Chapter 1

Background

1.1 Analyzing Algorithms

An algorithm is a list of specific instructions written to achieve some goal. The goal varies, and in casual conversation individuals have come to associate *algorithm* with the phrase "my algorithm" or "the algorithm," an entity with the goal to create profit for big tech companies by keeping users engaged on their apps. While those algorithms exist, they are only one type of algorithm. A cookie recipe is an algorithm, providing instructions a baker follows to yield a batch of cookies upon completion. GPS devices provide algorithms for driving from one location to another. In computer science, an algorithm is a sequence of lines of code that perform a computation, usually mathematical calculations or processing data. Because algorithms are so versatile, and there are so many different ways of accomplishing one goal, we analyze algorithms by their efficiency: how many resources they use and how much time they take to complete the task.

1.1.1 Big-O

To analyze efficiency, we have to define what we mean for an algorithm to be "efficient." For the sake of this thesis, we will focus on the time aspect of efficiency. The amount of time an algorithm takes to execute from start to finish is known as the *running time* of an algorithm. We can mathematically formalize running time withwhat's known as *Big-O* notation.

Big-O notation is a form of time analysis that 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 all scenarios of a problem efficiently, as it is less helpful if the algorithm only works well on small inputs. For instance, if I am following an algorithm to bake cookies and the steps involve rolling out the dough, mixing, and measuring ingredients by hand, the algorithm works well for me to make a batch of 24 cookies. However, if I start a popular 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. While it may make sense when baking cookies to handle these two cases differently, it is more helpful in computer science to know our algorithms will efficiently solve our problem no matter the use case because data commonly scales to very large sizes.

We *estimate* running time through asymptotic analysis. 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 thesis 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 involve 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 using 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 actions must be completed per input to the algorithm. For instance, if I already 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 compared to our other two categories. We consider an algorithm "inefficient" if it takes exponential time. An example of exponential time in Big-O is $O(2^n)$. As n grows larger in this case, the running time increases extremely rapidly. To compare polynomial time and exponential time, imagine we have some number $n = 1000$. If we square n, we get a polynomial expression: $n^2 = 1,000,000$. That is indeed large, but not compared to 2^n, which is "a number much larger than the number of atoms in the universe" [1].

1.1.2 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, we know there exist many complex problems that take longer than polynomial time to solve (such as those that take exponential time and belong to the class E). A less intuitive fact 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 a brute force approach: an algorithm that takes polynomial time for each input goes through every possible input to a problem sequentially until it finds a solution. If the first input happens to be the solution, then the algorithm has been carried out once, so a polynomial number of operations have been performed. Therefore, the algorithm terminated in polynomial time. It is not possible to guarantee that the first input to an algorithm will 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 run the lucky input and 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 an efficient polynomial time solution (if they had, the problem would be in P rather than NP). The hardest NP problems are known as *NP-complete* problems, where “complete” means 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 [1].

1.2 Mechanism Design

Mechanism design has traditionally been studied by economists and game theorists, though in recent years computer scientists have increasingly joined the field because mechanisms are often just algorithms. Mechanism design is 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 [2], auctioning search engine advertisement spots in a way that promotes honest bidding [3], assigning roommates from a pool of students such that each student ends up with a roommate they prefer [4], and maximizing aid while allocating food donations to food banks [5].

One of the most standard metrics for evaluating the effectiveness of a mechanism is whether it is *strategy-proof*, or prevents participants from taking advantage of the rules of the mechanism for their own gain. Many more metrics exist for evaluating subcategories of mechanism design, but almost all mechanisms can be evaluated under strategy-proofness. There are two subcategories of mechanism design relevant to this thesis: stable matching and fair division.

1.2.1 Stable Matching

Many real world scenarios can be modelled as matching problems, such as assigning medical students to hospitals for residency [6], or the aforementioned roommates example (known as the Stable Roommates problem). The standard (and heteronormative) matching problem is known as the Stable Marriage Problem [7], 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 market of participants is divided into two disjoint sides and matched from one side to the other. Each participant provides a list of preferences over the other side, where they rank those whom they would most like to be matched to. These are known as *two-sided* preferences. An essential 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 complete matching of couples with no blocking pairs.

[FIGURE HERE ILLUSTRATING STABLE MARRIAGE]

In 1962, David Gale and Lloyd Shapley published a seminal paper for the field in which they proposed the **deferred acceptance** (DA) algorithm, a solution that guarantees a stable matching for the Stable Marriage problem in polynomial time[8]. Gale and Shapley also proved their DA algorithm could solve the College Admissions problem: matching high school student applicants to colleges. College Admissions differs slightly from Stable Marriage: it is a *many-to-one* matching in which colleges accept multiple students rather than each college matching with a single student, and it has *incomplete preferences* because students do not rank (apply to) every possible college on the list. With a slight adaptation of the DA algorithm, it can solve this version of College Admissions. Since 1962, the DA algorithm has been adapted to fit many more problems with additional factors, 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 Roommates problem from above is an example of a one-sided market, in which participants are matched within one undivided pool of participants rather than between two distinct categories. Stability in a one-sided market retains the same meaning as in a bipartite setting. Over 20 years after Gale and Shapley's paper, Robert Irving published a paper with a solution to the Stables Roommates problem, in which he proved his algorithm outputs a stable matching of roommates in polynomial time [4].

Irving's algorithm, similarly to the DA algorithm in bipartite markets, provides the groundwork for the majority of research on one-sided matching problems.

1.2.2 Fair Division

The difference between matching and fair division (also known as fair allocation) is most often one-sided preferences. "One-sided" preferences has a different meaning than a "one-sided" market: one-sided preferences still means a bipartite market, but only one side cares what they are matched to. Because only one side cares, we no longer have the idea that "cheating" could occur; instead, we evaluate fair division under *envy*. Say we have a cake that is half chocolate and half vanilla, and we are dividing the cake between people at a party. 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 between participants and maximize the amount of *welfare*, where welfare is the benefit participants yield from the goods they receive (such as feeling significantly more joy from vanilla over chocolate).

[CAKE CUTTING FIGURE HERE]

The cake-cutting example is the canon example for divisible goods [9], but there is also a lot of study into the allocation of indivisible goods: items that cannot be cut up like a cake. The aforementioned food bank example, allocating donated items (such as non-perishable cans of food) to different food banks based on preferences they provide, involves indivisible goods [5].

In addition to envy and welfare, 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 welfare for the other participants [10]. So, we say allocation A *pareto-dominates* allocation B if allocation A is better or the same than allocation B for every participant. As more factors are added to fair allocation 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 achieve.

1.3 Adding constraints

What's known as the "vanilla" versions of these matching and division environments, which is to say the more simple or basic versions, have been studied thoroughly by now. So, to advance mechanism design research and to more accurately model real-world scenarios, scientists add *constraints* to the problems. Constraints come in many forms, and often make problems so complex they become NP-complete. In the scope of class assignment and related matching problems, constraints often come in the forms of quotas on class capacity and categories of students.

1.3.1 Quotas and Diversity

Quotas in allocation and matching problems can be implemented as upper and lower bounds on the mechanism. Upper bound quotas imply capacity constraints, such as a maximum number of open positions per hospital in the residency matching problem. Gale and Shapley's 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). Lower bound quotas in something like the College Admissions would encode in the mechanism that schools need to accept at least m students of some type [11]. Strict quotas are one such constraint that can cause finding a stable matching to be NP-complete, so quotas are often studied under "soft" constraints as well: for example, allowing a quota to be a target range rather than a specific number [12].

This quota-based extension of the College Admissions problem more accurately reflects reality by acting as diversity constraints. The goal of diversity constraints is usually to produce a matching/allocation in which some type of participants is spread evenly or represented accurately in the result. In the actual college admissions problem, these constraints are commonly known as Affirmative Action, in which there are upper-bounded quotas on majority students to give space for minority students. There are many papers that mathematically study that particular implementation of diversity, as well as prove that it does not always benefit the minority applicants [13]. As a result, diversity constraints can take many forms—they have been studied under frameworks of lower-bound quotas [11], proportionality (evenly distributing types of students in the matching) [14], 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) [15]. In each of those papers implementing diversity constraints, the constraints make the problems NP-complete.

1.3.2 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 find an exact solution efficiently. However, as we have seen with quotas and diversity, 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 a 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 what we mean by "near-optimal" based on 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 even more into the "good enough" mentality by focusing on the empirical results. Rather than making mathematical guarantees about the optimality of the solution, a heuristic algorithm will be evaluated for its performance on actual data. Usually, despite the worst-case scenario being mathematically unfeasible, the algorithm is able to output a solution when given real scenarios, and therefore we consider it to be 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 thesis.

1.4 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). Many matching and allocations problems can even be modelled as graphs.

[BASIC GRAPH FIGURE HERE]

Graphs can famously be used to model scheduling problems by adjusting their structure slightly. A basic graph problem has vertices and edges, but we can add factors such as *directional* going from one vertex to another, or some number of *colors* with which 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 of graph 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 [16].

[GRAPH COLORING FIGURE]

Another way to construct 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.

[WEIGHTED GRAPH FIGURE]

1.5 Related Work

The problem we discuss in this thesis is related to two extensively-studied problems: school choice and course allocation. 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.

1.5.1 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) [17]. 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 [18].

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 [11]. 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 [13]. 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 [19].

1.5.2 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 [20]. 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 [12] and group fairness [21] on the mechanism using convex optimization (also done similarly by Panda et al [15]). 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" [22]. The paper does not consider the effect of adding diversity constraints.

Bibliography

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] *The Holy Bible: King James Version*. Genesis 13:9. 1611. Chap. 13.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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`.

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