wharton. *Course match user manual*, 2020. URL https://mba-inside.wharton.upenn.edu/

wp-content/uploads/2020/08/Course-Match-manual-08152020.pdf.

## A Synthetic Student Generation

We let $S = \{1, \ldots, s\}$ be the set of respondents and $G = \{g_1, \ldots, g_m\}$ be the set of courses. For each course $g \in G$, respondent $i \in S$ provides a response in the range $\{1, \ldots, 8\}$ (where 8 corresponds to a required class). We normalize these responses to a preference vector for respondent $i$, $\vec{\theta^i} \in [0,1]^m$. Fundamentally, we assume that each respondent will randomly formulate binary preferences over the set of classes such that the normalized value $\theta^i_g \in [0,1]$ represents the probability that respondent $i$ wants to take the class $g$.

Using the preference vector $\vec{\theta^i}$, we define an inference procedure that produces a distribution over preference vectors that are likely similar, but not equal, to $\vec{\theta^i}$. The process begins by sampling binary vectors $\ell$ times according to probabilities $\vec{\theta^i}$, which form a data matrix $\mathcal{D}^i \in \{0,1\}^{\ell \times m}$; if we take the limit $\ell \to \infty$, $\mathcal{D}^i$ uniquely describes $\vec{\theta^i}$. Using the matrix $\mathcal{D}^i$, we infer a *multivariate beta* posterior distribution, $\texttt{mBeta}(\vec{\gamma^i})$, as defined in Westphal [41]. The vector $\vec{\gamma^i} \in [0,1]^{2^m}$ is an implicit parameter that captures dependencies among course preferences. In order to avoid combinatorial blowup, the inference process is adjusted so that these dependencies are limited to covariances. We refer to the random vector $\vec{\vartheta^i} \sim \texttt{mBeta}(\vec{\gamma^i})$ as the $i$-th *random student* and the $j$-th realization $\vec{\sigma}^{ij} \in [0,1]^m$, drawn from $\texttt{mBeta}(\vec{\gamma^i})$, as a *synthetic student*. As $\ell$ increases, each synthetic student $\vec{\sigma}^{ij}$ will differ less and less from $\vec{\theta^i}$, the original preference vector for the respondent.

We model the population of students for each academic status (e.g. freshmen) using *kernel density estimation* (KDE). This distribution is constructed as a uniform mixture of the distributions $\texttt{mBeta}(\vec{\gamma}^1), \ldots, \texttt{mBeta}(\vec{\gamma}^s)$, and may be efficiently sampled by drawing one of the kernels $\texttt{mBeta}(\vec{\gamma^i})$ uniformly at random, and then sampling from it. Because the number of rows $\ell$ in $\mathcal{D}^i$ determines the number of samples drawn from the marginal Bernoulli distribution $\texttt{Bern}(\theta^i_g)$ for each class $g$, $\ell$ can be seen as a smoothing parameter in much the same way that *bandwidth* controls kernel smoothing in conventional KDE.

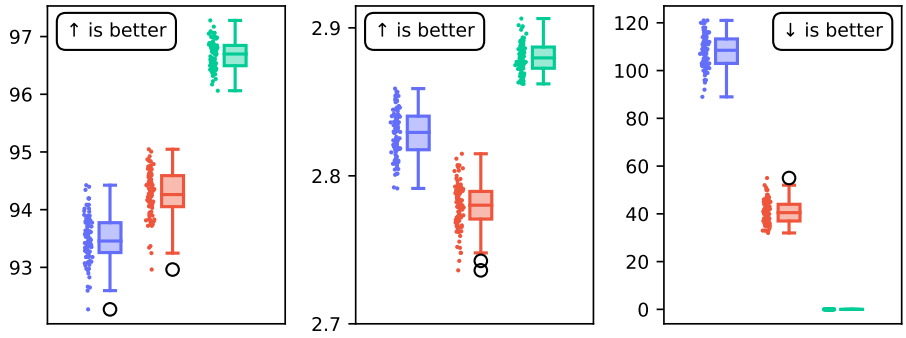
Survey respondents also specify a *course max*, or the maximum number of courses they are interested in taking. For each academic status, we model the course max preferences independently from course preferences. We first fit a multinomial distribution to the set of course max preferences among respondents for that status. Then, for each synthetic student $\vec{\sigma}^{ij}$ drawn according to $\vec{\vartheta^i}$, we independently draw a course max preference from the multinomial distribution.

## B Runtime Analysis for Yankee Swap with Item Multiplicity

Let $q_{\text{total}} = \sum_{g \in G} q(g)$ be the total number of item copies. As mentioned in Section 4, a naive implementation of Yankee Swap searches for a shortest path in the exchange graph in time quadratic in $q_{\text{total}}$. By redefining the exchange graph, we reduce the runtime dependency for searching a shortest path, which is now quadratic on $m$. Second, it turns out that calls to student valuations can be expensive; thus, our goal is to reduce the number of student valuation calls.

We assume that each agent is limited to a maximum clean bundle size of $c_{\max}$, i.e., $v_i(X_i) \leq c_{\max}$ for all $i \in N$ and any possible bundle $X_i$. Similarly, we assume that each agent can get a positive marginal utility of at most $d_{\max}$ items: $|D_i| \leq d_{\max}$ for any $i \in N$. Finally, we define $p_{\max}$ as the maximum length of any transfer path.

We now analyze the time complexity of Algorithm 1 from Section 4. Since the allocation $X$ is a matrix, updating an agent's bundle can be done in $O(1)$ time. In each iteration, the algorithm either allocates an unassigned item or removes an agent from the game. Thus, the outer loop of the algorithm runs at most




**(a)** % of seats assigned **(b)** Nash welfare **(c)** Num. of 0 util. agents

**Figure 5:** Welfare metrics for a full-size instance with 2,308 students. We compare SD (in blue), RR (in red), and YS (in green).

$q_{\text{total}} + n$ times. Identifying the unhappiest agent can be done in $O(\log n)$ time and, given that there are $O(m)$ nodes in the exchange graph $\mathcal{G}(X)$, we can find the shortest path in $O(m^2)$ time. Path augmentation is the most expensive part of the algorithm. Let $q_{\max} = \max_{g \in G} q(g)$ be the largest number of copies of any individual item. Since $R_{g \to g'}$ can have at most $q(g)$ elements, determining whether an agent is willing to exchange $g$ for $g'$ takes $O(\log q_{\max})$ time. Hence, phase 1 of the path augmentation can be computed in $O(p_{\max} d_{\max} \log q_{\max})$ time, and phase 2 in $O(p_{\max} d_{\max} c_{\max} (\log q_{\max} + \tau))$ time. Combining these observations yields following result.

**Theorem 1.** *Algorithm 1 runs in* $O\left((q_{total} + n)(\log n + m^2 + p_{\max} d_{\max} c_{\max} (\log q_{\max} + \tau)\right)$

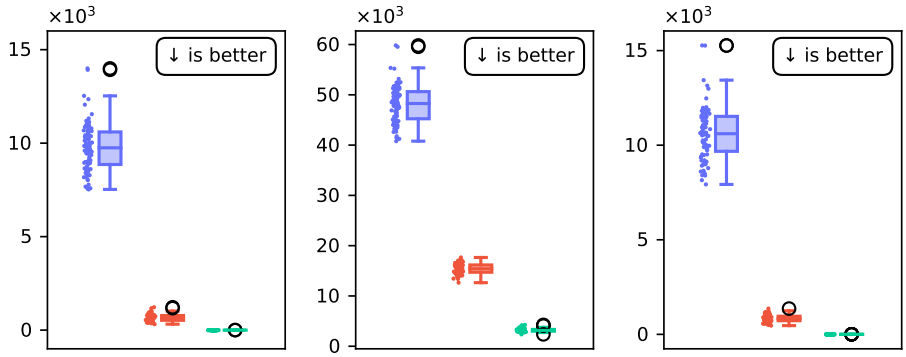
In the context of course allocation, $c_{\max} = 6$ is reasonable, since this is the maximal number of classes students are allowed to take in a single semester at University of Massachusetts Amherst. Students typically assign a high score to significantly fewer classes than $m$, thus $d_{\max}$ is also $\ll m$. While transfer paths could potentially be of length $m$, they are far shorter in practice, often consisting of no more than two or three item swaps. Finally, and maybe more importantly, separating the exchange graph representation from the agents through the tuple $R$ considerably reduces the number of student valuation function calls.

## C Additional Experiments

In Section section 6, we evaluated the performance of our four benchmark algorithms on a reduced instance derived from real student data. Here, we extend our analysis to a full-scale setting by augmenting the real survey responses with synthetic students to match the actual size of the department. We focus on the three scalable sequential algorithms — SD, RR, and YS. Additionally, we examine the behavior of the system under increased demand by artificially injecting additional students while keeping course capacities fixed. We evaluate performance in terms of utilitarian social welfare (`USW`) and the number of students receiving empty bundles.

### C.1 Full-Scale Evaluation

We run a full-scale experiment using the three scalable algorithms: SD, RR, and YS. To construct a realistic instance, we use all real survey responses and augment them with synthetic students to match the actual number of CS majors at University of Massachusetts Amherst, resulting in a population of 2,308 students. We generate 100 different samples of synthetic students and run the algorithms on each instance using a fixed student order.




**(a)** PMMS violations **(b)** EF violations **(c)** EF-1 violations

**Figure 6:** Fairness metrics for a full-size instance with 2,308 students

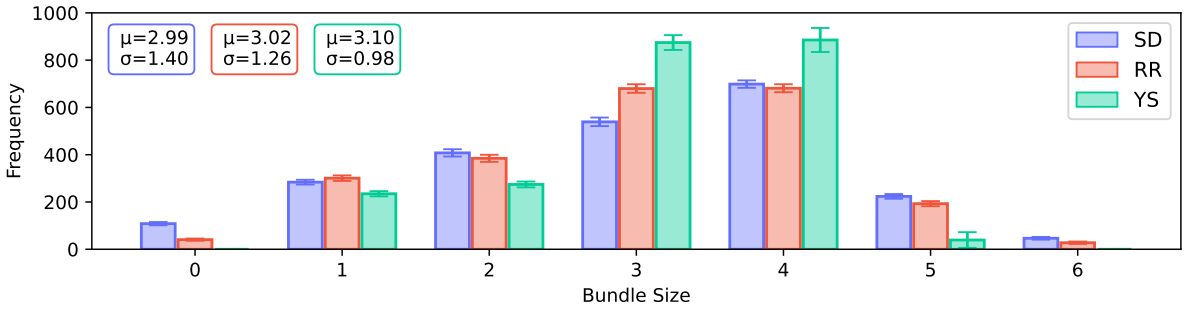



**Figure 7:** Averaged distribution of bundle sizes in experiments with 2,308 students.

YS, SD and RR exhibit similar behavior on both welfare (Figure 5) and fairness (Figure 6) metrics. SD is the least fair and efficient of the three methods, YS offers the best guarantees, and RR falls somewhere in between.

Figure 7 shows a histogram of bundle sizes for full-size instances. Again, both the serial dictatorship and the Round Robin mechanisms offer significantly greater disparities as compared to the Yankee Swap framework; indeed, as Figure 5c shows, $\sim$ 110 students receive an empty bundle under SD, $\sim$ 40 receive an empty bundle under RR, and no student receives an empty bundle under YS. Yankee Swap bundles are again heavily concentrated around 3-4 seats, which indicates a more equitable seat distribution.

### C.2 Simulating High-Demand Scenarios

How well do different mechanisms respond to increased student demand? To evaluate how different allocation mechanisms behave under increased student demand, we conducted 10 simulations per cohort size while maintaining real course capacities. This allows us to analyze what happens when we increase the number of students without adding more course seats. As shown in Figure 8, all three algorithms perform well in terms of utilitarian welfare, with YS slightly outperforming the others in terms of `USW`.

However, a different trend emerges when considering empty bundles (Figure 9). YS makes a significant effort to ensure every student receives at least one item, as reflected in the curve's shape, which suggests that YS prioritizes allocating each student at least one desirable class, until demand completely outstrips supply. Given that there is a total of 7,389 seats, enrolling more than 7,389 students inevitably leads to empty bundles. While RR also mitigates empty bundles, SD exhibits a linear increase in empty bundles as system stress rises.

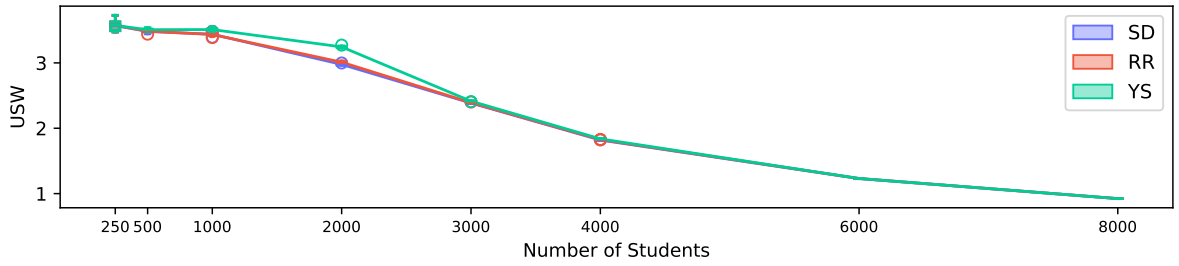



**Figure 8:** `USW` as a function of the number of students, while maintaining real course capacities.

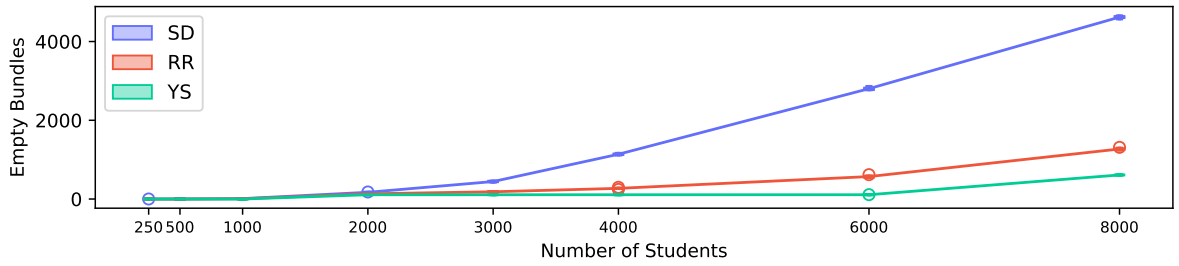



**Figure 9:** Number of empty bundles as a function of the number of students, while maintaining real course capacities.