Course Match: A Large-Scale Implementation of Approximate Competitive Equilibrium from Equal Incomes for Combinatorial Allocation*

Eric Budish† Gérard P. Cachon‡ Judd B. Kessler§ Abraham Othman¶

June 3, 2016

Abstract

Combinatorial allocation involves assigning bundles of items to agents when the use of money is not allowed. Course allocation is one common application of combinatorial allocation, in which the bundles are schedules of courses and the assignees are students. Existing mechanisms used in practice have been shown to have serious flaws, which lead to allocations that are inefficient, unfair, or both. A new mechanism proposed by Budish [2011] is attractive in theory, but has several features that limit its feasibility for practice: reporting complexity, computational complexity, and approximations that can lead to violations of capacity constraints. This paper reports on the design and implementation of a new course allocation mechanism, Course Match, that enhances the Budish [2011] mechanism in various ways to make it suitable for practice. To find allocations, Course Match performs a massive parallel heuristic search that solves billions of Mixed-Integer Programs to output an approximate competitive equilibrium in a fake-money economy for courses. Quantitative summary statistics for two semesters of full-scale use at a large business school (Wharton, which has about 1,700 students and up to 350 courses in each semester) demonstrate that Course Match is both fair and efficient, a finding reinforced by student surveys showing large gains in satisfaction and perceived fairness.

*Acknowledgments: A committee of faculty (Sigal Barsade, Gérard Cachon - chair, Dean Foster, Robert Holthausen, and Jagmohan Raju), staff (Peggy Bishop-Lane, Frank Devecchis, Stephan Dieckmann, Howie Kaufold, Alec Lamon, Jason Lehman, Nathan Mische, John Piotrowski, and Naomi Tschoegl) and students (Pardon Makumbe, Paul Nolen, Kathryn Scarborough, and Jessica Stoller) were responsible for the initial design and development of Course Match. Thanks is extended too for comments and feedback to the participants at the Marketplace Innovation Workshop (Columbia University, June 2015), the AMMA Conference on Auctions, Market Mechanisms and their Applications (University of Chicago, Aug 2015), and the Microsoft Research conference on Designing the Digital Economy (Oct 2015). Disclosure: the market design theory in Budish [2011] and computational procedure in Othman et al. [2010] are in the public domain. The software implementation of Course Match was funded by and is owned by Wharton. If Course Match is commercialized, then royalties could accrue to Othman and to a market design research lab overseen by Budish. The Wharton administration had no right of prior review of the present paper.

†eric.budish@chicagobooth.edu, University of Chicago

‡cachon@wharton.upenn.edu, University of Pennsylvania

§judd.kessler@wharton.upenn.edu, University of Pennsylvania

¶abrahamo@wharton.upenn.edu, University of Pennsylvania

1 Introduction

There are numerous settings in which resources must be allocated but markets with money are not permitted. Prominent examples include assigning kidneys to patients (Roth et al. [2004, 2005]), medical residents to hospitals (Roth and Peranson [1999], Roth [2002]), or students to public schools (Abdulkadiroglu and Sönmez [2003]). In many of these applications, each participant seeks one item: one kidney, one residency position, or one school. However, there are also settings in which the allocation problem is combinatoric because each participant requires a bundle of items, which increases the economic and computational complexity of the allocation problem. Workforce scheduling is a prime example; airline crews have preferences over a bundle of flights that they might be assigned, and nurses have preferences over bundles of shifts they might work. Other examples include the allocation of players to sports teams, shared scientific resources to users, and airport takeoff-and-landing slots to airlines.

The combinatorial allocation problem also arises in the context of student course scheduling, i.e., the course allocation problem. Each student generally wants more than one course, students have heterogeneous preferences across courses, students cannot attend courses that meet at the same time (or courses that they have already taken or for which they lack prerequisites), and courses have capacity limits, thereby making the seats in some courses scarce resources. This paper describes a new mechanism to solve the course allocation problem, called Course Match, and reports on its successful implementation at the Wharton School at the University of Pennsylvania ("Wharton"), a large business school with approximately 1,700 students and up to 350 courses in each semester. In addition to quantitative measures of the quality of the Course Match solution, we are able to confirm directly, with actual "before and after" survey data from Wharton students, real and substantial improvement in satisfaction and perceived fairness.

Roughly speaking, Course Match works as follows. Shortly before a semester begins, students report their preferences across the set of offered courses and each student is given an endowment of fake money. Next, using the reported preferences, endowments, course capacities, and course timetable, Course Match conducts a massive parallel heuristic search that solves billions of Mixed-Integer Programs to find a price for each course such that: (i) each student receives the best feasible schedule they can afford given their reported preferences, their endowment, and the course prices; and (ii) all course capacity constraints are satisfied.

The primary goal of a course allocation mechanism is to maximize student satisfaction. To achieve this, market designers often focus on two criteria: efficiency and fairness. In an efficient course allocation, it is not possible to make some students better off while leaving all other students equally well off. Fairness can be defined in a number of ways, but roughly speaking, a fair course allocation avoids outcomes in which some students greatly envy the course schedule of others, such as when some students have all of their most preferred courses while other students have none. Fairness is desirable in the context of course allocation because schools generally want to provide students with equal access to all courses. Although not generally emphasized in the theoretical literature, the successful implementation of an allocation mechanism also depends on its ease-

of-use. For example, the mechanism should not require students to complete an overly complex decision task nor require an excessive amount of time.

Most existing course allocation mechanisms — including both those described by theory and implemented in practice — can deliver either fair or efficient outcomes but not both, and many deliver neither. For example, in the serial dictatorship mechanism emphasized in some of the extant matching theory, students select their entire bundle of courses sequentially, generally with a random sequence. The lucky first student is assured her best schedule while the unlucky last student is relegated to select seats from a limited set of the least popular courses. Hence, while this mechanism is efficient, it scores poorly in terms of fairness. The draft mechanism used by Harvard Business School, in which students take turns choosing courses one-at-a-time rather than all-at-once (as in the drafting of professional sports teams), improves on the fairness of the dictatorship but has efficiency problems because of incentives to misreport preferences strategically (Budish and Cantillon [2012]). Auctions are used by a number of schools, including Wharton before the implementation of Course Match. With an auction, students are endowed with fake money and then bid, generally over multiple rounds, for course seats. Although auctions can be both efficient and fair in markets that use real money (e.g., selling a painting or allocating wireless spectrum), it has been shown that auctions with fake money have incentives problems that do not arise when the money is real, ultimately leading to allocations that are neither efficient nor fair (Sönmez and Ünver [2010], Krishna and Ünver [2008]). Furthermore, and even more importantly for our purposes, students reported low satisfaction with Wharton's auction mechanism.

Budish [2011] proposes a new mechanism for the combinatorial allocation problem, called the Approximate Competitive Equilibrium from Equal Incomes mechanism (A-CEEI), and demonstrates that it is has desirable properties of efficiency, fairness, and incentives. However, there are three major concerns with respect to its implementation in practice. First, because of the nature of the approximation errors, it may (and is likely to) violate course capacity constraints. This renders the solution infeasible for practice — a school may be required to abide strictly by capacity constraints. Second, the computational procedure in Othman et al. [2010] only finds solutions to the A-CEEI mechanism for "small," simulated problems, leaving open the question of whether it can be solved for an actual problem of Wharton's size in sufficient time. Third, the A-CEEI mechanism assumes students are able to report their preferences accurately. If a student is unable to report her preferences correctly, then A-CEEI would solve the "wrong" problem, which could lead to unsatisfied students. Therefore, it is unknown if A-CEEI is sufficiently robust to errors in reported preferences. Course Match addresses each of these issues. First, it adds two additional stages to the Budish [2011] mechanism so that a high-quality, feasible solution is constructed. Second, Course Match implements a software architecture that allows it to scale sufficiently to solve a Wharton-sized problem in a reasonable amount of time. Third, Course Match includes a rich preference reporting language and user interface to assist students in reporting preferences.

2 The Course Allocation Problem

The course allocation problem is to assign a set of courses to each student while satisfying capacity constraints and maximizing some notion of overall well-being. To be specific, there is a set of M courses, indexed by j, with integer capacities (the supply) $(q_j)_{j=1}^{M}$, and a set of N students. The capacity, q_j, is referred to as the "target capacity" for course j, or, for short, just the "capacity," because it is the desired (or target) maximum enrollment for the course. In practice, there also exists a maximum capacity, $\hat{q}_j$, for each course, such that a course allocation is not feasible, i.e., it cannot be implemented, if there are more than $\hat{q}_j$ students in a course, where clearly $q_j \leq \hat{q}_j$. For example, a course could have a desired maximum enrollment of 36 students, $q_j = 36$, but meet in a room that has 40 seats. The school has the option to set $\hat{q}_j = 36$, meaning that absolutely no more than 36 students can be enrolled in this course for a feasible solution. However, the school might also choose $\hat{q}_j = 40$, meaning that it is strongly preferred that there be no more than 36 students in the course, but a solution is feasible as long as there are no more than 40 students enrolled. Earlier work on the course allocation problem assumes $q_j = \hat{q}_j$, but for practical implementation, as discussed later, it is important to consider $q_j < \hat{q}_j$ as an option. That said, Course Match can solve problems with $q_j = \hat{q}_j$.

Each student i has a set $\Psi_i \subseteq 2^M$ of permissible course bundles, with each bundle containing at most $k \leq M$ courses. In the Wharton application, students are allowed to choose their own k, but, for ease of exposition, we assume, without loss of generality, that there is a common k. The set Ψ_i encodes both scheduling constraints (e.g., courses that meet at the same time) and any constraints specific to student i (e.g., prerequisites). Note that throughout the paper we use the term "course" to refer to what is sometimes in practice called a "section" or a "course section". In practice sometimes the same class material (i.e., listing in the course catalog) is offered at multiple times and/or by multiple professors; we refer to each section as a unique "course".

3 The A-CEEI Mechanism and Implementation Concerns

The A-CEEI mechanism has the following four steps. First, students report their preferences over schedules of courses. The theory assumes that student i reports her complete ordinal preferences over permissible schedules, denoted $\preccurlyeq_i$. Second, the mechanism randomly assigns to each student i a budget β_i of fake money (or "tokens"), which they use to purchase courses. The theory allows for budgets to be approximately, but not exactly, equal; budgets can be arbitrarily close together but no two budgets can be exactly the same. Third, a computer finds approximate market-clearing prices, i.e., a price p_j^* for each course j such that when each student i purchases the bundle of courses x_i^* that she likes best out of all schedules she can afford, the market approximately clears (we describe the nature of the approximation in detail below). Fourth, each student is allocated her most preferred affordable bundle given the preferences, budgets, and prices, i.e., student i is

assigned the bundle x_i^* which solves the problem

$$x_i^* = \arg\max_{\preccurlyeq_i} \left[x_i \in \Psi_i : \sum_j x_{ij} p_j^* \leq \beta_i \right]. \tag{1}$$

Note that the language we implement for reporting preferences, i.e., reporting $\preccurlyeq_i$, allows this problem to be represented as a Mixed-Integer Program (MIP).

Budish [2011] reports that A-CEEI has several attractive properties for large problems and approximately so for finite sized problems. First, it is Pareto efficient — no student can be made better off without making another student worse off. This property arises because the mechanism finds an allocation that is (approximately) a competitive equilibrium. Second, the allocation from A-CEEI satisfies a desirable fairness property. To explain, a mechanism is envy free if there do not exist two different students, i and i', such that student i prefers student i''s schedule to her own. Although A-CEEI cannot guarantee an envy free solution (indeed, no mechanism can), it does yield a solution that is "envy bounded by a single good," meaning that if student i envies student i''s schedule, then it is possible to remove a single course from student i''s schedule to eliminate the envy. Hence, the degree of envy with A-CEEI is limited; this property arises from the fact that the budgets are approximately, but not exactly, equal. (Using exactly equal budgets would be more attractive for fairness but could make it impossible to find prices that clear the market even approximately.) Finally, A-CEEI is strategy-proof, which means that it is in a student's best interest to report his or her true preferences, no matter how other students choose to report their preferences. This comes from the fact that student i's allocation x_i^* is her most-preferred bundle given her budget and the prices, and the student cannot affect either her budget (which is assigned randomly) or the prices (which depend on the preferences of *all* students, so in a large market prices are exogenous from the perspective of each individual student). As a result, a student does not need to consider the behavior or preferences of other students, which greatly simplifies the student's decision task, which in turn can help to increase satisfaction with the mechanism. Furthermore, given that the institution can assume preferences are reported truthfully, this mechanism provides useful data to better understand students, such as which courses, timeslots, or instructors they find desirable.

A-CEEI significantly improves on other mechanisms described in theory and used in practice. However, as noted above, there are three important concerns with the actual implementation of A-CEEI: (i) it is not guaranteed to find a price vector and course allocation that satisfies all capacity constraints, (ii) it may not find a solution quickly enough for a real-world sized problem, and (iii) it finds a solution that maximizes reported preferences, but is not guaranteed to find a desirable solution if students misreport or are unable to report their true preferences.

To understand the first concern with A-CEEI (feasibility), let z_j be the clearing error for course

j with price p_j:

$$z_j = \begin{cases} \sum_i x^*_{ij} - q_j & \text{if } p^*_j > 0; \\ \max\left[\left(\sum_i x^*_{ij} - q_j\right), 0\right] & \text{if } p^*_j = 0. \end{cases}$$

If the course is assigned a positive price, then the clearing error is the difference between the number of students assigned to the course (i.e., total demand) and the course's capacity. A course is *over-subscribed* if its demand exceeds its capacity, $\sum_i x^*_{ij} > q_j$, and it is *under-subscribed* if its demand is less than its capacity, $\sum_i x^*_{ij} < q_j$, and its price is strictly positive. If a course's price is zero, i.e., $p^*_j = 0$, then it can have $\sum_i x^*_{ij} < q_j$ without counting as clearing error, as is standard in the definition of competitive equilibrium. A price vector is said to clear the market (i.e., it is a clearing price vector) if it has no clearing error, i.e., $\alpha = 0$, where

$$\alpha \equiv \sqrt{\sum_j z_j^2}.$$

Unfortunately, a price vector with zero error may not exist, and has been shown not to exist for some problems. However, Budish [2011] shows that as long as no two students have precisely the same budget (i.e., there do not exist students i and j, $i \neq j$, such that $\beta_i = \beta_j$), there exists a price vector with market-clearing error of no more than $\alpha = \sqrt{kM/2}$. Students generally take 4-5 courses per semester, but some take as many as 8, so for the purpose of the worst-case bound say $k = 8$ and say an MBA program offers 300 courses so $M = 300$. In that case, the bound is achieved with a solution that has a squared clearing error no greater than $1,200 = \left(\sqrt{8 \times 300/2}\right)^2$: e.g., all 300 courses with a clearing error of 2 seats; or 48 courses with a clearing error of 5 seats and 252 courses with no clearing error; et cetera. As course capacities are not included in the bound, $\sqrt{kM/2}$, the clearing error as a fraction of the total number of available seats converges to zero quite fast as total capacity increases. But this is not sufficient for the solution to be considered feasible as is.

Clearing error due to under-subscription is not desirable (because a seat in a popular course is left unassigned), but this error is viewed as less problematic than over-subscription error. In particular, Wharton simply cannot implement a solution in which, due to over-subscription, $\sum_i x^*_{ij} > \hat{q}_j$.

The second concern with A-CEEI (computational effort) is primarily due to the complexity of the task. Recent work by Othman, Papadimitriou, and Rubinstein [2014] has proved that the combinatorial allocation problem is PPAD-complete, even with access to an oracle that can solve the student demand problem in constant time. PPAD is a complexity class that was originally developed and motivated in Papadimitriou [1994]; briefly, it represents search problems in which an algorithm can only follow successive steps down a path of indeterminate length. Many other search problems are PPAD-complete, including solving for Nash Equilibria, finding many kinds of market equilibria, finding repeated game equilibria, finding Brouwer fixed points, and detecting a completely colored node in the Sperner's lemma setting [Papadimitriou, 1994, Abbott et al., 2005, Codenotti et al., 2006, Huang and Teng, 2007, Borgs et al., 2008, Chen and Teng, 2009, Daskalakis et al., 2009, Kintali et al., 2009, Pálvölgyi, 2009, Chen and Teng, 2011, Vazirani and Yannakakis,

2011, Chen et al., 2013, Rubinstein, 2014]. An algorithm that could solve course allocation to the theoretical bound in polynomial time would also be able to solve all of these problems in polynomial time, too. However, the consensus among computer scientists is that, just like for the better known complexity class NP, there do not exist polynomial-time worst-case algorithms to solve PPAD problems [Papadimitriou, 1994, Daskalakis et al., 2009].

Although the theoretical results on computational effort are not encouraging, Othman et al. [2010] report on a Tabu-search heuristic algorithm that finds, with reasonable effort, price vectors that yield clearing error even lower than the bound in simulated problems. That algorithm examines many candidate price vectors, and uses the resulting degree of over-subscription and under-subscription to guide the search of additional price vectors. However, a real-sized problem is considerably larger than the problems solved in Othman et al. [2010].

The third concern with A-CEEI (preference reporting) is not discussed in the theoretical literature because it is simply assumed that the preference reporting language is sufficiently rich to capture a student's full set of preferences and students are able to correctly "speak" this language (i.e., they do not make errors reporting their preferences). These assumptions are unlikely to hold in practice. A real-sized problem may have 300 courses offered by 150 professors in 14 time slots, meaning that by necessity a real-world preference reporting language must be simpler than asking students to rank complete schedules ordinally from most- to least-preferred. Any simplification risks preventing students from reporting their actual preferences if their preferences cannot be expressed using the language provided. Furthermore, given the size of real problems, even a student whose preferences in principle can be expressed using the provided language may find it non-trivial to do so, perhaps especially if the provided language is rich. Hence, while A-CEEI might yield a solution that maximizes each student's reported preference given their budget, it might not maximize their actual preference, which jeopardizes student satisfaction (i.e., a student may blame themselves for the error, or, quite possibly, they could blame the mechanism).

4 The Course Match Solution

Shortly before a semester begins, Course Match elicits preferences from students. Course Match also requires a number of other inputs, including: (i) each student's budget, (ii) each course's target, q_j, and maximum capacity, $\hat{q}_j$, (iii) the meeting times for each course (a student cannot be registered for two courses that have overlapping meeting times), and (iv) the set of courses each student has already taken (because they cannot take the same course twice). Next, Course Match uses a computational engine to derive a course allocation that is reported to students about a week before classes start. A few days before classes begin, a drop/add period opens in which students, on a first-come-first serve basis, can drop a course, add a course with an open seat, or add themselves to a course waiting list that automatically advances if seats become available in the course. The primary purpose of the drop/add period is to enable students to make adjustments to their schedule in case their preferences change, especially once they start taking classes.

As described later in this section, Course Match implements a refinement of the A-CEEI mechanism. Consequently, based on the theoretical properties of A-CEEI, student budgets are set equal to a base budget plus a small idiosyncratic tie-breaking subsidy. At Wharton, the MBA students are divided into several groups: all second-year students are one group and the first-year students are divided into groups based on the semester and the number of core courses they have tested out of. (The Wharton MBA is a two-year program.) Each of the N_g students in group g is randomly assigned to a distinct tie-breaking budget surplus in $\{0.1, 0.2, \ldots, N_g/10\}$. This amount is sufficiently small that even the largest tie-breaking budget surplus is unable to increase a student's budget above the base budget of the next higher group — with 800 second-year students, the maximum tie-breaker is 80, which is 1.6% of their base budget of 5,000. Unused budgets from one semester do not carry over to subsequent semesters because allowing so would introduce incentives to misreport preferences and increase decision complexity (students would have to think about how much of their budget they want to reserve for future use).

The remainder of this section is divided into three parts. The first part describes the Course Match preference language used to elicit preferences from students. The second part details the computational engine used to derive an implementable course allocation (i.e., a solution in which none of the maximum capacity constraints are violated) for a Wharton-sized problem (computational performance is covered in Section 5). The third part discusses alternative approaches considered, and rejected, for addressing under-subscription and over-subscription.

4.1 Preference Elicitation

Although the theory of A-CEEI assumes that students report a complete ordering i over their set of valid schedules Ψ_i, in practice it is not possible for students to enumerate their preferences fully because the cardinality of Ψ_i is large. Hence, Course Match requires a simple preference language that students can understand and use to report their preferences with reasonable accuracy. Furthermore, the language must result in preferences that can be solved quickly and reliably by a computer. For our purposes, this means that they must be translatable into a MIP that finds a student's most preferred schedule at a given price vector (i.e., their most preferred affordable bundle). Course Match's elicitation procedure of translating complex human utilities into MIPs is similar to the process used in practice to elicit preferences in combinatorial auctions [Sandholm and Boutilier, 2006, Sandholm, 2007].

The reporting language in Course Match allows students to report a (cardinal) utility value for each course as well as for pairs of courses (i.e., "extra" utility value, positive or negative, associated with getting the two courses together in a schedule). Utilities for both individual courses and course pairs are weighted by credit units (CUs), the measure of how much a course counts towards a Wharton degree. Most courses are worth either 1.0 or 0.5 CUs. The student's utility for a schedule is then taken to be the (CU-weighted) sum of the student's utilities for the courses in that schedule, plus (or minus) any reported utilities for course pairs in that schedule. Schedules can be rank-ordered by their utilities to determine the student's best schedule given a price vector

and their budget, i.e., to solve (1). Observe that this language transforms problem (1) into a MIP. Following are more details on the data student input:

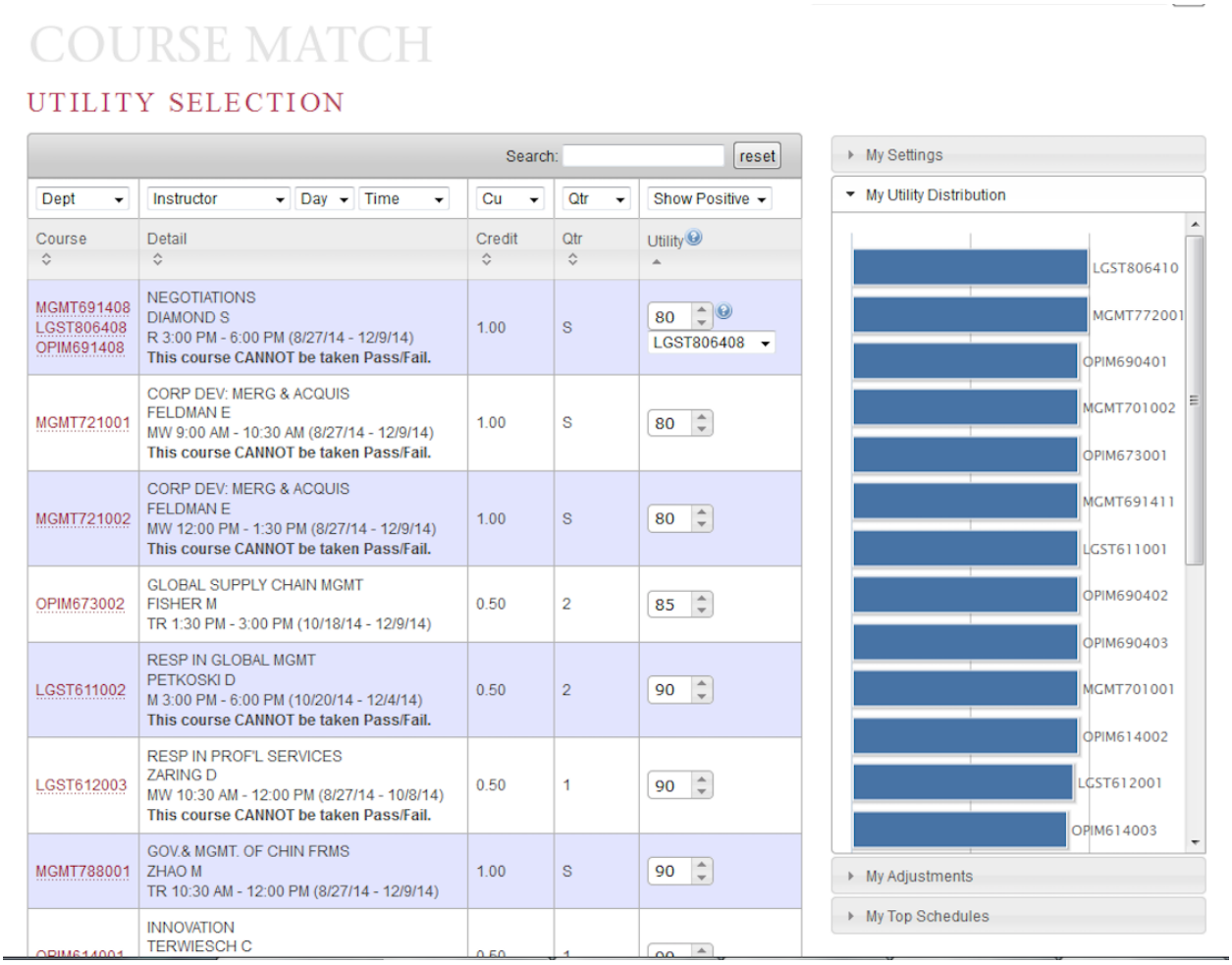
Individual Course Utilities Students report their utility (i.e., preference) for each course on a 0 to 100 integer scale: 0 means the student does not want the course and 100 means the course is the most preferred. While it is not strictly necessary to place a cap on the utility for each course, the cap provides an intuitive focal point for students. Given that the utility of a schedule is the CU weighted sum of the reported utilities, a 0.5 CU course can contribute at most 50 to the utility of a schedule, even though the student can still report utilities up to 100. This is done so that the sum of the maximum utilities for two 0.5 CU courses cannot be greater than the sum of the maximum utility for a 1.0 CU course. It was found that students preferred this approach over summing utilities without adjusting for credit units. Figure 1 displays a screen shot of the student preference reporting interface.

Pairwise Adjustments Students can select any pair of courses and apply either a positive or negative utility adjustment to the pair between -200 and 200. This allows students to express the preference that taking two courses is either more desirable than the sum of the courses individually (with a positive adjustment) or taking the two courses is less desirable than the sum of the courses individually (with a negative adjustment). For instance, if a student really wants to take an entrepreneurship and a venture capital course together, then the student can assign a high utility to each of them individually and a positive adjustment for the pair to indicate that the bundle is particularly desirable (worth more than the sum of the two individual utilities). Or, if the student wants to take either the entrepreneurship class or the venture capital class, but not both, the student can assign a negative adjustment to lower the value of the bundle in the schedule. This adjustment can be chosen so that the bundle's utility is non-positive, insuring that the bundle is never selected. While it is not necessary to have an upper bound of 200 for adjustments, it was found that unbounded adjustments created confusion with some students in preliminary tests. In particular, some students thought they could use adjustments to circumvent the upper bound on the utility for each course without realizing that utilities are relative, so the absolute scale doesn't matter. The upper bound on adjustments avoids this misunderstanding by a few students while not significantly limiting the ability of students to report their preferences. Allowing for adjustments on sets of 3+ classes was discussed but rejected; ultimately, the potential benefits from increased preference expression were judged not to be worth the additional complexity. Figure 2 shows the utility entry page with the "Adjustment" tab shown on the right hand side.

Capacity Constraints Students can specify the maximum number of credit units they wish to take in a semester.

Although students are able to think about their preferences over courses and pairs of courses intuitively, their ultimate preference is over the course schedule they receive. However, even ignoring

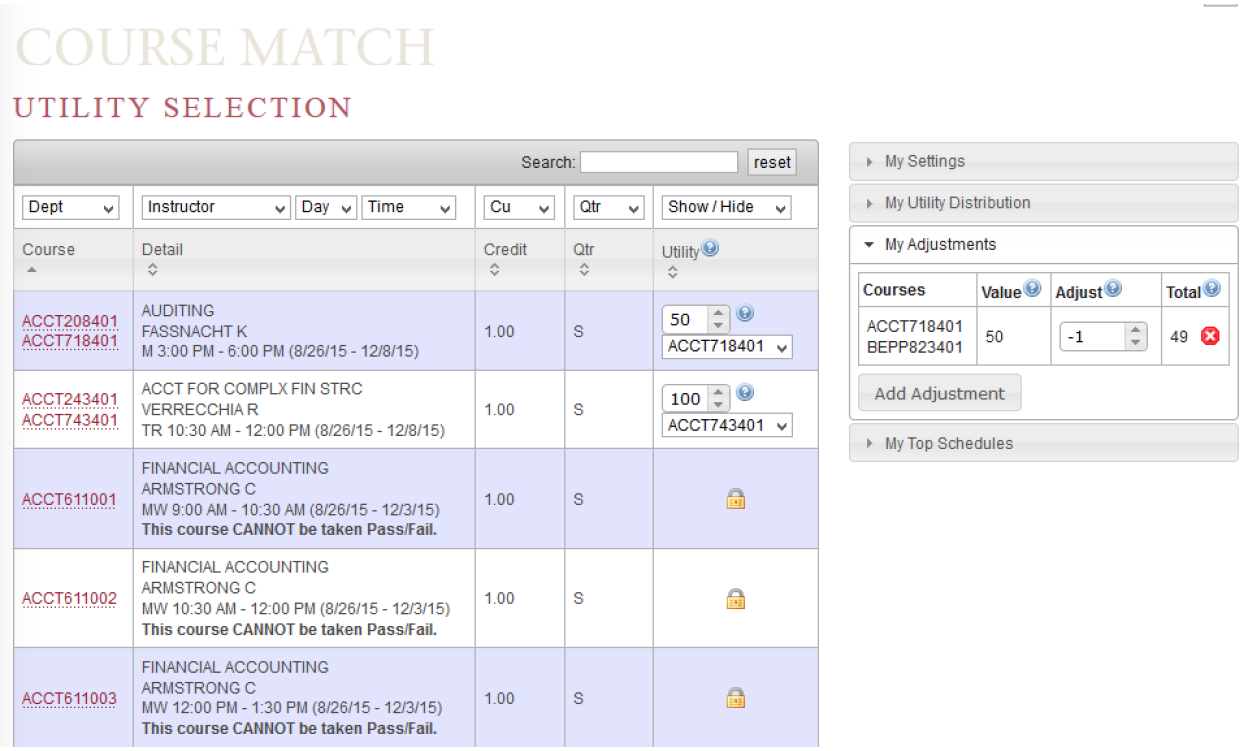
Figure 1: Main utility entry interface students use to input their preferences



Notes: The first column lists the course number (when a course is cross-listed in multiple departments, the student can choose which department she wants to take it from in the fifth column, e.g., to satisfy requirements for a major). The second column provides the course name and other course details. The third column lists the number of CUs of the course. The fourth column indicates whether the course is the full semester or whether it meets for only the first or second quarter of the semester (a semester is made of two quarters). The fifth column is where students enter their utilities. All inputs are defaulted at 0 and can be set to any integer between 0 and 100. The "My Utility Distribution" tab shows the course numbers ranked by the student's reported utilities.

the price vector and budgets, it is non-trivial for a student to map their reported preferences to a rank order over permissible schedules. Hence, the user interface also provides students with a way to view their "top schedules" given their reported utilities, as shown in Figure 3. This feature enables students to see, using a calendar view, the rank order of their schedules and the differences in utility values across schedules. For example, in Figure 3 there are relatively large gaps in utility between the first, second, and third schedules, but a small gap between the third and fourth. Students are told that they will receive the highest ranked schedule that they can afford once course prices are determined. Consequently, if they do not like the ranking of schedules as seen through this feature, they can refine their inputted course values and adjustments to better reflect their preferences over schedules.

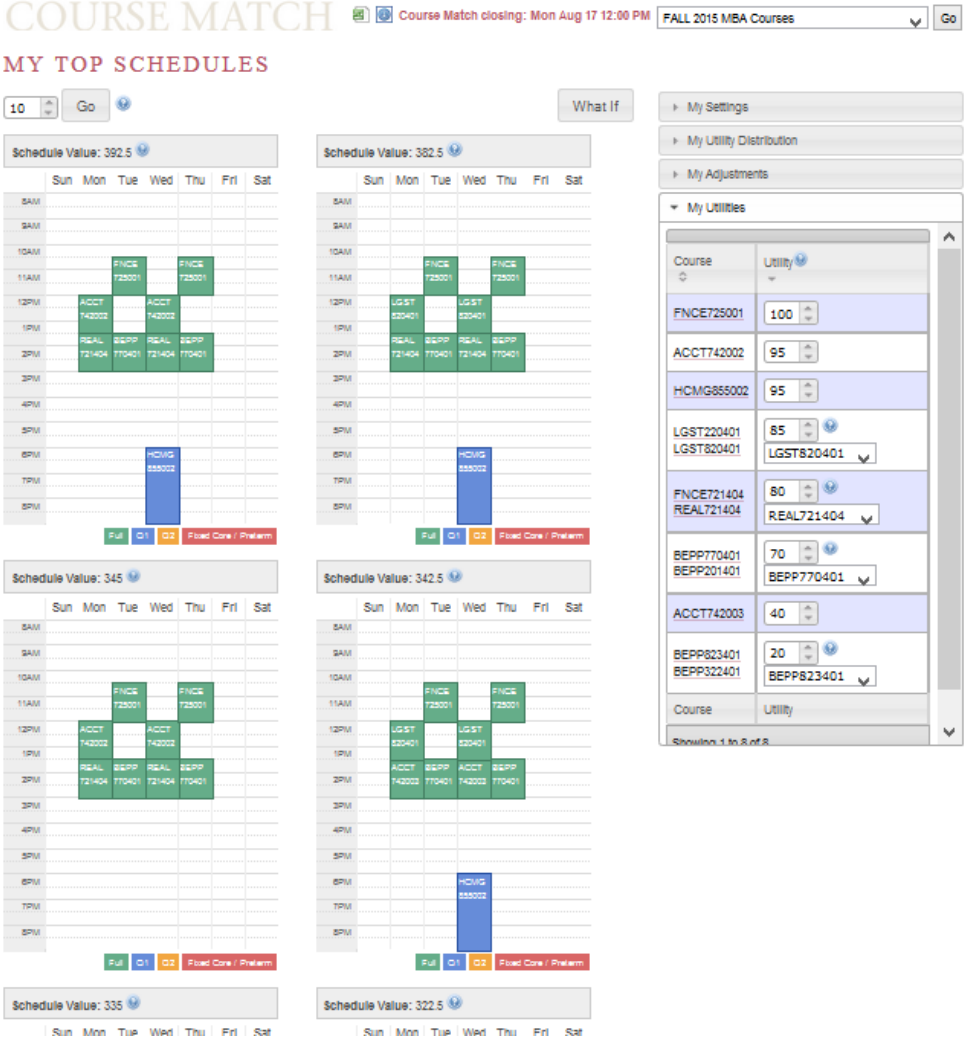
Figure 2: Alternate tab of utility placement screen showing interface for adjusting utilities placed on combinations of courses



Notes: The right panel illustrates pairwise adjustments. For the left panel see the notes to the previous figure. The lock in the fifth column indicates that the student is ineligible to take the course, e.g., because they have already taken it or lack a prerequisite.

The Course Match preference-reporting language was initially pilot tested in the laboratory [Budish and Kessler, 2015], as part of an experimental test of the overall suitability of the Budish [2011] mechanism for real-world use at Wharton.

Figure 3: A student's top schedules are generated from their reported preferences and shown in a separate step before the input is saved



4.2 Computational Engine

Course Match finds a solution to the course allocation problem with an algorithm divided into three stages. Stage 1, Price Vector Search, uses an enhanced version of the Othman et al. [2010] computational procedure to find a price vector p^* that is an approximate competitive equilibrium in the sense defined by Budish [2011]. This allocation may have both over-subscription and under-subscription errors. Stage 2, Eliminate Over-subscription, modifies the prices from Stage 1 so as to eliminate all over-subscription errors that cause violations of the strict capacity constraints (the $\hat{q}_j$ capacities); at this stage, the solution is feasible. Stage 3, Reduce Under-subscription, then attempts to reduce, to the extent possible, any under-subscription errors, without too much compromise of fairness considerations.

4.2.1 Stage 1: Price Vector Search

Stage 1 in Course Match computes the A-CEEI mechanism. To obtain a solution with minimal price clearing error below the theoretical bound, it follows a Tabu search heuristic originally developed in Othman et al. [2010]. The pseudocode for this stage is displayed in Algorithm 1.

The heuristic search is performed over the price space and is composed of a series of search starts until the allotted time for searching is reached (e.g., 48 hours). A search start proceeds through a series of steps, each with a candidate price vector, the first of which is a randomly generated price vector (line 3). With each step, a set of neighbor price vectors is generated (line 8). Each neighbor is an intuitive permutation of the candidate price vector. Neighbors that yield a course allocation identical to one of the previous candidate price vectors are dropped. This "Tabu" component of the search prevents visiting the same effective spot in the search space multiple times (i.e., even if two price vectors are not identical, if they generate identical course allocations, then they are effectively identical). The remaining neighbor with the lowest clearing error, based on the target capacities q_j, is selected as the new candidate (line 20), even if its clearing error is greater than the clearing error of the previous candidate price vector. This allows the search process to explore other regions of what is presumed to be a rugged landscape. However, the search start terminates if there is no improvement in the best clearing error across five consecutive steps (i.e., candidate price vectors) or if the allotted time is reached. If a search start terminates and time remains, another search start is initiated. Search starts are independent of each other in the sense that they might adopt equivalent price vectors, i.e., the Tabu list of visited price vectors is cleared with each search start (line 5).

As in Othman et al. [2010], neighbors are composed of the union of two distinct sets of neighborhoods:

Gradient Neighbors In a gradient neighbor, the price of every course is adjusted proportionally to its number of seats of under-subscription or over-subscription. So a course that is over-subscribed by four seats will see its price raised twice as much as a course over-subscribed by two seats, and the neighborhood is formed by considering a number of potential step sizes

Algorithm 1 Heuristic search algorithm through price space, originally developed in Othman et al. [2010].

Input: $\bar{\beta}$ the maximum student budget, $d\colon \mathbf{p} \mapsto \mathbb{Z}^M$ course demands at a price vector, $N\colon \mathbf{p} \mapsto (\tilde{\mathbf{p}}_k)_{k=1}^K$ function that generates the set of neighbors of a price vector sorted by clearing error α^2 ascending, t overall time limit

Output: $\mathbf{p}^*$ price vector corresponding to approximate competitive equilibrium with lowest clearing error.

besterror $\leftarrow \infty$ ▷ besterror tracks the best error found over every search start
repeat
  $\mathbf{p} \leftarrow \left(U[0,1] \cdot \bar{\beta}\right)_{j=1}^M$ ▷ Start the search from a random, reasonable price vector
  searcherror $\leftarrow \alpha(d(\mathbf{p}))$ ▷ searcherror tracks the best error found in this search start
  $\tau \leftarrow \emptyset$ ▷ τ is the *tabu list*
  $c \leftarrow 0$ ▷ c tracks the number of steps taken without improving error
  while $c < 5$ **do** ▷ Restart the search if we haven't improved our error in five steps
    $\mathbb{N} \leftarrow N(\mathbf{p})$ ▷ This requires evaluating the clearing error of each neighbor
    foundnextstep $\leftarrow$ **false**
    repeat
      $\tilde{\mathbf{p}} \leftarrow \mathbb{N}$.pop() ▷ Remove the front of the neighbor list
      $\mathbf{d} \leftarrow d(\tilde{\mathbf{p}})$
      if $\mathbf{d} \notin \tau$ **then** ▷ If $\tilde{\mathbf{p}}$ does not induce demands found in our tabu list, it becomes the next step in our search
        foundnextstep $\leftarrow$ **true**
      end if
    until foundnextstep **or** $\mathbb{N}$.empty()
    if $\mathbb{N}$.empty() **then**
      $c \leftarrow 5$ ▷ All neighbors are in the Tabu list; force a restart
    else ▷ $\tilde{\mathbf{p}}$ has the next step of the search
      $\mathbf{p} \leftarrow \tilde{\mathbf{p}}$
      τ.append($\mathbf{d}$)
      currenterror $\leftarrow \alpha^2(\mathbf{d})$
      if currenterror $<$ searcherror **then**
        searcherror $\leftarrow$ currenterror
        $c \leftarrow 0$ ▷ We improved our search solution, so reset the step counter
      else ▷ We didn't improve our solution from this search start, so add to the step counter
        $c \leftarrow c + 1$
      end if
      if currenterror $<$ besterror **then**
        besterror $\leftarrow$ currenterror
        $\mathbf{p}^* \leftarrow \mathbf{p}$
      end if
    end if
  end while
until current time $> t$

along this gradient vector (we used up to 12). These steps can be thought of as a tâtonnement performed by a Walrasian auctioneer.

Individual Adjustment Neighbors Each neighbor of this type is the product of changing the price of a small set of courses. Let C be the number of under or over-subscribed courses. To limit the number of neighbors, $\min\{C, 40\}$ neighbors are created. If $C \leq 40$, then each of the C neighbors adjusts the price of a single course. If $40 < C$, then the C courses are evenly (and randomly) assigned to 40 neighbors.[1] With each neighbor, the price of an over-subscribed course slated for adjustment is increased to reduce its demand by exactly one student, while the price of an under-subscribed course slated for adjustment is dropped to zero.

There exists a time constraint in Course Match because both administrators and students expect to see schedules for a semester produced within a few days of submitting preferences. However, an enormous amount of computational work is required to find an approximate equilibrium in the heuristic search. To give a sense of the computational effort required in a Wharton-sized problem, evaluating the clearing error at each candidate price vector requires solving approximately 1,700 MIPs (one for each student), each iteration requires evaluating approximately 50 neighbors, a typical search start may take 100 steps (i.e., candidate price vectors), and an entire Course Match run may perform 500 search starts. So in total, solving a Wharton-sized problem requires computing solutions to about 4.25 billion MIPs. If it takes one millisecond to solve each MIP — an optimistic assumption in practice — finding an approximate competitive equilibrium would still take about seven weeks.

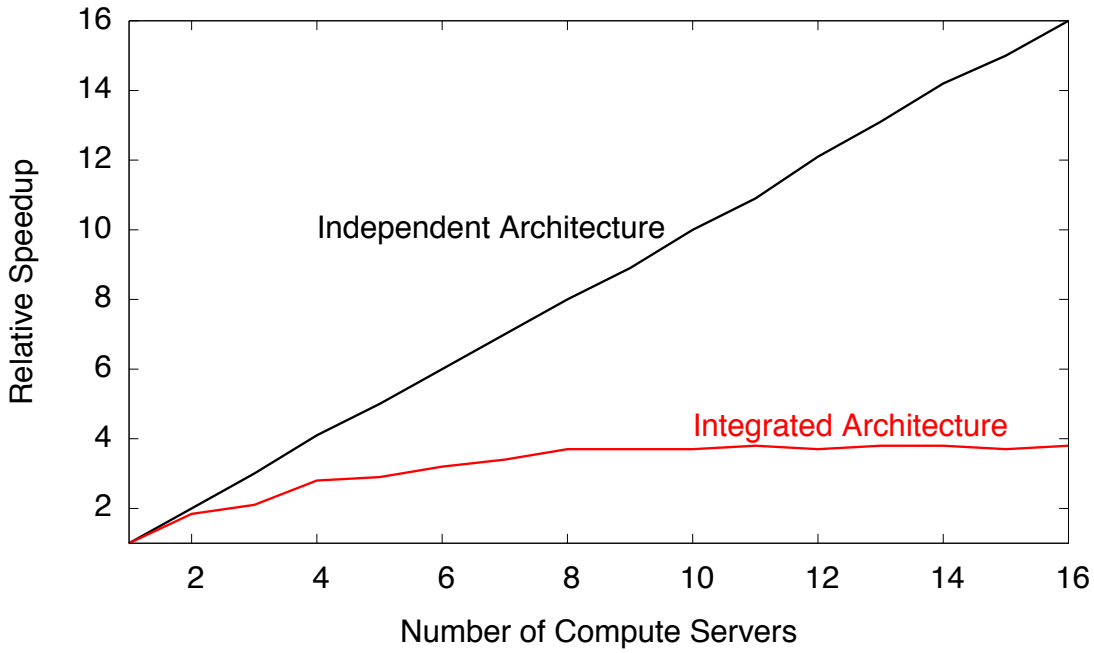
To reduce the time needed to find a solution in Stage 1, it is possible to parallelize some of the work. The natural point to parallelize is the evaluation of each of the neighbor price vectors with each step of a search start because given a price vector, each student's MIP is independent of all the other MIPs that need to be solved. Thus, at each step approximately 85,000 independent MIPs need to be solved (1,700 MIPs per price vector and 50 price vectors). In the software architecture described in Othman et al. [2010], those MIPs are solved on a single compute server consisting of distinct cores. In Course Match each compute server has 32 cores. About 3 cores are needed to perform non-MIP solving tasks, leaving about 29 cores dedicated to solving MIPs. Hence, with each search step those 29 cores can simultaneously solve MIPs, with each assigned approximately 2,931 MIPs (85,000 / 29). This approach does not scale linearly, but Othman et al. [2010] show scaling at 90% efficiency on multiple cores of the same compute server, i.e., using n cores is $0.9n$ times faster than one core. Nevertheless, when we ran computational experiments to evaluate this single-server architecture, we found significant potential gains in solution quality from using more computational power than a single server could provide in the 48 hours allotted. (A single compute

[1] The original Othman et al. [2010] procedure did not limit the number of neighbors in this way (i.e., instead of creating $\min\{C, 40\}$ neighbors it created C neighbors). Our exploratory analysis suggested that on Wharton-sized problems, bundling neighbors yielded more search starts that lowered error faster and terminated faster. Bundling neighbors provides two ways to accelerate a search in its intermediate stages: reducing the computation required at each candidate price vector and making several steps simultaneously.

server can complete about 60 search starts in 48 hours, and we found in our experiments that market-clearing error continued to go down with additional search well beyond 60 search starts.)

We explored two ways to add additional computational capacity to improve on the performance of the Othman et al. [2010] architecture. The first, which we call the integrated architecture, extends Othman et al. [2010] in an intuitive way from one compute server to a cluster of compute servers — maintain a few cores to perform non-MIP solving tasks and dedicate the remaining cores, across different compute servers, to solve MIPs. With the integrated architecture, all of the cores within the cluster are working on the same search step of the same search start, i.e., at any given moment, the cluster is working on a single search start. The second approach, which we call the independent architecture, operates multiple compute servers independently within a cluster. In this case, a cluster of n compute servers works on n different and independent search starts at the same time. Furthermore, each compute server must dedicate several cores to non-MIP solving tasks, leaving fewer cores within the cluster to work on MIPs relative to the integrated architecture. However, with the independent architecture there is no need for communication across compute servers because they operate independently. In contrast, with the integrated architecture, communication must occur across compute servers because they are working on the same search start.

Figure 4: Comparison of horizontal scalability between running a distinct search on each compute server (an independent architecture) versus running a single search using all compute servers (an integrated architecture)



The additional communication overhead with the integrated architecture is substantial and significantly decreases the ability to use additional compute servers to speed up the process. To illustrate, we compared both the integrated and independent architectures by performing the Course

Match search for a half hour on each of a progressively increasing number of Amazon `cc2.8xlarge` compute servers. Figure 4 shows the relative speedup (measured in terms of the number of MIPs solved in a half hour) using each approach. The independent architecture is able to achieve a linear speedup in the number of compute servers used — it runs 16.0 times faster when 16 compute servers are used. In contrast, the integrated architecture appears to plateau at 3.8 times the speed of a single compute server when using eight or more servers. This finding is consistent with other results on the relative performance of increasing the number of compute servers when parallelizing complex algorithms (e.g., Sun and Rover [1994]). Thus, Course Match adopts the independent architecture to achieve the required computational speed.

4.2.2 Stage 2: Eliminate Over-subscription

The heuristic search in Stage 1 outputs a price vector that has market-clearing error below the theoretical bound. However, this error may consist of both over-subscription and under-subscription, and over-subscription can cause a solution to be infeasible. That is, it is possible that the price vector from Stage 1 results in an allocation in which some course's maximum capacity constraint $\hat{q}_j$ is violated. The goal of Stage 2 is to transform the Stage 1 solution into one that does not violate any maximum capacity constraint, i.e., a solution that can feasibly be implemented in practice.

In the context of the Stage 2 algorithm, a course is said to be over-subscribed if it is assigned strictly more than $\hat{q}_j$ students. To eliminate over-subscription, Stage 2 relies on the property that starting from any given price vector, demand for any single course j is monotonically decreasing in course j's price. To be specific, Stage 2 iteratively identifies the most over-subscribed course and then raises its price to eliminate half of its over-subscription. It finds the necessary price increase via a binary search. Pseudocode for Stage 2 is given below as Algorithm 2.

Algorithm 2 Iterative over-subscription elimination algorithm, reducing by half the excess demand of the most over-subscribed course with each pass.

Input: $\mathbf{p}^*$ heuristic search solution price vector from Algorithm 1, $\bar{p}$ scalar price greater than any budget, ϵ smaller than budget differences, excess demand function $\hat{d}(\mathbf{p})$ that maps a price vector to the demand of a course beyond its maximum capacity.

Output: Altered $\mathbf{p}^*$ without over-subscription

1: $j' \leftarrow \arg\max_j \hat{d}_j(\mathbf{p}^*)$ ▷ j' is the most over-subscribed course
2: **while** $\hat{d}_{j'}(\mathbf{p}^*) > 0$ **do**
3: $d^* \leftarrow \lfloor \hat{d}_{j'}(\mathbf{p}^*)/2 \rfloor$ ▷ Perform binary search on the price of course j' until over-subscription equals (at most) d^*.
4: $p_l \leftarrow p^*_{j'}$
5: $p_h \leftarrow \bar{p}$
6: **repeat** ▷ Our target price is always in the interval $[p_l, p_h]$, which we progressively shrink in half in each iteration of this loop
7: $p^*_{j'} \leftarrow (p_l + p_h)/2$
8: **if** $\hat{d}_{j'}(\mathbf{p}^*) \geq d^*$ **then**
9: $p_l \leftarrow p^*_{j'}$
10: **else**
11: $p_h \leftarrow p^*_{j'}$
12: **end if**
13: **until** $p_h - p_l < \epsilon$
14: $p^*_{j'} \leftarrow p_h$ ▷ Set to the higher price to be sure over-subscription is at most d^*
15: $j' \leftarrow \arg\max_j \hat{d}_j(\mathbf{p}^*)$ ▷ Find the most over-subscribed course with the new prices
16: **end while**

Although a course's over-subscription always decreases when its price is raised, its over-subscription can increase again at a later step in the algorithm when another course's price is increased. Nevertheless, because prices are only increased and there exists a vector of suitably high prices at which no courses are over-subscribed, Algorithm 2 eventually terminates with a solution that has no over-subscription.

The choice to eliminate half of over-subscription with each iteration is somewhat arbitrary but was selected to balance the runtime and efficiency concerns that emerged from the choices at each extreme. For instance, at one extreme, only a single student could be removed from the most over-subscribed course, i.e., line 3 of Algorithm 2 could be $d^* \leftarrow \hat{d}_{j'}(\mathbf{p}^*) - 1$. In our exploratory analysis, this was found to be very slow in practice, requiring a huge number of iterations as prices were slowly raised between sets of desirable complementary courses. At the other extreme, over-subscription could be eliminated entirely from the most over-subscribed course, i.e., line 3 of Algorithm 2 could be $d^* \leftarrow 0$. While this raises the price of courses quickly and produces a feasible solution in fewer iterations, we found that it produced allocations with very high clearing error, because it can make price adjustments that are too large and therefore yield higher course prices than necessary to get a solution without over-subscription.

4.2.3 Stage 3: Reduce Under-subscription

After Stage 2 eliminates over-subscription, the solution is feasible, but it is now likely to have a considerable amount of under-subscription error, i.e., demand in positively priced courses less than the target capacity, $\sum_i x^*_{ij} < q_j$. If this solution were adopted, then the empty seats in positively priced courses are likely to be acquired quickly in the drop/add period at the start of the semester. There are several reasons why it is not desirable to have under-subscribed seats acquired that way. First, a student with a strong preference for a seat in a popular course may lose the seat to a student with less of an interest in the course, which works against the goal of trying to allocate courses in such a way that the seats in each course are assigned to the students with the strongest preference for the course. Second, as the Course Match budgets are no longer relevant in the drop/add period, a student may be able to acquire a seat in a popular course for essentially zero cost. In that case, the student obtains a better schedule than would have been affordable within Course Match, creating concerns around actual and perceived fairness. Third, the drop/add period rewards students who are able to participate at the time it opens, and at least some students are likely to be unable to participate at any selected time in the days before the semester begins.

The goal of Stage 3 is to provide a better solution to under-subscription than the drop/add period (or a similar aftermarket). Ideally, empty seats in positively priced courses would be assigned to the students with the strongest preferences for those seats, while not creating an incentive for students to misrepresent their initial preferences to Course Match. With that guiding precept in mind, the Stage 3 algorithm appears in pseudocode below as Algorithm 3. To simplify notation in the pseudocode, we define a special choice function for each student i, $\breve{x}^*_i$. This choice function takes as arguments the student's initial budget, β_i, and the set of courses formed by taking the union of the courses in the student's initial allocation and the courses that currently have open seats, denoted $\mathbb{X}_i$. The choice function then returns the student's most preferred affordable schedule out of the courses in this set $\mathbb{X}_i$, using the prices p^* found in Stage 2, the student's original reported preferences $\preccurlyeq_i$, and the student's new budget $\breve{\beta}_i$ (updated from β_i by increasing the budget in a way described below). Since the student's new budget will be higher than their original budget, the student can always afford their schedule as of the end of Stage 2, i.e., their allocation can only improve from what they received in Stage 2. Formally, the choice function is defined as:

$$\breve{x}^*_i(\mathbb{X}_i, \breve{\beta}_i) \equiv \arg\max_{\preccurlyeq_i} \left[x_i \in \Psi_i : \sum_j x_{ij} p^*_j \leq \breve{\beta}_i, x_{ij} \in \mathbb{X}_i \right].$$

Stage 3 iteratively selects students and assigns a selected student the best schedule the student can afford from the set of seats that are in the student's current schedule and in courses with open seats (i.e., enrollments less than target capacity). After a student selects these seats, they are no longer available for other students. By definition, a student's schedule after Stage 2 is best for the student with their current budget. Hence, if there were no change in budgets, there would be no change in the allocated courses. Consequently, in Stage 3 each student is awarded

Algorithm 3 Automated aftermarket allocations with increased budget and restricted allocations.

Input: Input allocations $x_{ij} = 1$ if student i is taking course j, restricted demand functions $\breve{x}_i^*(\mathbb{X}_i, \breve{\beta}_i)$, S students ordered by class year descending and then by budget surplus ascending.
Output: Altered allocations x_{ij}.

repeat
  done $\leftarrow$ **true**
  $\mathbb{U} \leftarrow \left[\text{Course } j \in M : \sum_j x_{ij} < q_j\right]$. ▷ $\mathbb{U}$ is the set of currently under-subscribed courses
  for Student $i \in S$ **do** ▷ Iterate over the students in a fixed order
    $x_i' \leftarrow \breve{x}_i^*(\mathbb{U} \cup x_i, 1.1 \times \beta_i)$ ▷ Re-optimize over a restricted set of courses with 10% more budget
    if $x_i \neq x_i'$ **then**
      done $\leftarrow$ **false**
      $x_i \leftarrow x_i'$
      break ▷ Break out of the **for** loop, so that only one student changes their allocation in each pass
    end if
  end for
until done ▷ Done only if we do a pass without any student changing their allocation

a 10% increase in their budget. The objective of this budget subsidy is to allow many students to receive a reasonable improvement in their schedules rather than to have a few students receive large improvements. Furthermore, it is important to emphasize that students are not allowed to acquire seats in full courses even if their extra budget would allow them to afford such seats at the current prices. In sum, Stage 3 sacrifices some equity (because students early in the selection process have a wider selection of courses on which to spend their new higher budget) to improve upon efficiency (reduce under-subscription) while using the rich preference data to allocate those seats in a rational manner. Note as well that the addition of Stage 3 does not affect the conclusion that the mechanism is strategy-proof in the large.[2]

A student's course allocation after Stage 3 may cost more than the student's initial budget, meaning that students may "receive schedules that are even better than what they could afford." However, this is not viewed as a concern for four reasons:

1. Not every student uses the extra budget nor do they use the entire budget (e.g., a 1% increase in their budget may be sufficient to get their most preferred schedule);

2. The Stage 2 prices are in some sense too high (because there is only under-subscription and no

[2]The argument that Course Match formally satisfies the incentives criterion of strategy-proof in the large [Azevedo and Budish, 2015] is as follows. In the limit as the market grows large, each student i regards both the prices p^* from Stage 2 and their initial budget β_i as exogenous to their own report. Moreover, in the limit as the market grows large, the probability that Stage 3 affects student i's allocation goes to zero, because market-clearing error goes to zero as a fraction of the economy as the economy grows large. Therefore, in a large market, the student can do no better than to act as a price taker and report her preferences truthfully. We also believe that Stage 3 is unlikely to affect students' incentives to report truthfully in small markets, because it seems extremely unlikely (if not impossible) that a student could predict which courses will have market-clearing error, and even with this information it is not obvious how, if at all, to strategically misreport.

over-subscription), so adding small amounts to budgets helps to compensate for this pricing error;

3. The 10% increase in a student's budget in Stage 3 is not worth the same as a 10% budget increase in the earlier stages because this budget increase can only be used to acquire courses with empty seats; and

4. Wharton administrators viewed a 10% increase as a tolerably small adjustment to budgets relative to the efficiency gains achieved.

The students selected early in the Stage 3 sequence have an advantage — they can receive access to desirable courses before other students. Hence Course Match sequences students in the following manner:

1. All second-year students come before any first-year student;

2. Within a year, students are ordered in increasing order of the tie-breaking budget subsidy, i.e., those with the smallest tie breaking budgets act first.

The second-year-first ordering reflects the cultural expectation at Wharton that second-year students should have preferential access to courses relative to first-year students. Within a year, however, the ordering has an intuitive fairness quality. Students with the smallest budget subsidies lose the tiebreaker in the previous stages of Course Match and are compensated by having a better position in the final stage of Course Match. Exploring the theoretical fairness and efficiency properties of our ordering scheme and alternatives is an open question for future work.

At the conclusion of Stage 3 we have our final course allocation. In this final allocation all maximum capacity constraints are satisfied and each student receives (at least) the best schedule they can afford given their initial budget and the final prices from Stage 2.

4.2.4 Alternative Approaches to Market-Clearing Error

We considered a number of alternative approaches to the problems of over-subscription and under-subscription resulting from Stage 1 that we ultimately rejected in favor of Stages 2 and 3 described above:

Drop Students from Over-Subscribed Courses

If after Stage 1 there are more students assigned to course j, $\sum_i x^*_{ij}$, than its capacity (either q_j or $\hat{q_j}$), then a sufficient number of students could simply be dropped from the course. This could be done with a lottery that randomly selected students to drop with equal likelihood. However, such an approach might drop students who have the strongest interest in the course, which reduces both efficiency and fairness. Alternatively, the drop process could be based on some observable data, for example, dropping the students who assigned the lowest utility to the course (or, to be more sophisticated, the students who would have the smallest percentage

reduction in utility if they were dropped from the course). However, such rules create an incentive for students to misreport their preferences because they treat each student's reported preferences like bids in an auction (i.e., the highest q_j bids win) and would introduce the problems associated with strategic bidding present in auction mechanisms, which Course Match seeks to eliminate. Furthermore, as a result of a random approach for dropping students, it would no longer be true that each student "receives the best schedule they can afford", which was considered to be a valuable claim for gaining student acceptance of the program.

Artificially Lower Target Capacities

The capacity assigned to course j, q_j, need not equal the maximum capacity, $\hat{q_j}$. One option is to set the target capacity, q_j, such that there is a sufficient buffer between it and the maximum capacity, i.e., $\hat{q_j} - q_j$ is "large enough." For instance, if a course is held in a room with 50 seats ($\hat{q_j} = 50$), then instead of choosing 50 as the target capacity, choose something like 40 seats ($q_j = 40$). If the Stage 1 solution has fewer than 10 over-subscribed seats, then the solution is actually feasible for the real problem.

There are several issues with this approach:

1. It is not possible *a priori* to know which courses need a buffer or how large that buffer should be. One could attempt to design an algorithm to determine the buffer quantities, but this is likely to be a challenging, non-linear, probabilistic search problem.
2. This approach is likely to leave empty seats in popular courses (i.e., courses with positive prices). Although such a solution could be implemented, it would not be well received by students.
3. There is no guarantee that this approach actually yields a feasible solution, because if the buffer is not chosen to be large enough, then it remains possible that $\sum_i x^*_{ij} > \hat{q_j}$ for some j.
4. Finally, artificially lowering the target capacities reduces the amount of slack in the allocation problem, which makes the Stage 1 search problem harder. To see why, first note that with enough slack the allocation problem is trivial: price all courses at zero and give each student their most preferred schedule. But as capacity becomes more restrictive, fewer courses have a zero price, which makes the effective dimension of the search problem larger. In our exploratory analysis, large reductions in course capacity made the search problem substantially harder, producing low-quality solutions.

Weighted Search

Given that over-subscription is costlier than under-subscription, a reasonable remedy is to penalize over-subscription by a greater amount than under-subscription (e.g., penalize over-subscription five times as much as under-subscription) in the Stage 1 price vector search.

We found that this approach resulted in substantially worse overall search performance, producing results that, much to our surprise, actually increased over-subscription. We conjecture that this counter-intuitive result is due to the fact that the error function plays two roles in the Stage 1 price vector search process. The first role is straightforward: is the candidate solution a good solution (i.e., low clearing error)? The second role is subtle: the clearing error score guides the search from a random initial starting point to a good solution (line 11 of Algorithm 1). When the weighting vector of the search is altered, the hope is to affect only the former role (solution quality), and produce solutions more skewed towards under-subscription. But we believe it also detracts from the second role, which is to guide the search process to a better solution. This may occur because the search problem is very challenging, with many local minima. As shown in Table 1, the path towards a good solution almost always involves taking steps through bad solutions. Informally, in challenging search problems, you need to enter a valley before you can climb a different hill. By weighting the search vector, the search is far less likely to enter intermediate solutions with over-subscription, and hence more likely to get stuck in a local minimum.

5 Computational Results

This section reports on the output of a production run of Course Match for the Spring 2014 semester. The run used seven Amazon EC2 `cc2.8xlarge` instances for 48 hours and was conducted in December 2013. In Section 5.1 we discuss the performance of the Stage 1 heuristic search, and in Section 5.2 we present results on solution quality across all three stages of Course Match. Section 5.3 discusses analogous results for non-production runs conducted to assess robustness; full details of these robustness runs are presented in Appendix A.

5.1 Search Results

Table 1 reports summary statistics of the Stage 1 price vector search process of the Spring 2014 Course Match production run.

Two observations from Table 1 are of particular interest. First, fewer than 5% (20/418) of the search starts followed a strict hill climb, meaning that the search improved until it found a local minima that was the best-found solution in that start. Put another way, in more than 95% of search paths, the best performing solution was found on a path that at some point moved to neighbors that temporarily decreased the solution quality. (Recall that Algorithm 1 allows the search to take up to five of these steps before terminating the search start.) This suggests that the search space is challenging and filled with local minima. The success that our Tabu search achieved in this setting is in line with past successful applications of Tabu search in challenging domains [Watson et al., 2003].

Second, even with the large amount of computational power behind it, only a vanishingly small fraction of the potential search space was actually explored. Consider a dramatically simpler

Table 1: Stage 1 price vector search summary statistics for the Spring 2014 Course Match production run

Compute servers	7
Hours	48
Number of courses	351
Number of search starts	418
Search paths performing a strict hill climb	20
Price vectors explored	2.0 million
Total MIPs solved	4.5 billion

Notes: A "strict hill climb" is a search path that does not improve its best-found solution by temporarily moving to a price vector with higher clearing error.

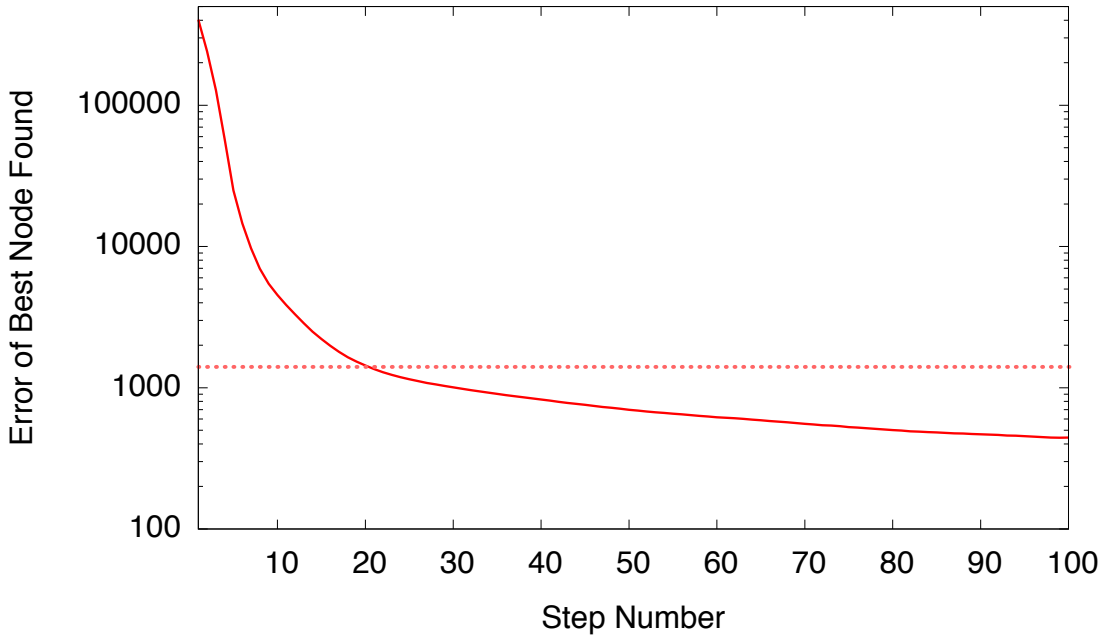
version of our search problem, in which courses can be assigned only one of two prices, 0 or 1. The number of price vectors explored in this run could exhaustively enumerate only a 21-course allocation problem, far smaller than the 351 courses under consideration. In fact, the actual problem is much larger. If we discretized the price space by unit prices up to 5,000 (a second year's base budget) for each course, an exhaustive enumeration of potential price vectors would have cardinality $5,000^{351} \approx 10^{1,300}$, of which we explore 2×10^6. For context, $10^{1,300}$ is much larger than the number of atoms in the universe, and 2×10^6 is much smaller than the number of atoms in a grain of sand. Our heuristic search procedure is powerful enough to produce practical solutions despite the size of the problem.

Figure 5 shows a plot of the average squared error of the best solution found on each search start, plotted against the number of steps taken in a search start (i.e., the number of times through the inner loop of Algorithm 1). The error falls quickly for about the first ten steps before the rate of decrease tapers off. The straight line on the plot represents the theoretical bound on clearing error. Since the number of courses in Spring 2014 was $M = 351$, and the largest requested bundle was $k = 8$ courses, there exists at least one solution to the allocation problem with total squared clearing error of at most $Mk/2 = 1,404$. The average search start crossed this bound after taking 21 steps.

Figure 6 provides a hint as to why the error falls so quickly in the early stages of the search. Recall that at a given price vector the search can step to a member of one of two neighborhoods: either one that adjusts only the price of a single course, or a gradient step that simultaneously adjusts the price of every course. The figure shows that early in the search gradient steps are selected almost exclusively, and then there is a steep drop-off in their frequency of selection after around 20 steps. Observe that this matches the drop in squared error, suggesting that squared error drops quickly when gradient steps are being taken but much slower when they are of limited use. However, the gradient steps are not ignored at later stages of the search. Our results indicate that gradient steps are selected about 15% of the time after 50 steps of a search start.

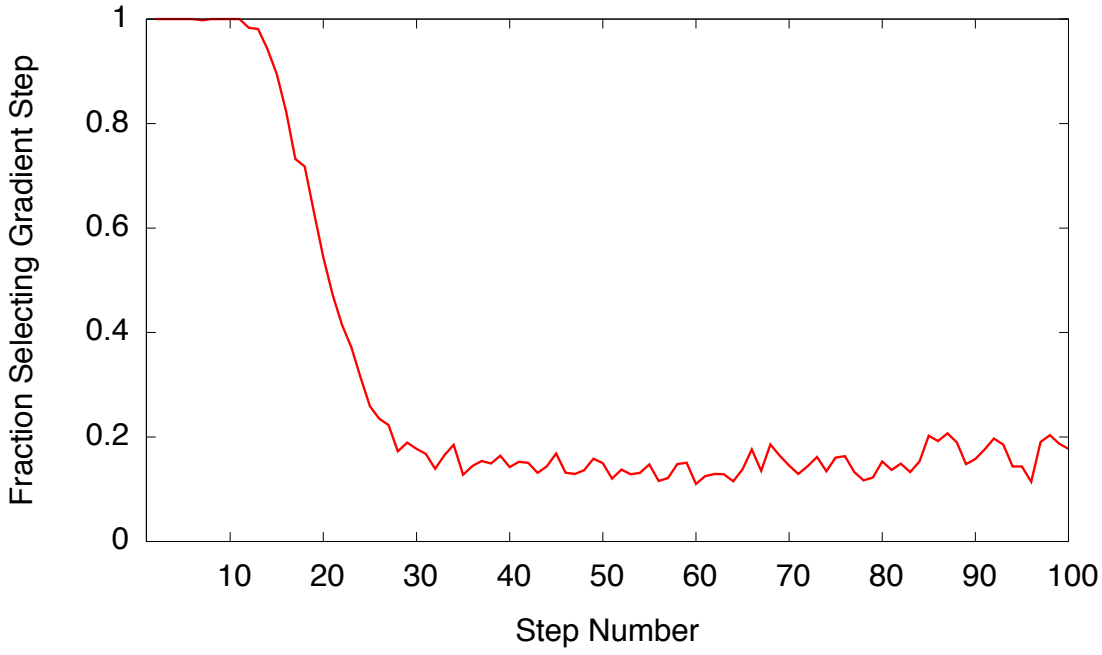
Recall from Algorithm 1 that the search restarts when a series of five sequential steps do not

Figure 5: The squared error of the best-found solution in each step of the Stage 1 price vector search (i.e, line 24 of Algorithm 1), averaged over all search starts, in the Spring 2014 Course Match production run.



Notes: The theoretical error bound is given by the dotted line. The y axis is log scaled.

Figure 6: The average fraction of search steps that selected a gradient neighbor (i.e, $\tilde{\mathbf{p}}$ is of the gradient neighbor type in line 20 of Algorithm 1) at a given number of steps in the Spring 2014 Course Match production run.



improve its squared error. Figure 7 shows the fraction of search starts that survive taking a given number of steps without restarting. It shows that the pace of restart begins to quicken around the 50th step.

Figure 7: The average fraction of search starts surviving to a given number of steps (i.e., we have not broken out of the intermediate loop begun on line 7 of Algorithm 1) in the Spring 2014 Course Match production run.

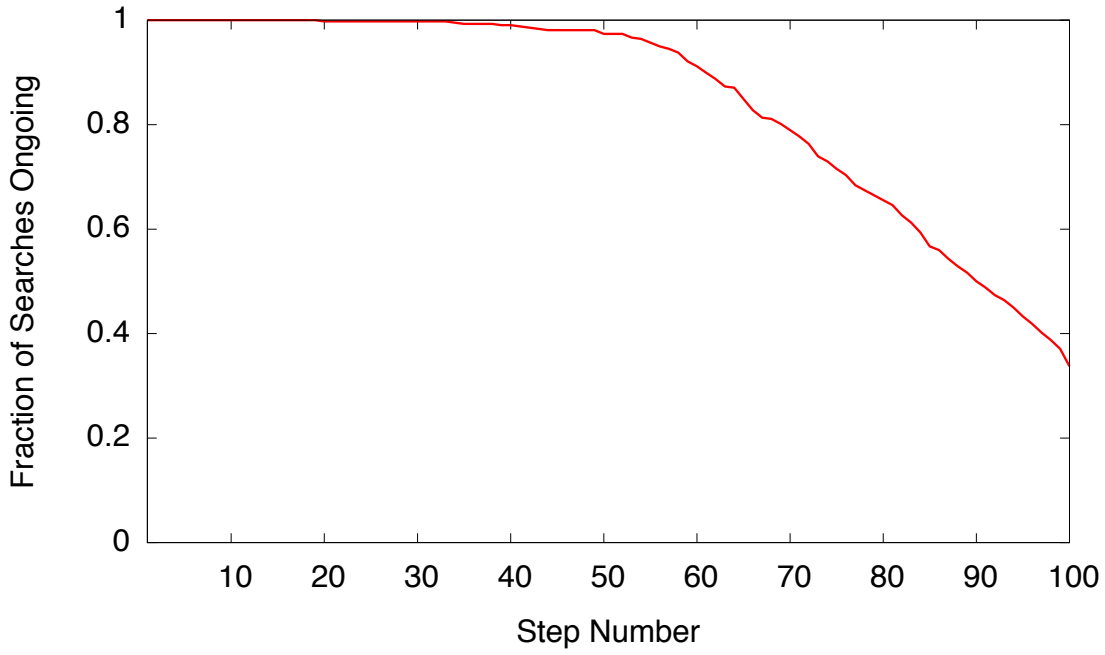


Table 2: Stagewise results from the Spring 2014 Course Match production run.

Compute	Stage 1: Price Vector Search α^2				Stage 2: Over-subscription Elimination α^2				Stage 3: Under-subscription Reduction α^2			
Server	+	-	Seats	Loss	+	-	Seats	Loss	+	-	Seats	Loss
1	51	40	32	0.19%	0	141	67	0.80%	0	31	17	0.07%
2	23	28	24	0.20%	0	205	97	1.26%	0	30	16	0.07%
3	75	35	25	0.31%	0	138	74	0.96%	0	21	17	0.07%
4	36	125	71	0.68%	0	228	106	1.00%	0	83	33	0.33%
5	59	48	36	0.28%	0	201	87	1.15%	0	52	24	0.09%
6	53	48	32	0.22%	0	186	86	0.87%	0	32	22	0.07%
7	47	56	42	0.54%	0	202	86	1.13%	0	17	13	0.02%

Notes: Each column refers to the solution *after* that stage completes. Squared clearing error, α^2, is provided for over-subscription (+) and under-subscription (-) separately, with over-subscription in all cases measured relative to maximum capacity, $\hat{q}_j$, and under-subscription measured relative to target capacity, q_j. "Loss" is the percent deadweight loss from under-subscribed seats in positive-price courses (i.e., the total value of those seats, based on their price, relative to the total value of all seats).

5.2 Stage Comparison

Table 2 presents measures of solution quality for each of the seven compute servers for each stage of the Course Match algorithm. It reports squared market clearing error α^2, broken separately into error arising from over-subcription and error arising from under-subscription; the total error in seats; and a measure of deadweight loss, defined as the value of the under-subscribed seats divided by the total value of all seats. As each compute server operates independently, each compute server yields a distinct solution after each stage. There are several interesting results from Table 2.

- The compute server with the best solution changes across stages and metrics. Server 2 has the lowest squared error after the Stage 1 price vector search; Server 1 after the Stage 2 over-subscription elimination; and Server 7 had the lowest deadweight loss and fewest under-subscribed seats following the conclusion of the Stage 3 under-subscription reduction. This result strongly suggests the value of running the entire search process in parallel across many different compute servers.

- The amount of squared error increased on each server as a result of the Stage 2 over-subscription elimination process. Because squared error is symmetric, in general the solutions at the end of the Stage 1 price vector search will have total error approximately equal between over-subscription and under-subscription. Consequently, we should expect squared error to increase when we move to a solution without over-subscription.

- The number of empty seats does not strictly correspond to the total deadweight loss. In some

solutions the unallocated seats are in low price courses while in others they are in high price courses. In the Server 4 solution each seat in the average under-subscribed course is worth 0.01% of the economy, while in the Server 7 solution under-subscribed seats are worth about a sixth of that.

- In Stage 3 under-subscription reduction there is on average a 77% reduction in the number of positive-price under-subscribed seats and a 90% reduction in deadweight loss. This is a significant increase in efficiency from a relatively small adjustment in budgets (10%).

- Since the Stage 1 price vector search utilizes an enhanced version of the prior state-of-the-art algorithm [Othman et al., 2010], comparison of solution quality after Stage 1 to solution quality after Stage 3 gives a sense of the improvement Course Match makes to the prior state-of-the-art. Since Othman et al. [2010] uses a single-server architecture and Course Match uses a multi-server architecture, the most relevant comparison is between the average performance of the Stage 1 search and the best performance of the Stage 3 search. Course Match improved total squared error from 103 to 17 (84% reduction), improved error in seats from 37 to 13 (65% reduction), and improved deadweight loss from 0.54% to 0.02% (94% reduction). If instead we used the best Stage 1 performance as the benchmark (essentially moving Othman et al. [2010] to our independent architecture), the improvements would be 67% for total squared error, 46% for seats and 89% for deadweight loss. Perhaps more importantly, the prior state-of-the-art yielded a solution that was infeasible due to violation of capacity constraints, whereas Course Match yields a feasible solution. A disadvantage relative to the prior state-of-the-art is the extra budget inequality needed in Stage 3 to achieve these results.

Of the seven solutions, the Server 7 solution was advanced for implementation by the Wharton program administrators because it had the lowest deadweight loss. There were then last-minute course modifications by Wharton administrators which necessitated rerunning the search (with less computational time); the best solution in this re-run had deadweight loss of 0.07%. In future production runs, Wharton plans to continue to select the solution with the lowest deadweight loss after Stage 3.

5.3 Robustness

In Appendix A, as a robustness check, we perform the analyses of Sections 5.1-5.2 for the Fall 2013 semester. The detailed computational performance data had not been maintained from the original production run so we re-ran Course Match on the Fall 2013 preference and capacity data in early 2016. To get an apples-to-apples comparison, we also re-ran Course Match on the Spring 2014 data in early 2016.

At a high level, results from Fall 2013 are substantially similar to results from Spring 2014. However, a few interesting differences and observations emerged:

- While the pattern depicted in Figure 6 obtains for Fall 2013 as well, the Fall 2013 search consistently relied more heavily on the individual adjustment neighbors than on the gradient

steps. The individual search starts went on for longer without getting stuck but also improved more slowly (see the Appendix versions of Figures 7 and 5, respectively).

- The overall solution quality in Fall 2013 was worse than that for Spring 2014, with the difference becoming particularly evident after over-subscription is eliminated in Stage 2. See the Appendix version of Table 2. We hypothesize that this difference may be because included in the Fall is a large set of courses of which students are required to take exactly one from the set, where each course in the set has a small target capacity (10 or fewer seats) and student preferences across the set are primarily driven only by when the course is offered (rather than topic or instructor). Consequently, adding or removing a student from one of these courses has a relatively large impact on enrollment, which makes fine-tuned pricing adjustments challenging. Nevertheless, the overall amount of error for Fall 2013 is quite small in absolute terms: 27 seats, and deadweight loss of 0.16%.

- To our surprise, the performance of Course Match on the Spring 2014 data was noticeably better in our early 2016 analysis versus the December 2013 production run reported in the body of the paper. The number of price vectors explored and MIPs solved increased by about 60%, from 2.0 million and 4.5 billion to 3.2 million and 7.1 billion, respectively. The final solution had error of just 3 seats and deadweight loss of less than 0.01%. This difference is attributable to an improvement in the computational performance of the Amazon cloud. This appears to be the first documentation of Amazon increasing the performance of identical compute instances over time.

In addition to these robustness results reported in Appendix A we note as well that Course Match has been running successfully at Wharton since Fall 2013 (6 semesters as of the present writing). The computational performance and solution quality reported in the present paper are representative of the performance and quality in subsequent semesters.

6 Economic Results

In this section we summarize the economic performance results from the Fall 2013 and Spring 2014 semester runs of Course Match at Wharton. We begin by describing the quantitative properties of the allocations themselves and then provide survey data demonstrating that students were very satisfied with those allocations.

6.1 Efficiency

Table 3 shows quantitative summary statistics from the Fall 2013 and Spring 2014 runs. The two semesters had similar inputs and outputs. One salient difference between the two semesters was that Spring 2014 had many more courses with maximum capacities above target capacities. The quality of the final allocation in Spring 2014 also appears to have been higher; its deadweight loss was about one-third of the deadweight loss of the Fall term, although both semesters had very low

deadweight loss in absolute terms, less than one fifth of one percent of the total economy in each semester.

Table 3: Quantitative summary statistics from the Fall 2013 and Spring 2014 allocations

	Fall 2013	Spring 2014
Students	1650	1700
Courses	285	344
Courses with maximum above target capacity, i.e., $q_j < \hat{q}_j$	78	262
Total capacity overhead $(\sum_j (\hat{q}_j - q_j) / \sum_j q_j)$	0.8%	2.1%
Number of courses with demand above target capacity	13	49
Fraction of total capacity allocated	71%	74%
Number of courses with a positive price	154	199
Number of under-subscribed positive price courses	15	18
Deadweight loss as percent of economy	.19%	.07%
Highest course price as fraction of average budget	1.31	0.88

Figures 8a and 8b show the relationship between course capacities and prices in the Fall 2013 and Spring 2014 allocations. Each circle represents a course. The color of a circle is determined by the allocation of the course relative to its target capacity. Red circles are courses with low allocation relative to target capacity, white circles are courses that exactly match target capacity, and blue circles are courses that exceed target capacity (but do not exceed maximum capacity). The saturation of the colors reflects the absolute error relative to target capacities: a class with no students would be dark red and a class that fills its maximum capacity would be dark blue. In each semester there is a weak relationship between course price and capacity. Furthermore, while most (but not all) of the courses with allocations below target capacity have a zero or low price, courses with allocations above target capacity (blue) occur at all prices, and are not concentrated just in the high priced courses.

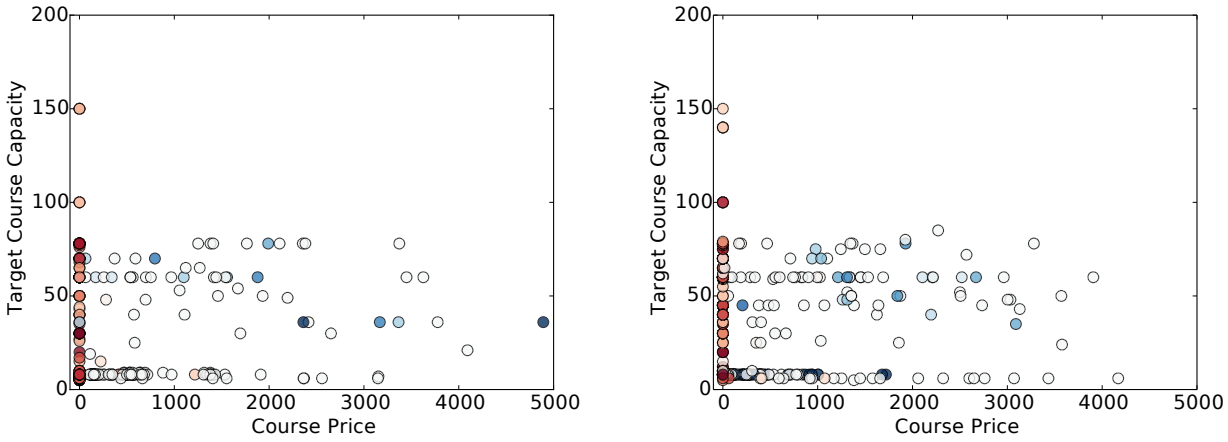
Figures 9a and 9b plot the number of seats over or under target capacity for courses with positive prices and under-subscription (red circles), or allocations above their target capacities (blue circles). There is a weak relationship between the absolute clearing error and course price. Absolute clearing error is also somewhat limited — never more than 5 seats. Furthermore, as one would hope, over-subscription (based on target capacity) is more common relative to under-subscription with high-price courses and less so with low-price courses.

Table 4 shows how Stage 2 (over-subscription elimination) and Stage 3 (under-subscription reduction) affect the solution. In Fall 2013, about 34% of course prices are changed to eliminate over-subscription, while in Spring 2014 about 28% of course prices are changed. As a result of these price changes, about 23% of students in the Fall and 19% of students in the Spring change their course allocations to eliminate over-subscription. Finally, with under-subscription reduction only about 13% of students in the Fall and 9% of students in the Spring change their allocations.

Figures 10a and 10b show the budget expenditure by students over the three stages for the Fall

Figure 8: Course Capacities and Prices

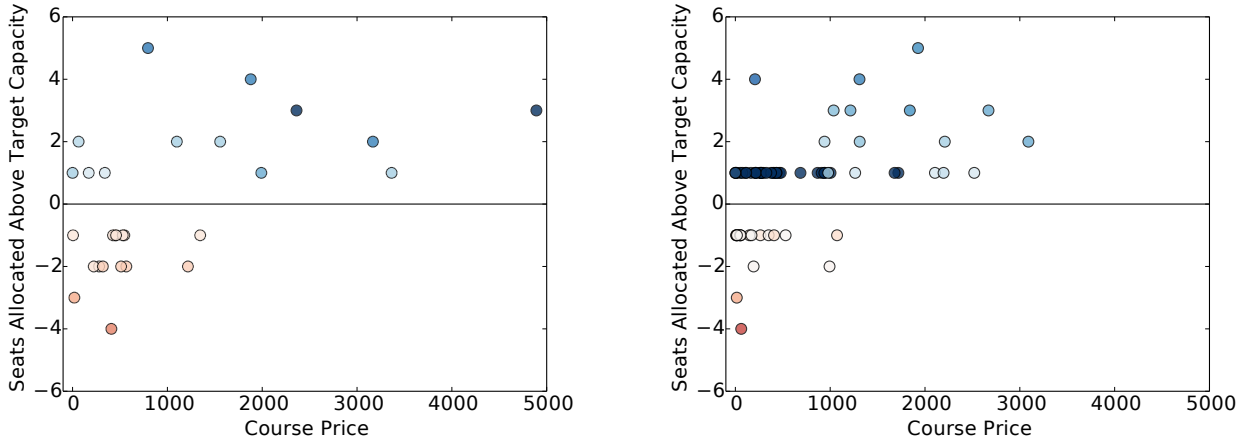
(a) Fall 2013 **(b)** Spring 2014



Notes: Each circle represents a course. The color of a circle is determined by the allocation of the course relative to its target capacity. Red circles are courses with low allocation relative to target capacity, white circles are courses that exactly match target capacity, and blue circles are courses that exceed target capacity (but do not exceed maximum capacity). The saturation of the colors reflects the absolute error relative to target capacities: a class with no students would be dark red and a class that fills its maximum capacity would be dark blue.

Figure 9: Course Market-Clearing Errors and Prices

(a) Fall 2013 **(b)** Spring 2014



Notes: Courses in Fall 2013 and Spring 2014 that either exceeded their target capacities (i.e., they made use of additional max capacity), or had positive price but were under-subscribed (i.e., they contributed to deadweight loss). The saturation of the colors reflects allocations relative to target capacities: a class with no students would be dark red and a class that fills its maximum capacity would be dark blue.

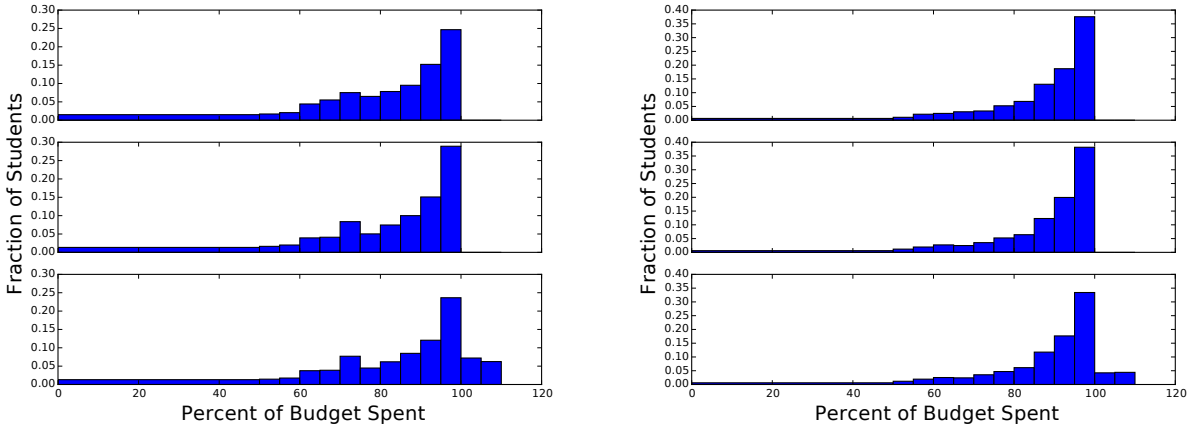
Table 4: The degree of effect of over-subscription elimination and automated aftermarket on courses and students

	Fall 2013	Spring 2014
Courses	285	344
Courses changing price in over-subscription elimination	98	97
Students	1650	1700
Students changing allocation in over-subscription elimination	381	327
Students changing allocation in under-subscription reduction	222	147

Figure 10: Percent of Budget Spent by Stage

(a) Fall 2013, Stages 1-3

(b) Spring 2014, Stages 1-3



Notes: The percentage of initial budget spent by students after (top panel) Stage 1 Price vector search, (middle panel) Stage 2 Over-subscription elimination, and (bottom panel) Stage 3 Under-subscription reduction. For clarity, students spending $< 50\%$ of their budget are grouped into a single bucket.

2013 and Spring 2014 semesters. Each figure consists of three subplots. The top subplot buckets students by the amount of their initial budget spent after Stage 1 (price vector search). The middle subplot shows these buckets after Stage 2 (over-subscription elimination), and the bottom subplot shows these buckets after the completion of Stage 3 (under-subscription reduction). In Stages 1 and 2, students cannot spend more than 100% of their initial budget and while most students spend most of their budget (say 90% or more), there remains a large number of students who spend a smaller portion of their budget. In Stage 3 students are able to spend an additional 10% of their initial budget on under-subscribed courses. Nevertheless, fewer than 15% of students actually spend more than their budget, which suggests that possible equity concerns associated with Stage 3 are not significant.

6.2 Fairness

One measure of fairness is envy — when a student prefers the allocation received by another student over her own. We already know that there should be some envy in the Course Match allocations because Course Match divides students into groups with significantly different budgets (e.g., all second-year students get larger budgets than all first-year students) and even within the same group their can be envy due to variations in the tie breaking subsidy (though, as mentioned earlier, that envy is bounded by a single course). As it is mathematically impossible to design a mechanism that is completely free of envy, it is not clear how much envy is tolerable and how much is too much. To provide a benchmark, it would be ideal to compare the envy in the Course Match allocation to the envy produced by a competing mechanism, such as the auction Wharton previously used. However, because bids are unlikely to represent true preferences in the Wharton Auction, it is not possible to use previous Auction data to determine levels of envy. (However, see Budish and Kessler [2015] for experimental evidence comparing envy between these two mechanisms.)

As we cannot use envy to measure fairness, we provide two alternative, but related, measures of fairness that enable us to compare Course Match to Wharton's previous auction mechanism. The first focuses on how evenly seats in the most expensive courses are distributed among the students and the second, called the Gini index, measures how evenly the "wealth" in this economy is distributed, where the value of a seat equals its price.

Table 5 shows the percentage of second-year students with S courses among the most expensive N courses. (We focus on second-year students because first-year students vary significantly in the number of courses they desire due to course requirements and waivers.) Two values of N are reported, 20 and 40, because the number of seats among the top 20 courses is approximately equal to the number of second-year students and the number of seats among the top 40 courses is approximately equal to the total number of MBA students. Because the Spring and Fall semesters differ in the types of courses offered and student needs, each Course Match semester is compared against the same semester in the previous year that used the Wharton Auction. If all students want a seat among the top N courses (which is unlikely to be true, but most students probably have some preference for at least one of these courses) then an equitable solution would concentrate the distribution in the $S = 1$ and $S = 2$ columns. In all cases, Course Match concentrates substantially more of its distribution in those desirable columns than the Auction. For example, comparing the Auction in Fall 2012 to Course Match in Fall 2013, 63% of students received 1 or 2 courses among the top 20 with the Auction, while 86% did with Course Match. Consequently, with the Auction, 32% of students were excluded from the top 20 courses ($S = 0$) whereas only 13% were absent from the top 20 courses with Course Match. And while no student with Course Match was allocated more than 3 seats among the top 20 courses, with the Auction, a lucky 5% of students received 3 or more seats in that set. The same pattern emerges when we consider the 40 highest priced courses and the Spring semesters. In sum, with Course Match, relative to the Auction, fewer students are excluded from the most expensive courses and very few (if any) students are able to acquire many seats among those courses.

Table 5: Percentages of second-year students with S courses among the N highest priced courses

Mechanism	Semester	N	$S=0$	$S=1$	$S=2$	$S=3+$
Auction	Fall 2012	20	32%	43%	20%	5%
Course Match	Fall 2013	20	13%	71%	15%	0%
Auction	Fall 2012	40	14%	34%	31%	20%
Course Match	Fall 2013	40	4%	44%	50%	2%
Auction	Spring 2013	20	54%	38%	8%	0%
Course Match	Spring 2014	20	32%	65%	4%	0%
Auction	Spring 2013	40	20%	42%	29%	9%
Course Match	Spring 2014	40	6%	52%	42%	0%

Instead of focusing on the distribution of seats in the most expensive courses, it is also possible to measure the distribution of seats overall. If the value of a seat is taken to be its price, then the sum of the prices of the seats in a student's schedule can be taken as a measure of the wealth the student has earned from the allocation. Fairness suggests that wealth should be evenly distributed. The Gini coefficient measures the inequality of a distribution and is typically used for measuring income inequality. The Gini coefficient ranges between 0 and 1: if every person has identical wealth, the Gini coefficient is 0; if a single person controls all of the wealth, the Gini coefficient is 1. Hence, wealth is more concentrated among a few as the Gini coefficient increases.

The interpretation of the Gini coefficient in the context of income inequality is clear — because it is reasonable to assume that everyone prefers more income to less, an increase in the Gini coefficient implies more of the valuable resource (income) is assigned to fewer people, which means less equality. With course allocation the interpretation of the Gini coefficient is not as straightforward. It is possible that a student receives their most preferred schedule even though the schedule has zero total cost — it just so happens that the student values courses that are not capacity constrained. Thus, the Gini coefficient probably overstates the degree of income inequality. Nevertheless, popular courses are likely to have higher prices, and so the total cost of a student's schedule is a reasonable proxy of the value the economy places on her schedule.

Table 6 reports the Gini coefficients for the Fall and Spring semesters, one with the prices generated by the Auction in the last year it was implemented, and the other with the prices generated by Course Match in its first year of implementation. The coefficients are calculated for three different student groupings: only first-year students, only second-year students, and all students. Recall from Section 4.1 that students are granted roughly equal budgets within a year, but unequal budgets between years (and that was true with the Auction as well). This systematic unfairness is considered natural in the course allocation setting at business schools. Consequently, intra-year Gini coefficients are a better measure of fairness in this setting than the Gini coefficients of allocations including both years. By design (and therefore, as expected), the Gini coefficient for the whole student body is higher than the intra-year Gini coefficients.

As is evident, Course Match in all pairwise comparisons with the Auction reduces the Gini

Table 6: Gini coefficients of student allocations. Fall 2012 allocations are taken as round 5 of the Auction

Mechanism	Semester	First-Year Students	Second-Year Students	All Students
Auction	Fall 2012	0.33	0.36	0.54
Course Match	Fall 2013	0.13	0.22	0.32
Auction	Spring 2013	0.25	0.39	0.34
Course Match	Spring 2014	0.10	0.12	0.15

coefficient, meaning that Course Match produces an allocation that more evenly distributes wealth among the students. To calibrate these scores somewhat (and with the understanding that the Gini coefficient with course allocation likely overstates income inequality), the Auction's Fall 2012 intra-year Gini coefficients are roughly in line with the United Kingdom's post-transfer income distribution (0.34), while both the Fall 2013 and Spring 2014 Course Match allocations have intra-year Gini values lower or much lower than the country with the lowest post-transfer income distribution, Denmark (0.24). The Spring 2014 schedule, in particular, has intra-year Gini coefficients that are very low, around 0.1.

6.3 Student Perception

Although Course Match has desirable results in terms of various efficiency and fairness measures, the ultimate metric of its success is student satisfaction. Prior to 2013, Wharton measured satisfaction with the Auction course registration system with a single question on an annual stakeholder survey given to MBA students. 2013 was the last year in which the Auction was implemented for course allocation. In anticipation of the Auction's retirement, two additional questions were added to the survey in 2013 — one directed towards satisfaction with the course schedule a student received and the other on a student's impression of the fairness of the Auction. The same questions were asked in 2014 with "Course Match" replacing "the auction" in the wording of the question.

Table 7 reports the percentage of students responding with one of the top two scores (6 or 7). With respect to "effectiveness," student satisfaction had been decreasing over time, with the lowest score occurring in the last year of the Auction, but then this measure improved considerably in 2014, the year Course Match was implemented. Course Match performed very well on the other two measures as well — "satisfied with my schedule" increased from 45% to 64% and "fairness" experienced an even larger increase, from 28% to 65%. These dramatic improvements also provide indirect evidence suggesting that students were able to report their preferences to the Course Match system successfully — if they were not successful in doing so, either because the language was too limiting or because they had difficulty "speaking" it, then it is unlikely that satisfaction scores would be high. In sum, the student survey results provide strong evidence that Course Match is effective at providing efficient solutions and increasing perceptions of fairness.

Table 7: Percent of 6 or 7 responses on three questions related to course allocation at Wharton

	2010	2011	2012	2013	2014
Effectiveness of the course registration system (1 = Poor, 7 = Excellent)	43%	43%	34%	24%	53%
I was satisfied with my schedule from {the auction, Course Match} (1 = very unsatisfied, 7 = very satisfied)	-	-	-	45%	64%
{The auction, Course Match} allows for a fair allocation of classes (1 = strongly disagree, 7 = strongly agree)	-	-	-	28%	65%

Notes: Year refers to the end of the academic year in the spring semester, e.g., 2014 is the academic year covering Fall 2013 and Spring 2014. Only the "effectiveness" question was asked in 2012 and earlier. The 2013 results apply to the Auction, the last year of its use. The 2014 results apply to Course Match.

7 Conclusion

Course Match is a large-scale implementation of the Approximate Competitive Equilibrium from Equal Incomes (A-CEEI) mechanism for course allocation that is capable of producing implementable (i.e., feasible) solutions within a sufficiently small lead time by cost effectively harnessing the power of cloud computing. The resulting allocation was attractive on quantitative measures of economic efficiency and fairness, such as deadweight loss and the equality of the distribution of popular courses. Perhaps most importantly, relative to the previously used auction mechanism, Course Match substantially increased student perceptions of effectiveness, satisfaction, and fairness.

A critical feature for the success of Course Match is its "strategy-proof" property — a student's best strategy is to report her true preferences no matter what preferences other students report or what capacities are assigned to each course. This greatly simplifies the student's reporting task because the student need not form beliefs about how other students will "play" or what clearing prices might be for courses. In contrast, the Wharton Auction (as well as all other course-allocation mechanisms implemented in practice) was not strategy-proof. For example, if a student desires a course but believes that it will have a zero clearing price, then the student should rationally submit a low bid and save tokens to bid on other courses. However, the student may make a mistake and not receive the course she desires if the clearing price turns out to be higher than expected. This bidding mistake is not trivial and it could even lead a student with ample tokens to receive zero courses. Such errors do not happen with Course Match because Course Match effectively bids on behalf of students after all of the clearing prices have been revealed. Hence, Course Match never

"bids" more than the clearing price (which would waste budget tokens) nor "bids" less than the clearing price for a desired course (thereby losing the seat). Furthermore, although Course Match is able to solve the MIP that yields a student's best schedule given the reported preferences and price vector, it is highly unlikely a student would be able to identify the best schedule with the same information consistently, i.e., computers are better than humans at solving MIPs. In sum, Course Match performs all of the tasks that are best performed by a computer (e.g., finding a price vector and assigning seats to students given that price vector) while leaving the students the one task the computer cannot do, i.e., reporting their own preferences. Consequently, a substantial amount of human computation effort (e.g., forming beliefs, choosing bids) is eliminated and replaced with cloud-based computing power.

The cliché "garbage in, garbage out" applies with Course Match — while the Course Match mechanism has many desirable theoretical properties, if the preference language given to students is not sufficiently rich (i.e., it does not allow students to express critical preferences) or if students are not able to "speak" this language (i.e., they cannot use the language to correctly report their preferences), then Course Match may not yield desirable results. We are not able to provide direct evidence of the quality of the Course Match preference reporting language and user interface, but the high overall student satisfaction scores provide indirect evidence that the Course Match language is sufficiently rich and easy to use. See also Budish and Kessler [2015] for laboratory evidence on the efficacy of the reporting language.

We are able to document that Course Match is a superior mechanism for taking a given set of courses and allocating them to students. Another approach to improve student satisfaction is to change the set of course offerings, i.e., which courses are offered, which classrooms they are offered in and when they are offered. Given that students report their true preferences to Course Match, it is possible to observe the demand for each course that would occur if there were no capacity constraints. This enables the school administration to distinguish between two courses, each with 60 seats and 60 students enrolled, but one that would have 200 students and the other that would have 20 students in the absence of capacity constraints. The course with the higher demand is far more popular, even though it has the same enrollment as the less-popular class, which must have full enrollment only because some students were not able to get into their more preferred courses (i.e., 40 of the 60 students enrolled would not have included the course in their most preferred schedule). It is plausible that the effective use of the preference data available through Course Match could lead a school to make smarter decisions about its course offerings, which could lead to further substantial gains in student satisfaction.

We do not claim that the Course Match computational architecture is "optimal." Indeed, an important question left for future research is whether there are better approaches to finding approximate market-clearing prices than that described here. We do show, however, that the Course Match computational architecture works at Wharton. To borrow a common analogy (e.g., Roth [2002]), ours is an exercise of engineering rather than physics. Additionally, some back-of-the-envelope calculations suggest that even the largest course allocation problems are within reach

of the Course Match architecture. For example, Ohio State has about 60,000 students at its main campus, as compared with 1,700 students at Wharton, and thousands of courses each semester as compared with up to 350 courses at Wharton. However, whether a student is at a large school or a small school, they are likely to report preferences for around 15-30 courses each semester, given that they are likely interested in taking about 4-5 courses each semester. Because a MIP solver can immediately discard all courses that a student has no interest in taking, the difficulty of solving a student's MIP is likely to be no more difficult at Ohio State than at Wharton. There are $60,000/1,700 \approx 35$ times as many students at Ohio State than at Wharton, however, and thus 35 times as many MIPs to solve at each price vector. It is unclear whether the price search problem is harder or easier at Ohio State — it could be harder because there are considerably more courses, but could be easier because there is likely to be less overlap in demand for those courses (e.g., courses at the dental school are effectively a separate market from courses at the liberal arts school), and maybe because fewer courses are likely to be capacity constrained. If we assume that the search problem at Ohio State is the same difficulty as the search problem at Wharton, and incorporate the 60% speedup of Amazon Web Services we observed between late 2013 and early 2016, we would need on the order of $35/1.6 \approx 20$ times as much computational time to solve the Ohio State problem as was used for the Spring 2014 Wharton production run. This is a lot of computational power but is certainly feasible given the current scale of Amazon Web Services. Thus, even the largest realistic problems are within reach of the Course Match architecture. Furthermore, other applications of the combinatorial allocation problem, such as workforce scheduling, are likely to be much smaller, and therefore easily handled by the Course Match architecture.

References

T. Abbott, D. Kane, and P. Valiant. On the complexity of two-player win-lose games. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 113–122, 2005.

A. Abdulkadiroglu and T. Sönmez. School choice: A mechanism design approach. *The American Economic Review*, 93(3):729–747, 2003.

E. Azevedo and E. Budish. Strategy-proofness in the large. *Working Paper*, 2015.

C. Borgs, J. Chayes, N. Immorlica, A. T. Kalai, V. Mirrokni, and C. Papadimitriou. The myth of the folk theorem. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 365–372, New York, NY, 2008. ACM.

E. Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061 – 1103, 2011.

E. Budish and E. Cantillon. The multi-unit assignment problem: theory and evidence from course allocation at harvard. *American Economic Review*, 102(5):2237–2271, 2012.

E. Budish and J. Kessler. Experiments as a bridge from market design theory to market design practice: Changing the course allocation mechanism at wharton. *Working Paper*, 2015.

X. Chen and S.-H. Teng. Spending is not easier than trading: on the computational equivalence of fisher and arrow-debreu equilibria. *ISAAC '09 Proceedings of the 20th International Symposium on Algorithms and Computation*, pages 647–656, 2009.

X. Chen and S.-H. Teng. A complexity view of markets with social influence. In *ICS*, pages 141–154, 2011.

X. Chen, D. Paparas, and M. Yannakakis. The complexity of non-monotone markets. In *STOC*, pages 181–190, 2013.

B. Codenotti, A. Saberi, K. Varadarajan, and Y. Ye. Leontief economies encode nonzero sum two-player games. In *Electronic Colloquium in Computational Complexity*, pages 5–55, 2006.

C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a nash equilibrium. *Communications of the ACM*, 52(2):89–97, 2009.

L.-S. Huang and S.-H. Teng. On the approximation and smoothed complexity of leontief market equilibria. In *Lecture Notes in Computer Science*, pages 96–107, 2007.

S. Kintali, L. J. Poplawski, R. Rajaraman, R. Sundaram, and S.-H. Teng. Reducibility among fractional stability problems. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 283–292, Washington, D.C., 2009. IEEE Computer Society.

A. Krishna and M. U. Ünver. Improving the efficiency of course bidding at business schools: Field and laboratory studies. *Marketing Science*, 27(2):262–282, 2008.

A. Othman, E. Budish, and T. Sandholm. Finding Approximate Competitive Equilibria: Efficient and Fair Course Allocation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 873–880, Toronto, Canada, 2010.

A. Othman, C. Papadimitriou, and A. Rubinstein. The complexity of fairness through equilibrium. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, EC '14, pages 209–226, New York, NY, 2014. ACM.

D. Pálvölgyi. 2d-tucker is ppad-complete. In *Proceedings of the 5th International Workshop on Internet and Network Economics*, WINE '09, pages 569–574, Berlin, Heidelberg, 2009. Springer-Verlag.

C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal Computer and System Sciences*, 48(3):498–532, 1994.

A. E. Roth. Economist as engineer: Game theory, experimentation, and computation as tools for design economics. *Econometrica*, 70(4):1341–1378, 2002.

A. E. Roth and E. Peranson. The redesign of the matching market for american physicians: Some engineering aspects of economic design. *The American Economic Review*, 89(4):748–780, 1999.

A. E. Roth, T. Sönmez, and M. U. Ünver. Kidney exchange. *The Quarterly Journal of Economics*, 119(2):457–488, 2004.

A. E. Roth, T. Sönmez, and M. U. Ünver. Pairwise kidney exchange. *Journal of Economic Theory*, 125(2):151–188, 2005.

A. Rubinstein. Inapproximability of nash equilibrium. *Working Paper*, 2014.

T. Sandholm. Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions. *AI Magazine*, 28(3):45–58, 2007.

T. Sandholm and C. Boutilier. Preference elicitation in combinatorial auctions. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, pages 233–263. MIT Press, 2006. Chapter 10.

T. Sönmez and M. U. Ünver. Course budding at business schools. *International Economic Review*, 51(1):99–123, 2010.

X. H. Sun and D. T. Rover. Scalability of parallel algorithm-machine combinations. *IEEE Transactions Parallel Distributed Systems*, 5(6):599–613, 1994.

V. V. Vazirani and M. Yannakakis. Market equilibrium under separable, piecewise-linear, concave utilities. *Journal of the ACM*, 58(3):10, 2011.

J.-P. Watson, J. Beck, A. E. Howe, and L. Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143(2):189–217, 2003.

A Robustness Appendix

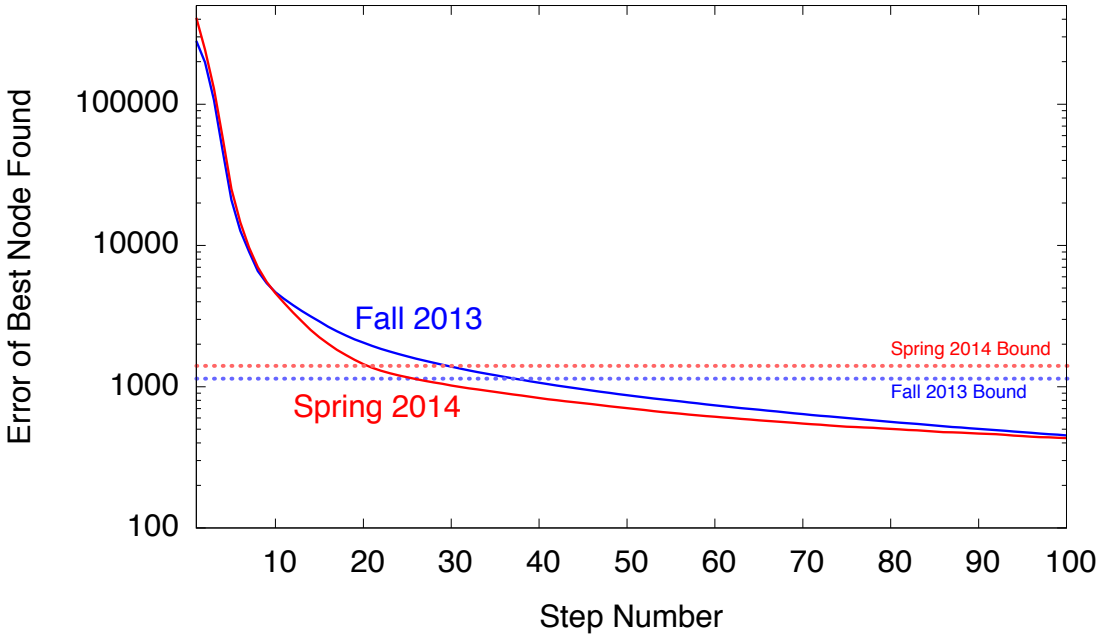
In this Appendix we repeat the analyses of Sections 5.1-5.2 on Fall 2013 preference and capacity data. These analyses were performed in January 2016 using seven Amazon EC2 `cc2.8xlarge` instances for 48 hours. Since the computational performance of the Amazon cloud computing environment has meaningfully improved since the time of the data reported in the main text, we also repeat the analyses of the Spring 2014 preference and capacity data to provide an apples-to-apples comparison between the two semesters. These analyses were performed in February 2016. Discussion of these results is in the body of the text in Section 5.3.

Table A-1: Stage 1 price vector search summary statistics for non-production runs using data from the Fall 2013 and Spring 2014 semesters

	Fall 2013	Spring 2014
Computer servers	7	7
Hours	48	48
Number of courses	285	351
Number of search starts	562	659
Search paths performing a strict hill climb	14	27
Price vectors explored	4.0 million	3.2 million
Total MIPs solved	8.1 billion	7.1 billion

Notes: A "strict hill climb" is a search path that does not improve its best-found solution by temporarily moving to a price vector with higher clearing error.

Figure A-5: The squared error of the best-found solution in each step of the Stage 1 price vector search (i.e, line 24 of Algorithm 1), averaged over all search starts, in the non-production runs using data from the Fall 2013 and Spring 2014 semesters.



Notes: The theoretical error bound for each semester is given by the dotted line of corresponding color. The y axis is log scaled.

Figure A-6: The average fraction of search steps that selected a gradient neighbor (i.e, $\tilde{\mathbf{p}}$ is of the gradient neighbor type in line 20 of Algorithm 1) at a given number of steps in the non-production runs using data from the Fall 2013 and Spring 2014 semesters.

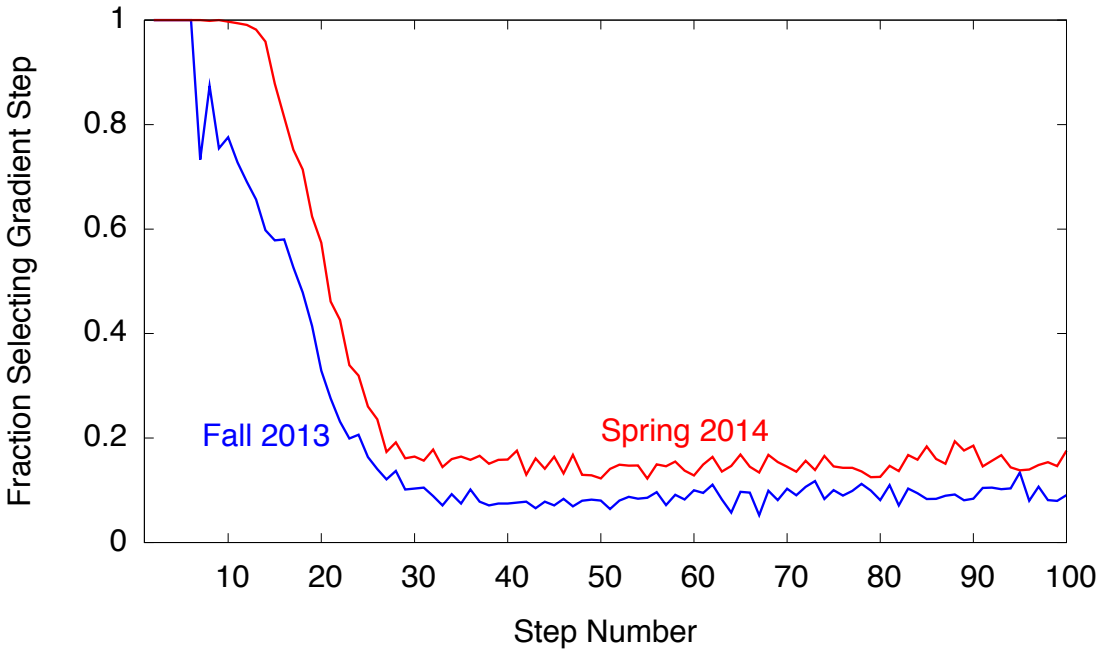


Figure A-7: The average fraction of search starts surviving to a given number of steps (i.e., we have not broken out of the intermediate loop begun on line 7 of Algorithm 1) in the non-production runs using data from the Fall 2013 and Spring 2014 semesters.

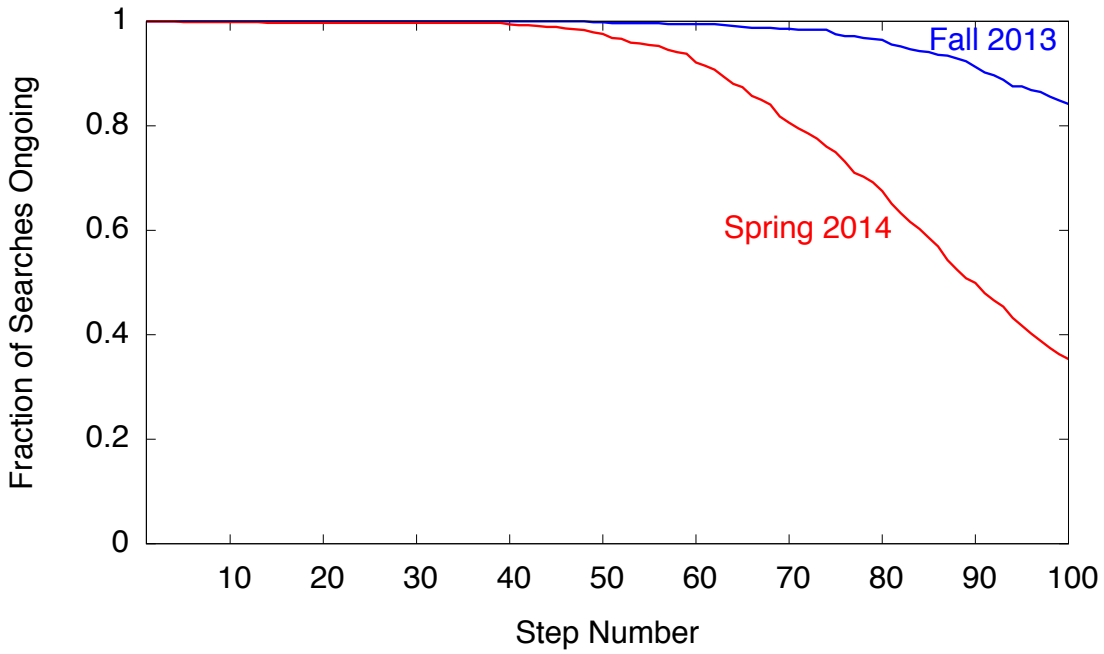


Table A-2: Stagewise results from the non-production run using data from the Fall 2013 and Spring 2014 semesters

A-2 (Fall)

Compute	Stage 1: Price Vector Search α^2				Stage 2: Over-subscription Elimination α^2				Stage 3: Under-subscription Reduction α^2			
Server	+	-	Seats	Loss	+	-	Seats	Loss	+	-	Seats	Loss
1	62	29	23	0.42%	0	453	149	2.51%	0	258	60	0.40%
2	56	59	35	0.49%	0	351	121	1.85%	0	186	52	0.39%
3	65	41	21	0.38%	0	458	138	2.61%	0	185	47	0.33%
4	36	46	26	0.53%	0	455	147	2.38%	0	252	60	0.41%
5	59	38	28	0.44%	0	303	113	1.87%	0	89	27	0.16%
6	53	44	28	0.44%	0	318	112	1.91%	0	142	40	0.25%
7	47	34	24	0.35%	0	412	122	2.39%	0	277	57	0.41%

Notes: Each column refers to the solution *after* that stage completes. Squared clearing error, α^2, is provided for over-subscription (+) and under-subscription (-) separately, with over-subscription in all cases measured relative to maximum capacity, $\hat{q}_j$. "Loss" is the percent deadweight loss from under-subscribed seats in positive-price courses (i.e., the total value of those seats, based on their price, relative to the total value of all seats).

A-2 (Spring)

Compute	Stage 1: Price Vector Search α^2				Stage 2: Over-subscription Elimination α^2				Stage 3: Under-subscription Reduction α^2			
Server	+	-	Seats	Loss	+	-	Seats	Loss	+	-	Seats	Loss
1	39	62	46	0.26%	0	178	86	1.07%	0	33	15	0.07%
2	53	67	43	0.33%	0	243	107	1.42%	0	31	13	0.08%
3	45	49	39	0.33%	0	195	89	1.03%	0	34	12	0.03%
4	40	157	85	0.63%	0	265	123	1.25%	0	92	30	0.23%
5	45	46	36	0.22%	0	216	82	0.99%	0	53	17	0.06%
6	36	52	36	0.33%	0	185	87	1.11%	0	3	3	0.00%
7	45	56	36	0.34%	0	154	74	0.81%	0	16	8	0.02%

Notes: Each column refers to the solution *after* that stage completes. Squared clearing error, α^2, is provided for over-subscription (+) and under-subscription (-) separately, with over-subscription in all cases measured relative to maximum capacity, $\hat{q}_j$. "Loss" is the percent deadweight loss from under-subscribed seats in positive-price courses (i.e., the total value of those seats, based on their price, relative to the total value of all seats).