Chapter 1

Background

1.1 Mechanism Design

Traditionally, economists and game theorists most vigorously study the set of problems involving the construction of systems that "fairly" (under various definitions of the word) address scenarios involving a host of participants with adverse or overlapping goals. Some examples of problems that fall under this classification are as follows: designing a voting system that successfully eliminates strategizing by any party [1], auctioning search engine advertisement spots in a way that promotes honest bidding [2], assigning roommates from a pool of students such that each student ends up with a roommate they prefer [3], and maximizing aid while allocating food donations to food banks [4].

It turns out that many mechanism design problems can be solved algorithmically, and as such computer scientists have increasingly joined the economists in the study of mechanism design theory. There are two subcategories of mechanism design relevant to this paper: stable matching and fair division.

1.1.1 Stable Matching

Matching problems are highly applicable to many real-world scenarios, such as assigning medical students to hospitals for residency [5] or the aforementioned roommates example. The canonical (and heteronormative) matching problem is known as the Stable Marriage Problem [6], in which the goal is to match an equal number of men and women such that no participant remains uncoupled. This is known as a bipartite matching problem, or one in which the participants are divided into disjoint sides and matched from one side to the other. The other important aspect of the Stable Marriage Problem is that the matching should prevent "cheating," which occurs when a man and a woman who weren't matched to each other prefer to be together over their assigned partners and will "elope," leaving the other two participants alone. This couple would be known as a blocking pair, because their elopement "blocks" the matching from being complete. Herein lies the stability aspect of stable matching: we

wish to find an algorithm that outputs a stable matching of couples, with no blocking pairs.

In 1962, David Gale and Lloyd Shapley published a seminal paper for the field of stable matching in which they proposed the deferred acceptance (DA) algorithm and proved it to output a stable matching [7]. Since then, the algorithm has been adapted to fit many more scenarios with various types of considerations, including the residency problem. However, there remain subsets of problems that do not fall under the same paradigm as the DA algorithm: for instance, problems with a one-sided market. The stable roommate problem is an example of one such problem, in which participants are to be matched from the same pool rather than two distinct categories of participant and stability retains the same meaning as in the bipartite setting. Over 20 years after Gale and Shapley's paper, in 1985 Robert Irving proved stability and efficiency for an algorithm that solves the roommate problem [3]. Irving's algorithm also now serves as a jumping-off point for the majority of further research on one-sided matching problems.

1.1.2 Fair Division

The difference between matching and fair division often lies in the primary metric used to evaluate the effectiveness of the solution. Matching is most predominately studied under stability, and fair division under analysis of envy. When dividing a chocolate-vanilla cake, a person who prefers vanilla but receives chocolate will be envious of those who receive a vanilla slice. In a fair division problem, the goal is to minimize the amount of envy. The cake example is the canon example for divisible goods, but there is also a lot of study into the division of indivisible goods, which introduces a new set of considerations for maximizing welfare (the benefit participants yield from the goods they receive) and minimizing envy. The food bank example is one in which the goal pertains to fairly allocating indivisible goods [4].

Formally, envy fair division will often be evaluated under pareto-optimality. An allocation is considered pareto-optimal if there is no other allocation that is strictly better for at least one participant while retaining the same level of welfare for the other participants [8]. As more factors are added to these problems— such as considering the order in which goods arrive for distribution, scaling the number of participants, and more specific definitions of welfare—envy freeness and pareto-optimality become more difficult to ascertain.

1.2 Adding constraints

What's known as the "vanilla" version of these matching and division environments, which is to say the more simple or basic versions, have been heavily studied over years of computer science and economics literature. So, the problems are often made more difficult or made to more accurately reflect real-world problems by adding *constraints*, therefore introducing new facets of the classic problem to solve. Constraints can

come in many forms, and can often make problems too difficult to solve efficiently or impossible to solve at all. In the scope of class assignment and related literature, constraints often come in the forms of quotas on class capacity and categories of students.

1.2.1 Quotas and Diversity

Quotas in allocation and matching problems can be implemented as upper and lower bounds. Upper bound quotas implies a capacity constraint, such as a maximum number of open positions per hospital in the residency matching problem. The College Admissions problem, in which a college with n applicants can admit up to q students, is traditionally studied with these capacity constraints (where q is for quota). In fact, the College Admissions problem was introduced with Gale and Shapley's Stable Marriage DA algorithm [7]. In recent years, these quotas have been extended in a variety of ways, such as introducing lower bounds in which schools also need to accept at least m students of some type [9]. Because adding quotas can make the problem too difficult, quotas are often studied under "soft" constraints as well: for example allowing a quota to be a target range rather than a specific number [10].

Another extension of the College Admissions problem that more accurately reflects reality comes in the form of diversity constraints. In the actual college admissions problem, these constraints are often known as Affirmative Action, in which there are majority upper-bounded quotas to give space for minority students, and there are many papers that mathematically study that particular implementation as well as prove that it does not always benefit the minority applicants [11]. As a result, diversity constraints can take many forms—they have been studied under frameworks of lower-bound quotas [9], proportionality (evenly distributing types of students in the matching) [12], and group fairness (ensuring each distinct group of students is as happy with their matching as possible based on the happiness of each individual student in the group) [13].

1.2.2 Big-O

Considering constraints such as quotas and diversity often make the problem much more difficult, and in fact in many cases it becomes unfeasible to solve. Here we define "unfeasible" to mean inefficient, which we can mathematically formalize with what's known as *Big-O* notation.

Big-O notation is a way of analyzing the efficiency of algorithms in computer science. The analysis takes place *asymptotically*, or as the size of the input becomes infinitely large. It is important to analyze algorithms asymptotically so we can guarantee that an algorithm fully solves a problem efficiently, as it is less helpful if it only works on small inputs. For instance, if I as an individual have an algorithm to bake cookies (a recipe) and the steps of rolling out the dough and measuring ingredients by hand work well for a batch of 24 cookies, that is sufficient for my use case. However, if I start a cookie franchise and suddenly I need 2,400 cookies a day, my original

algorithm is not going to be the most efficient way to make that many cookies, and therefore it is not a guaranteed algorithm for making cookies. It is more helpful in computer science to know our solutions will be helpful no matter the use case, instead of making many specific different algorithms for different cases.

Analyzing algorithms asymptotically allows us to estimate the running time of an algorithm, or how long an algorithm will take to terminate and output the result. In Big-O notation, we write $f(n) = O(g(n))$, where some algorithm, $f(n)$, will take at most $g(n)$ time to run. The O signifies the "at most" in that statement, or more formally it denotes an upper bound on the asymptotic running time.

Running time can be classified into different categories. The most relevant categories to this paper are constant time, polynomial time, and exponential time. Constant time, $O(1)$, can be abstractly thought of as any operation that can be completed instantly. Constant time is the fastest running time an algorithm can have. Most algorithms need to do more complicated operations than those that can be done in constant time, however, so our next category of time analysis is polynomial time. Recall that a polynomial is a mathematical expression that combines terms of variables and constants with various operations (e.g. $x^4 + 2x^3 + 7$ or $x^2 + y^2$). When we think about polynomial time, we think about how many operations must be completed per input to the algorithm. For instance, if I have cookie dough and I need to bake the cookies, my algorithm might be for every tablespoon of cookie dough roll the dough into a ball and then sprinkle cinnamon on top. That is 2 actions to complete for every tablespoon of cookie dough. So, if we have n tablespoons, we could say this step takes $O(n^2)$ time (note this would be more specifically quadratic time, but quadratic time is a subset of polynomial time). For our purposes, we can consider polynomial time to be sufficiently efficient. Our last category is exponential time, which is very slow in the world of running time and defines what we mean by "unfeasible". We consider an algorithm inefficient if it takes exponential time, such as $O(2^n)$. As n grows larger in this case, the running time increases rapidly. If we let $n = 1000$, $n^2 = 1,000,000$ which is indeed large, but not compared to 2^n, which becomes "a number much larger than the number of atoms in the universe" [14].

1.2.3 P vs NP

In time complexity theory, the field of mathematical analysis centered on the running time of algorithms, we give names to the aforementioned categories. Specifically, the most important complexity category for our purposes is P, which denotes the class of problems that can be solved with polynomial time algorithms. That means we can think of P as a collection of problems where every problem in the collection has an algorithm that will find a solution in polynomial time. Perhaps intuitively, one would assume that there exist many problems that necessitate algorithms that take longer than polynomial time (such as those that take exponential time, or belong to the class E). What becomes less intuitive is that there also exist problems that we cannot prove do or do not belong in P.

To understand what I mean by this, we need to discuss another important complexity class called NP: the class of problems solved by *nondeterministic polynomial* time algorithms. In very simplified terms, one could think of "nondeterministic" as brute force, where an algorithm that does a polynomial number of operations for each input goes through every possible input to a problem sequentially and carries out every operation on each input until a solution is discovered. If the first input happens to be the solution, then a polynomial number of operations have been performed, and therefore the algorithm terminated in polynomial time. It is not possible to guarantee that the first input to an algorithm would always yield a solution, but we can imagine that if there were n possible inputs to an algorithm and we happened to have n computers that can all run the algorithm on a distinct input, one of those computers would give a solution in polynomial time (if a solution exists). We could not *determine* which computer it would be, but we would know it exists. In real life, we do not have n computers for every n-input problem we wish to solve. However, if we were somehow given the solution output by that lucky computer, we could *verify* that it is correct by running the lucky computer's input on the real computer we do have in polynomial time. All of this is to say that NP is the class of problems that have algorithms for which we can verify a solution exists in polynomial time. We cannot necessarily find that solution in polynomial time, but if it was given to us we can prove it solves the problem in polynomial time.

Generally, we consider the class of problems in NP to be very difficult to solve, as it means no one has ever found an algorithm that can find a solution efficiently in polynomial time (if they had, the problem would be in P rather than np). the hardest NP problems are known as *NP-complete* problems, "complete" meaning the problem can be generalized to take the form of every other problem in NP. So, if a polynomial time algorithm was discovered for an NP-complete problem, it would mean that every np problem could be solved in polynomial time, and therefore all of those problems would actually just be in P [14].

1.2.4 Optimization

When we work with an NP or especially an NP-complete problem, we generally do not attempt to find an algorithm that gives an exact solution; the fact that the problem is in NP means we don't know if it is even possible to do that efficiently. However, many problems with relevant real-world applications are in NP—does that mean we simply give up trying to find a solution? Maybe we have to let go of the idea of an exact solution, but that doesn't have to stop us from trying to find a solution we deem *good enough*.

"Good enough" can mean many things depending on the problem, which is the driving force behind *optimization* Optimization problems attempt to find a near-optimal (if not optimal) solution under given constraints. Often, these will take the form of *approximation* algorithms, which will relax some of the constraints or allow violation of the constraints up to some upper bound to guarantee a solution that is as close to the optimal solution as mathematically possible. We can clarify "near-

optimal" based on the value of the optimal solution: if an optimal solution has some value, we want to find a solution that maximizes (or minimizes) how close we get to that value. Another optimization approach is with *heuristic* algorithms, which leans more into the "good enough" mentality by focusing on the empirical results. Rather than making mathematical guarantees about the solution, the algorithm will be studied for its performance on actual data, and usually the performance is adequate to be considered a reasonable solution. The goal of both approximation and heuristic algorithms is to find an adequate solution quickly when the optimal solution cannot be reasonably found. There are other approaches to solving optimization problems, but these approaches are the most relevant to this paper.

1.3 Graph Theory

In discrete mathematics, a graph is a collection of nodes called *vertices* connected by *edges* (denoted $G = (V, E)$ where G is the graph, and it is composed of a set of vertices V and a set of edges E). Mathematicians like graphs because their structure makes them very effective for modeling a variety of real-world situations such as social networks (where vertices are individuals, and edges connect people to their acquaintances) or even the spread of disease (where vertices are people and edges represent an infection from one person to another).

Graphs can famously be used to model scheduling problems as well by adjusting the structure slightly. A basic graph problem has vertices and edges, but we can add factors such as *directional* edges that denote going from one vertex to another and not back, or some number of *colors* where we want to color vertices such that no two adjacent vertices (vertices connected by an edge) are the same color. The latter example is actually one of the most studied NP-complete problems: is it possible to color a graph G using k colors? It is also the structure under which scheduling has been extensively studied, where vertices represent events, edges encode overlapping time conflicts between events, and the number of colors is the number of available time slots into which those events can be scheduled [15].

Another relevant construction of graphs is to add *weights* to edges. A weight in this context is some value derived based on the constraints of the problem, and this form of graph often lends itself to optimization. As in, the weighted graph can be used to model problems where we want to maximize or minimize the weights. For instance, we can construct a graph where each vertex represents a city and each edge represents a highway from one city to another. If we were planning a roadtrip, we could add a weight to each edge to signify the travel time of that highway from one city to another. Then, if we were planning a road trip to all the cities on the graph, we could find the fastest route by finding the path with the smallest weight.

1.4 Related Work

The problem we discuss in this thesis is related to the extensively-studied problems of school choice and course allocation, both of which harbor a pool of significant mechanism design work in the field of economics. School choice in mechanism design is the problem of assigning high school students to high schools such that parents are satisfied, each student has a school to attend, and no school admits more students than they can accommodate. Course allocation in mechanism design is the problem of constructing a system to assign students to classes based on which classes they prefer to take.

1.4.1 Two-sided matching

In 2003, Abdulkadiroğlu and Sonmez published a seminal paper on school choice, in which they view the problem through the bipartite lens of schools and students each with distinct preferences. This lens allows them to apply the Gale-Shapley DA algorithm as a solution as well as their own algorithm, which they called Top Trading Cycles (TTC) [16]. In Top Trading Cycles, if the matching has a cycle of students in which each student prefers the next student's school, each assignment is given to the next student until the cycle is broken.

Six years later, in 2009, Abdulkadiroğlu, Pathak, and Roth amended the 2003 paper with a case study into the New York City high school allocation mechanism, scrutinizing the DA algorithm under strategy-proofness and stability. They show empirically that there is a loss of efficiency in the algorithm when adding strategy-proofness and stability to the environment [17].

Many papers, including some outside of school choice specifically, extend one or both of Abdulkadiroğlu's papers in further research. In 2010, Biró et al studied school choice in a college setting with upper and lower quotas, constraining the environment in a new way. They determined that stable matchings do not always exist with those quotas, and in fact determining whether a stable matching exists at all is NP-complete [9]. Three years later, Hafalir et al analyzed the DA algorithm for school choice under quotas for diversity, where they found that using upper quotas produced a matching that was pareto-dominated by a matching with lower quotas for minority students [11]. There has since been much more research into the stability effects of quotas on the two-sided school choice problem, including papers that implicate stability with diverse students of multiple types [18].

1.4.2 One-sided matching

Since 2003, research also branched from two-sided school choice into one-sided matching, where only one side of the market has preferences. This was often studied under the framework of course allocation, a mechanism design problem which arose from the existing school choice literature. In 2011, Budish and Cantillon published the seminal paper on course allocation at Harvard (which cites Abdulkadiroğlu's 2003 and 2009

papers) in which they evaluate Harvard's mechanism under strategyproofness and pareto-optimality [19]. There have been slight alterations to their framework since then, such as the effects of incomplete preferences on participant welfare.

Only in recent years have scholars started studying one-sided matching with quotas and other diversity constraints. In fact, the constraints closest to that in this paper (groups of students as well as sub-types within the groups) only seem to exist in literature from recent months of 2025. Specifically, Santhini et al researched the empirical effects of both soft quotas [10] and group fairness [20] on the mechanism using convex optimization (also done similarly by Panda et al [13]). There is not readily as much theory-based research on one-sided matching with constraints as two-sided matching, and those that do study it under a theoretical framework use more of a fair division analysis than stability.

As far as we can tell, the closest paper to our construction of stability in this thesis is an unpublished 2011 paper by Abdulkadiroğlu, who generalizes the two-sided and one-sided settings to one environment where the TTC algorithm can be reduced to work for both kinds of preferences. In the paper, he frames stability to be "if, whenever a student prefers a school to her assignment, that school is enrolled up to its capacity by students who have higher priority at that school" [21]. The paper does not consider the effect of adding diversity constraints.

Bibliography

[1] William S. Zwicker. "Introduction to the Theory of Voting". In: *Handbook of Computational Social Choice*. Ed. by Felix Brandt et al. Cambridge University Press, 2016, pp. 23–56.

[2] William Vickrey. "COUNTERSPECULATION, AUCTIONS, AND COMPETITIVE SEALED TENDERS". In: *The Journal of Finance* 16.1 (1961), pp. 8–37. DOI: https://doi.org/10.1111/j.1540-6261.1961.tb02789.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1961.tb02789.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1961.tb02789.x.

[3] Robert W. Irving. "An efficient algorithm for the "stable roommates" problem". In: *Journal of Algorithms* 6.4 (1985), pp. 577–595. ISSN: 0196-6774. DOI: https://doi.org/10.1016/0196-6774(85)90033-1. URL: https://www.sciencedirect.com/science/article/pii/0196677485900331.

[4] Martin Aleksandrov et al. "Online Fair Division: analysing a Food Bank problem". In: *CoRR* abs/1502.07571 (2015). arXiv: 1502.07571. URL: http://arxiv.org/abs/1502.07571.

[5] Alvin E. Roth. "The Origins, History, and Design of the Resident Match". In: *JAMA* 289.7 (Feb. 2003), pp. 909–912. ISSN: 0098-7484. DOI: 10.1001/jama.289.7.909. eprint: https://jamanetwork.com/journals/jama/articlepdf/195998/jrf20024_909_912_1693348147.98749.pdf. URL: https://doi.org/10.1001/jama.289.7.909.

[6] Jon Kleinberg and Éva Tardos. *Algorithm Design: 1. Stable Matching*. Lecture Notes. 2005. URL: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/01StableMatching.pdf.

[7] David Gale and Lloyd S. Shapley. "College admissions and the stability of marriage." In: *American Mathematical Monthly* 69 (1962), pp. 9–15. URL: https://www.jstor.org/stable/2312726.

[8] Sean Ingham. *Pareto-optimality*. URL: https://www.britannica.com/money/Pareto-optimality.

[9] Péter Biró et al. "The College Admissions problem with lower and common quotas". In: *Theoretical Computer Science* 411.34 (2010), pp. 3136–3153. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2010.05.005. URL: https://www.sciencedirect.com/science/article/pii/S0304397510002860.

[10] Santhini K A, Raghu Ravi, and Meghana Nasre. "Matchings under one-sided preferences with soft quotas". In: (Mar. 2025). DOI: 10.2139/ssrn.5163631.

[11] Isa E. Hafalir, M. Bumin Yenmez, and Muhammed A. Yildirim. "Effective affirmative action in school choice". In: *Theoretical Economics* 8.2 (2013), pp. 325–363. DOI: https://doi.org/10.3982/TE1135. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.3982/TE1135. URL: https://onlinelibrary.wiley.com/doi/abs/10.3982/TE1135.

[12] Thành Nguyen and Rakesh Vohra. "Stable Matching with Proportionality Constraints". In: *Operations Research* 67.6 (2019), pp. 1503–1519. DOI: 10.1287/opre.2019.1909. eprint: https://doi.org/10.1287/opre.2019.1909. URL: https://doi.org/10.1287/opre.2019.1909.

[13] Atasi Panda et al. "Group Fair Matchings using Convex Cost Functions". In: (2025). arXiv: 2508.12549 [cs.GT]. URL: https://arxiv.org/abs/2508.12549.

[14] Michael Sipser. *Introduction to the Theory of Computation*. 2nd ed. Course Technology, 2006.

[15] S G Shrinivas S G Shrinivas, Santhakumaran Vetrivel, and N Elango. "APPLICATIONS OF GRAPH THEORY IN COMPUTER SCIENCE AN OVERVIEW". In: *International Journal of Engineering Science and Technology* 2 (Sept. 2010).

[16] Atila Abdulkadiroğlu and Tayfun Sönmez. "School Choice: A Mechanism Design Approach". In: *American Economic Review* 93.3 (June 2003), pp. 729–747. DOI: 10.1257/000282803322157061. URL: https://www.aeaweb.org/articles?id=10.1257/000282803322157061.

[17] Atila Abdulkadiroğlu, Parag A. Pathak, and Alvin E. Roth. "Strategy-Proofness versus Efficiency in Matching with Indifferences: Redesigning the NYC High School Match". In: *American Economic Review* 99.5 (Dec. 2009), pp. 1954–78. DOI: 10.1257/aer.99.5.1954. URL: https://www.aeaweb.org/articles?id=10.1257/aer.99.5.1954.

[18] Ryoji Kurata et al. "Controlled school choice with soft bounds and overlapping types". English. In: *Journal of Artificial Intelligence Research* 58 (Jan. 2017). Publisher Copyright: © 2017 AI Access Foundation., pp. 153–184. ISSN: 1076-9757. DOI: 10.1613/jair.5297.

[19] Eric Budish and Estelle Cantillon. "The Multi-unit Assignment Problem: Theory and Evidence from Course Allocation at Harvard". In: *American Economic Review* 102.5 (May 2012), pp. 2237–71. DOI: 10.1257/aer.102.5.2237. URL: https://www.aeaweb.org/articles?id=10.1257/aer.102.5.2237.

[20] Santhini K. A. et al. "Group Fairness and Multi-Criteria Optimization in School Assignment". In: *6th Symposium on Foundations of Responsible Computing (FORC 2025)*. Ed. by Mark Bun. Vol. 329. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 20:1–20:20. ISBN: 978-3-95977-367-6. DOI: 10.4230/LIPIcs.FORC.2025.20. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FORC.2025.20.

[21] Atila Abdulkadiroglu. "Generalized matching for school choice". In: *Unpublished paper, Duke University.[1311, 1312]* (2011).