Essays on college admissions and fair team formation

Elizaveta Victorova

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy in Economics

Adam Smith Business School
College of Social Sciences
University of Glasgow

May, 2022

Abstract

The thesis consists of three Chapters.

The first Chapter considers college admission process with restricted applications. Each student has heuristic beliefs about the probability of being admitted and applies to a limited number of programmes within the colleges. We find that reducing the number of applications may result in an increase in total utility.

Second Chapter focuses of a team formation problem in markets with indivisible goods without transfers. Each agent is allocated equal number of goods. Since fairness notions are unattainable, we consider the relations between approximate fairness properties. We find that, contrary to general case, most approximate fairness and minimal guarantee properties are logically independent in relatively big markets. We show that when there are two agents, envy-freeness up to one good and optimality are compatible. We examine some of the well-known assignment rules and find that Round Robin and Generalised Round Robin are not efficient, although satisfy some of the approximate fairness properties. Nash Max, Utilitarian Max and Leximin procedures are efficient, but do not guarantee that any of the approximate fairness notions hold.

Third chapter investigates allocating students to equal number of elective courses. Compared to Chapter 2, here we examine many-to-many markets. We show that the Deferred Acceptance (DA) mechanism may violate the quotas, and design two modifications. We show that the modifications may allow for additional profitable manipulations compared to DA. We then simulate the allocation procedures and show that, on average, there are no more than 1% of additional profitable manipulations. When considering the number of manipulable markets, we find that roughly 20% and 18% of the markets are manipulable under our two modifications, which is low compared to the similar outcomes in one-to-many markets without lower quotas.

Contents

Abstract i

Acknowledgements vii

0 Introduction 1

1 Truncated allocations under DA mechanism: consequences on students' strategies and welfare 4

1.1 Introduction 4

1.2 Model 7

1.2.1 Allocation process 8

1.3 Results 10

1.3.1 Unrestricted reports 10

1.3.2 Restricted reports 11

1.3.3 Welfare 13

1.3.4 Discussion 16

1.4 Extensions of the model 17

1.4.1 Application assumptions 17

1.4.2 Student types 18

1.4.3 Three programmes 19

1.4.4 Welfare 24
1.5 Numerical Experiment 27
1.6 Conclusion 32
1.7 Appendix. Proofs of propositions 34

2 Fair team formations with allocations of equal size 40
2.1 Introduction 40
2.1.1 Motivation 43
2.2 Literature 46
2.3 Model 49
2.4 Properties: definitions 50
2.4.1 Pareto Optimality 50
2.4.2 Envy-Freeness 51
2.4.3 Proportionality 52
2.5 Assignment rules: definitions 53
2.6 Compatibility of properties 55
2.7 Mechanisms' Properties 65
2.8 Conclusion and Future Work 72
2.9 Appendix. Summary of results 74

3 Student allocation to equal number of elective courses 75
3.1 Introduction 75
3.2 Literature 77
3.3 Model 79
3.3.1 Why we cannot use original DA 79

3.3.2 Modified DA 81
3.4 Strategy proofness 83
3.4.1 ModDA1 89
3.4.2 ModDA2 92
3.4.3 Simulations 94
3.5 Conclusions and Future Work 96

A Appendix to Chapter 1 **101**

B Appendix to Chapter 3 **112**

List of Figures

1.1 Student's estimate of the probability of acceptance into p 10

1.2 Final matching, baseline 11

1.3 Optimal student's strategy 13

1.4 Students' allocation: case 1 if $s_b^e > s_b'$ (top) and $s_b^e < s_b'$ (bottom) 15

1.5 Students' allocation: case 2 16

1.6 Final matching with unrestricted applications 21

1.7 Optimal student's strategy 22

1.8 Student's expected utility comparison depending on the preference profile 23

1.9 Optimal student's strategy 24

1.10 Example of an increase in total utility (application to one programme) 25

1.11 Example of an increase in total utility (application to two programmes) 26

1.12 Number of reallocations and change in total utility with normally distributed exam scores 29

1.13 Number of reallocations and change in total utility with uniformly distributed exam scores 29

1.14 Number of reallocations and change in total utility with normally distributed exam scores and distorted previous year's cutoff scores 30

1.15 Number of reallocations and change in total utility with uniformly distributed exam scores, four colleges 31

1.16 Student's expected utility comparison, student prefers programme a . . 34

1.17 Student's expected utility comparison, student prefers programme b . . 35

List of Tables

1.1 Expected utility of an application a_i of student i 21

2.1 RR allocation 65

2.2 Pareto Improvement (PI) 66

2.3 Summary or results 74

3.1 Fraction of beneficial misreports among all misreports under DA and ModDA1 95

3.2 Fraction of beneficial misreports among all misreports under DA and ModDA2 95

Acknowledgements

I am very grateful for all the support I received during the last few years.

This thesis would not be possible without my supervisors. I am thankful to Dr Patrick Harless for his support and guidance at the start of my PhD research. I am grateful to Prof Anna Bogomolnaia, Prof Herve Moulin and Dr Constantine Sorokin for their expertise, advise and patience.

I would like to extend my sincere thanks to the University of Glasgow for providing me with the opportunity to study here. Being awarded with the College of Social Sciences Scholarship opened many doors.

Thank you, Arthur, Jerome, Rohan, Damiano, Vladimir, Johanna, Fivos, Li, Mo, Hualin, Arsenii, Nikolas, Helena for the hours of studies, discussions and suggestions.

I am deeply grateful to Anton, Arman, Dasha and Kirill.

Special thank to my parents for setting a good example, encouraging my curiosity and taking an active part in my life and education.

Declaration

I declare that, except where explicit reference is made to the contribution of others, this dissertation is the result of my own work and has not been submitted for any other degree at the University of Glasgow or any other institution.

Printed Name: *Elizaveta Victorova*

Chapter 0

Introduction

The thesis is composed of three chapters. Each chapter is a paper contributing to matching literature.

Chapter 1 focuses on deferred acceptance (DA) allocations in college admissions. The student-proposing DA mechanism is well-known in the literature for being stable and resulting in the best possible stable allocation for students. According to our model, however, students are applying to a limited number of programmes within each college. This implies that the students need to decide which programmes to apply to and which ones to amend in the reported preferences. The constraint is drawn from real-life examples of college admissions when students are only allowed to apply to up to a certain number of colleges or programmes within a college. We assume that students have heuristic beliefs about the probability of being admitted into each programme. The probabilities are based on the previous year's cutoff scores. We focus on changes to the total utility of all students compared to baseline case in which the students are allowed to apply to all programmes. We investigate cases with two and three colleges analytically. We find that, contrary to expectations, the total welfare of students may increase as a result of truncating preference lists. This happens when high-scoring students predict the cut-off scores incorrectly, and their 'baseline seats' are allocated to lower-scoring students who gain higher utility from the

programme. We then run some numerical experiments to verify our findings, and find that the results of our model are replicated by the simulations.

Chapter 2 studies fair division of goods in which every agent receives equal number of goods. This is a specific case of a general fair division when each agent receives any number of goods, and the number of goods depends on the utility of each good and its fairness implications. Allocating equal number of goods is a problem that is often seen in reality. Doctors are assigned the same number of night shifts per year, professors supervise equal number of students and students pick equal number of elective courses to graduate with a specific degree. These examples are often resolved in a heuristic manner or on a first come first served basis. Such approaches are impossible in professional sports. Drafting new player in major leagues means choosing equal number of new players by the teams. The process is well-regulated in attempt to make drafting as fair as possible. We study the properties of approximate fairness in this set-up. Usual notions of envy-freeness and proportionality are unfeasible, so we tailor their approximate equivalents to our case. We also consider minimal guarantees: minimal and lexicographic envy-freeness. We find that, contrary to general case, most approximate fairness and minimal guarantee concepts are independent and only imply each other when number of agents or number of goods is very small (two, sometimes up to three). Considering goods, bads or mixed goods has no effect on whether the properties hold, which is drastically different in the general case. We find that among the well-known allocation mechanisms, there are none that satisfy both optimality and approximate fairness. Round Robin mechanism is approximately fair up to one good and guarantees minimal envy-freeness, but is not optimal. Mechanisms that maximise the minimum utility among all the agents or maximise the sum or product of all utilities are optimal, but do not guarantee any

of the approximate fairness or minimal guarantee properties.

Chapter 3 focuses on allocating students to courses in a many-to-many matching with equal quotas. Each student needs to choose the exact number of elective courses, and the allocation takes the preferences of both students and courses into account. We assume that both students and courses have responsive preferences. First, we show that using the DA mechanism is not feasible as the allocation may result in a student being allocated to a few copies of the same course. We therefore design two modifications of the DA rule (ModDA1 and ModDA2) to avoid such allocations. If the DA assignment of students to courses does not violate any restrictions, we make sure that ModDA1 and ModDA2 results in the same allocation as DA. We then study the implications of modificaitions on strategy-proofness. We find that the modifications make some of the DA manipulations ineffective or even harmful, but also allow for appearance of new manipulations that were not effective under DA. We then simulate the allocation procedures and find that the number of profitable manipulations increases by less than 1% on average. Most of the manipulations are beneficial under both DA and its modifications. Overall, roughly 20% of the markets are manipulable under ModDA1 and about 18% under ModDA2, which is low compared to the similar outcomes in one-to-many markets without lower quotas in [4].

Chapter 1

Truncated allocations under DA mechanism: consequences on students' strategies and welfare

Abstract

In many countries, students are allowed to apply to a limited number of programmes as part of the college admission process. We find surprising consequences of such limitations on welfare: decreasing number of applications may have a positive effect on students' total utility. Assuming that students base their application decision on a heuristic belief about acceptance probability, we show this result when there are two programmes and demonstrate that it persists if the number of programmes is increased to three. We replicate these results in a numerical experiment.

1.1 Introduction

Millions of students seek admission to colleges each year, often in a centralised system[1]. Students submit their preferences to a central planner who allocates the seats to students according to a predetermined proce-

[1]Countries which include Australia [18], China [45], Turkey [5], Greece and South Korea [21] follow centralised procedures based solely on a national examination. Additionally, Hungary [6], Spain [36] and Germany [14] follow centralised processes that also account for student's preferences and other factors.

dure. Policy-makers choosing the allocation mechanism aim to increase students' welfare while balancing other social goals.

We study a stylised admission process model with a continuum of students and two programmes. Students take a standardised test and, after learning their scores, submit a preference list to a central planner. They apply to specific programmes in different colleges rather than applying to colleges. Students may only include a limited number of programmes on the submitted list, so, they must strategically choose which ones to include on the list. As students' preferences differ, some programmes may be more popular making admission more competitive. To estimate the admission probability, students apply a heuristic based on the programme-specific cutoff scores reported the previous year. Specifically, students estimate the probability that a programme's cutoff score takes a particular value to be linear function of its distance from last year's cutoff score. Although all students apply the same heuristic, some are more confident in their beliefs and are distinguished by the size of the interval of scores on which they place positive probability. The distribution of students' scores is independent of their preferences. In our baseline model, students may include both programmes to the reported preference list. We then limit the number of applications to one and compare the resulting welfare to the baseline case. We first derive equilibria in each case (Proposition 2). Our main result shows that, surprisingly, the restriction may improve students' welfare (Proposition 3). We then consider some extensions and show that the increase in welfare may also be present if there are three programmes.

The programmes admit students based on the exam scores by deferred acceptance (DA) mechanism that was adapted[2] to incorporate priorities in one-sided markets, where colleges are not strategic players, by Abdulkadiroglu and Sönmez [1]. Hafalir et al. [21] base the priorities

[2]The original version of the rule is due to Gale and Shapley [20]

of the students on their exam scores. They, however, assume that all students have the same preferences and can only apply to one programme even when their number is large. Abdulkadiroğlu et al. [2] study student's welfare resulting from Boston and DA mechanisms. They show that Boston mechanism is underrated in the literature because of unrealistic assumptions and in fact possesses several desirable features that DA does not, and students may be better off under Boston mechanism rather than DA. Miralles [30] simulates more realistic cases and shows that the results agree with [2]. This suggests that all results need to be checked for robustness.

Students' choice of optimal strategy depends on the expected utility received from studying a programme which, in turn, depends on the probability of being accepted. Thus, it is important to take uncertainty about other students' preferences into account. Allan Hernandez-Chanto [22] analyses centralised assignment to majors within the University of Costa Rica. As the rules remain constant year after year students estimate the probability of being admitted into each major based one the past cutoff scores published by the university using statistical tools. In a paper by Julia Varga [44] that studies the effect of expected wages and admission probabilities on students' application strategies, students only know the cutoff score in the preceding year. Admission probability is calculated as a ratio of student's total score to the cutoff score, it is linear in score and reaches zero only when the total score of the student is zero as well. Both authors try to be accurate in their estimations, however, when students have some information and estimate the chances of the applications to be successful, they tend to make mistakes. Experimental studies by Pais and Pintér [33] and Pais et al. [34] show that amount of information in a matching game has an impact on the strategies students choose under DA, top trading cycles and Boston mechanisms. Under partial or full information students of-

ten misreport their preferences even when truth-telling is the dominant strategy. Thus, in our study we do not assume the correct beliefs and introduce a belief system to analyse students' problem.

The paper proceeds as follows. Section 1.2 introduces the model, Section 1.3 derives our main results. Section 1.4 extends the model and shows robustness of our results as the number of programmes increases. We run some numerical simulation with greater number of colleges in Section 1.5. Section 1.6 concludes.

1.2 Model

There is a continuum of students $I = [0, 1]$ and two programmes $P = \{a, b\}$ to which they can apply. Students' preferences differ so that the fraction α_a prefer a and the remaining fraction $\alpha_b = 1 - \alpha_a$ prefer b. We denote the top-ranked programme in student i's preferences as t_i, the other one as b_i. Students' utilities are common among students:

$$u_i(p) = \begin{cases} 1 & \text{if admitted into } p = t_i \\ 1/2 & \text{if admitted into } p = b_i \\ 0 & \text{if not admitted} \end{cases} \tag{1.1}$$

Each student has an exam score E_i drawn from a continuous cdf $F(\cdot)$ with density $f(\cdot)$. Each programme has a capacity $q_i \in [0, 1]$. Students estimate their admission chances based on programme-specific cutoff scores $S = \{s_a, s_b\}$ in the previous year's admission process. Without loss of generality, we assume that $s_a \geq s_b$, which implies that programme a was more competitive than b.

We examine a model with linear utilities, which means that student's evaluation of the programme depends linearly on the ranking of the programme in the student's preferences. Assuming that the structure of the utility function is the same for all students is the first step in moving

away from identical preferences. This model can be altered to allow more general utilities. For example, b_i may yield utility x above or below $1/2$. Exceeding $1/2$ could imply that a student has strong preference of being admitted anywhere rather than remaining unmatched. On the contrary, a student may have a low value for a programme, then x would be close to zero. In addition, the utility from being admitted into b_i may be different for each student i. In this paper, we focus our attention and derive the results based on utility function (1.1). We provide some comments about potential changes if $u_i(b_i) = x$, however, detailed explanations of the modifications of the utility function are a subject of future research.

1.2.1 Allocation process

An *application* of a student i is a list $a_i \in \mathcal{P} = \{(a), (b), (a, b), (b, a)\}$ where programmes are listed in order of preference. We study two cases: unrestricted and restricted reports. When the length of the report is unrestricted, students report their preferences over both programmes $a_i \in \{(a, b), (b, a)\}$; if the submitted preferences are restricted then $a_i \in \{(a), (b)\}$. Students are assigned to programmes in the colleges using the student-proposing deferred acceptance algorithm, colleges compare students based on their exam scores:

Step 1. Each student applies to his top choice in a_i. Colleges put the best q_j applicants on the waiting lists of each programme $j = a, b$ and reject the rest.

$\vdots$

Step m. Each student that was rejected at the previous step applies to his next choice in a_i. For each programme j, the best to date q_j applicants are put on the waiting list and the rest are rejected.

$\vdots$

Termination. The algorithm terminates when no new offers are made [3]. All students on the waiting lists are admitted in the respective programmes, other students are unmatched. The resulting allocation is denoted by μ.

Students consider the likelihood of being accepted into the programmes when applying. Students are distinguished by confidence types τ and estimate their acceptance probabilities by comparing their exam score to the previous year's published cutoff score, some placing greater confidence in this estimate. Specifically, a student of type τ estimates his probability of acceptance to programme p as:

$$\begin{aligned} \Pr\big(\text{accepted to p}\big|s_p + \tau \leq E_i\ \big) &= 1 \\ \Pr\big(\text{accepted to p}\big|s_p - \tau < E_i < s_p + \tau\big) &= \frac{E_i - \big(s_p - \tau\big)}{2\tau} \\ \Pr\big(\text{accepted to p}\big|E_i < s_p - \tau\big) &= 0. \end{aligned} \tag{1.2}$$

As illustrated in Figure 1.1, if student's exam score E_i is considerably greater than the previous year's cutoff score s_p of a programme p then the student is absolutely sure that he will be admitted into p if he applies. If it is much less that s_p then the student believes that he has no chance of being admitted. If E_i is close to the previous year's cutoff score, the student estimates his acceptance probability to be a linear function of his score.

Each student i maximises his expected utility by choosing $a_i \in \operatorname{argmax} \mathbb{E}\, u_i(a_i)$. An *equilibrium* of $\Big(I, P\Big)$ is a set of strategies $a \in \mathcal{P}^I$, such that for each student $i \in I$ and each $a_i, a_i' \in \mathcal{P}$

$$\mathbb{E}\, u_i(a_i) \geq \mathbb{E}\, u_i(a_i').$$

If indifferent between two applications, the student lists his more preferred programme first.

[3] This may happen for two reasons: 1) there were no rejections at the previous step or 2) all students who were rejected at the previous step do not have any programmes left on their applications.

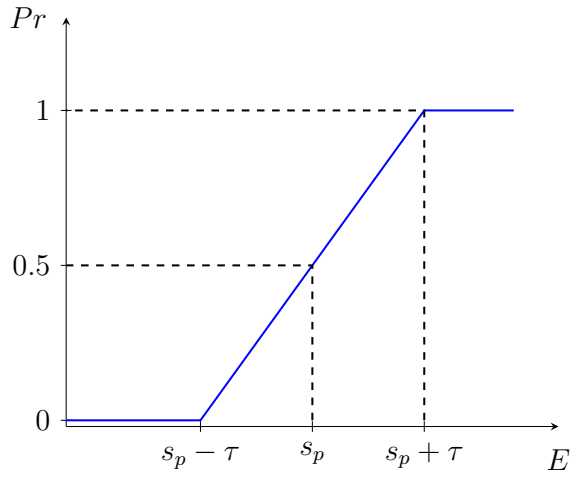


Figure 1.1: Student's estimate of the probability of acceptance into p

1.3 Results

We solve for equilibrium with unrestricted and restricted reports and compare final allocations and utilities.

1.3.1 Unrestricted reports

As a baseline, consider an admission process in which students can apply to both (all) programmes. When the preferences are not truncated, our case is not different from the general one-to-many case. It is known that when all proposing agents have capacity one, DA mechanism is strategyproof for the proposing side [31]. We formulate this well-known result as Proposition 1.

Proposition 1. *When the applications are unrestricted, students truthfully report their full preferences in equilibrium.*

This result is only true under full reports. When reports are restricted, students strategize, hence, the true preferences are not revealed. We are studying how this fact affects the welfare of the students by comparing the cases with restricted reports to the baseline.

In the baseline, the mechanism matches the top $(q_a + q_b)$ students

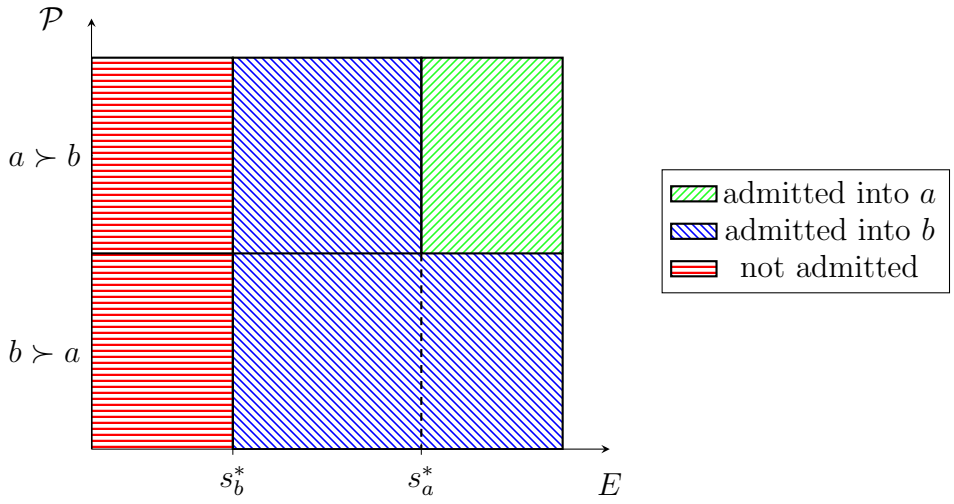


Figure 1.2: Final matching, baseline

to the programmes and leaves the rest unmatched. Denote the cutoff scores that result from the allocation mechanism as s_a^* and s_b^*.

If $s_a^* = s_b^*$, then among top $(q_a + q_b)$ students exactly q_a prefer programme a and q_b prefer programme b, all of them are accepted into their first choice programme. If programme a is more competitive than programme b, that is, if $s_a^* > s_b^*$, programme a is over-demanded. Since the distribution of scores is independent of students' preferences, some of the students, who prefer a to b, are allocated to programme b. The allocation is illustrated in Figure 1.2. The cutoff scores s_a^* and s_b^* solve:

$$\begin{aligned} q_a &= \alpha_a \Big[1 - F(s_a^*)\Big] \\ q_b &= \Big[\alpha_b \big(1 - F(s_a^*)\big) + F(s_a^*) - F(s_b^*)\Big] \end{aligned} \tag{1.3}$$

1.3.2 Restricted reports

Now we consider the case of two programmes where students are constrained to apply to only one programme. Combining the utilities (1.1) and beliefs (1.2) and restricting all students to be of the same type τ, the expected utility $\mathbb{E}\, u_i(a_i)$ of student i's application a_i depending on his exam score E_i is:

$\mathbb{E}\, u_i(a_i)$	$E_i \geq s_{a_i} + \tau$	$s_{a_i} + \tau > E_i \geq s_{a_i} - \tau$	$s_{a_i} - \tau > E_i$
$a_i = t_i$	1	$\frac{E_i - (s_{t_i} - \tau)}{2\tau}$	0
$a_i = b_i$	$\frac{1}{2}$	$\frac{E_i - (s_{b_i} - \tau)}{4\tau}$	0

Based on the expected utilities, we now identify the optimal strategies of students by solving their maximisation problem. If the expected utilities are the same for both applications, a student applies to his preferred programme t_i. This decision is based on hopeful thinking: since there is no difference in expected utility, why not try to apply to a more preferred programme. Other tie-breaking rules include applying to a less competitive programme or randomising. We discuss both in Section 1.4.

Proposition 2. *In equilibrium, students who prefer a more competitive programme apply to their preferred programme when their exam scores are larger than or equal to s_a^e or smaller than or equal to s_b^e and to the other programme if the score is in the interval $\left(s_b^e, s_a^e\right)$. Values of the estimated cutoff scores s_a^e and s_b^e depend on the parameters of the model. Students who prefer a less competitive programme apply to their preferred programme.*

Proof. The proof is in the appendix. □

Figure 1.3 illustrates the optimal strategy of each student[4]. Students who prefer a more competitive programme face a tradeoff between higher payoff of their preferred programme and lower probability of being accepted, while students who prefer a less competitive programme always yield higher expected utility when applying to their preferred programme.

[4]Changing $u_i(b_i)$ from 1/2 to x would impact s_a^e and s_b^e. As a result, the number of students who prefer a but apply to b would be different (the dotted blue area in the top part of the graph, where $a \succ b$, would shift and become wider or narrower depending on the value of x).

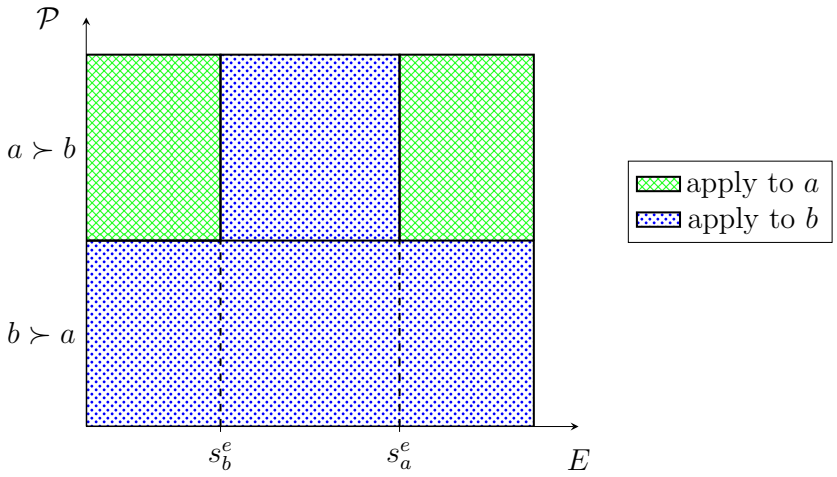


Figure 1.3: Optimal student's strategy

The optimal strategies closely resemble the outcome of the allocation process without truncation of the applications, as shown in Figure 1.2. When $s_a^e = s_a^*$ and $s_b^e = s_b^*$, the outcome of the allocation with restricted reports will not change compared to the baseline case. If, however, the estimated cutoff scores are different, then the total utility of students may increase or decrease. We examine such outcomes below.

Corollary 1. *The values of the estimated cutoff scores* s_a^e *and* s_b^e *are:*

$(a) s_b \leq s_a - \tau$	(b) $s_a - \tau \leq s_b \leq s_a$
$s_a^e = s_a$	$s_a^e = 2s_a - s_b - \tau$
$s_b^e = s_b - \tau$	$s_b^e = s_b - \tau$

Proof. The proof is in the appendix. □

1.3.3 Welfare

We measure students' welfare as the total utility of students that results from the allocation process[5]. Using the students' optimal strategies, we

[5]Welfare of the students depends on the values they assign to each programme conditional on being admitted. Hence, it changes not only with a change in allocation, but also with a change in the utility function. If $u_i(b_i) = x$, both the utility function and the allocation are affected. Given all possible preference profiles, the frequency of increases and decreases in total utility due to reports' truncation may change too.

compare the total utilities of the allocation with restricted applications to the welfare in the baseline case. We compare the programmes according to their cutoff scores in the baseline case and call a programme with a higher cutoff score more competitive.

Proposition 3. *1) If students who prefer a more competitive programme a overestimate their probability of admission into a, total utility increases. 2) If students who prefer a more competitive programme a underestimate their probability of admission into a and overestimate the probability of admission into b, total utility decreases. 3) If students who prefer a more competitive programme a underestimate their probability of admission into both a and b, the total utility comparison is ambiguous.*

Proof. The proof is in the appendix. □

Proposition 3 shows that the total utility can both increase and decrease. Ambiguity in the third case is caused by the fact that we do not consider any specific values of the parameters such as quotas and students' type and previous year's cutoff scores. When the values are known, we can draw an unequivocal conclusion regarding the change in welfare. In the propositions in this paper, we call the total utility comparison ambiguous if it depends on the values of the parameters.

Case 1. Students who prefer a more popular programme a overestimate their probability of admission to a, total utility increases.

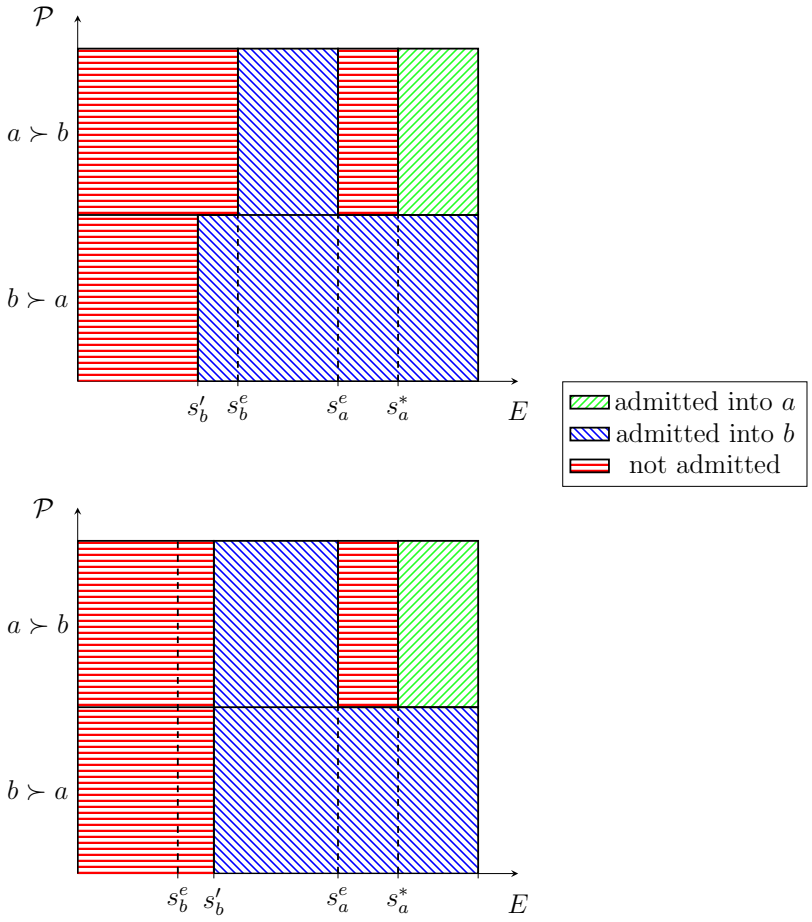


Figure 1.4: Students' allocation: case 1 if $s^e_b > s'_b$ (top) and $s^e_b < s'_b$ (bottom)

Restricting the reports changes the cutoff scores that result from the allocation process from s^*_a and s^*_b to s'_a and s'_b. Because of the misestimation, students who would be admitted into b in a baseline case applied to a and were left unmatched. Their places are allocated among lower-scoring students who applied to b, lowering the cutoff score of a less competitive programme to $s'_b < s^*_b$, while $s'_a = s^*_a$ remained the same. The allocation of students to programmes that follows from their optimal strategies is illustrated in Figure 1.4. Denote the smallest exam score of a students who prefers a and is admitted into b as $s''_b = \max\{s'_b, s^e_b\}$. The cutoff scores s^*_a and s'_b solve:

$$q_a = \alpha_a \Big[1 - F(s^*_a)\Big]$$
$$q_b = \alpha_a \Big[F(s^e_a) - F(s''_b)\Big] + \alpha_b \Big[1 - F(s'_b)\Big]$$

Case 2. Students who prefer a more popular programme a underestimate their probability of admission to a and overestimate the probability of admission into b, total utility decreases.

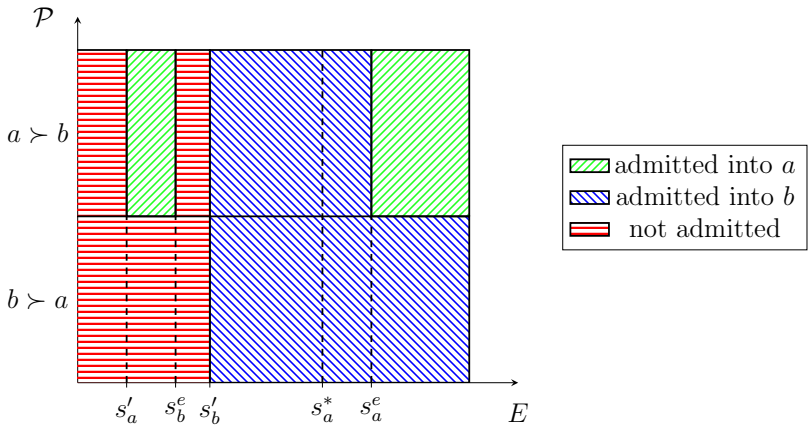


Figure 1.5: Students' allocation: case 2

Some of the students who would be admitted into a in a baseline case applied to b. Their places are allocated to applicants whose scores are below s^e_b, they applied to their preferred programme as they estimated their chances to be accepted into either of the programmes as zero. The final allocation in shown in Figure 1.5, the cutoffs solve:

$$q_a = \alpha_a \Big[1 - F(s^e_a) + F(s^e_b) - F(s'_a)\Big]$$
$$q_b = \alpha_a \Big[F(s^e_a) - F(s''_b)\Big] + \alpha_b \Big[1 - F(s'_b)\Big]$$

1.3.4 Discussion

Intuition suggests that allowing more applications increases total utility and eliminates justified envy. Surprisingly, the conclusion depends on students' beliefs.

Restricting applications decreases the total utility when high-scoring students who prefer a more popular programme believe that they have a low chance of admission and apply to their safe choice instead. As a results, too many students who prefer a more competitive programme

are assigned to their least preferred programme, decreasing the total utility and leaving seats in the more competitive programme unfilled. The unfilled seats are reallocated to low-scoring students with the same preferences. Restricting applications may also increase the total utility, but it does so by allowing low-scoring students to replace high scoring students with different preferences, who are now unassigned. Both allocations are not envy-free, moreover, the first one yield lower total utility compared to the baseline.

1.4 Extensions of the model

1.4.1 Application assumptions

In our models we assumed that if the expected utilities are the same for both applications, a student applies to his preferred programme t_i. The expected utilities are the same at the estimated cutoff s_a^e and on the interval of low exam scores in $[0, s_b^e]$ where $\mathbb{E}\, u_i(t_i) = \mathbb{E}\, u_i(b_i) = 0$. The latter implies that the students apply to their top choice when believing that they will not be admitted. However, they might prefer to apply to the least competitive programme or randomise.

Applying to a less competitive programme

If students apply to b when $\mathbb{E}\, u_i(t_i) = \mathbb{E}\, u_i(b_i) = 0$, the qualitative results remain the same. The increase and decrease in the total utility still follow Proposition 3.

When students who prefer a overestimate their probability of admission to a, the new allocation is as described by Figure 1.4 (bottom) and the total utility increases. Figure 1.4 (top) no longer describes the allocation; since all the students apply to b when their exam score is below s_a^e, s_b^e is effectively equal to zero.

When students who prefer a underestimate their chances of being admitted into a, the effect is the same as given the initial assumptions except the seats in a that are not matched to high-scoring students because of the misestimation now remain unfilled. This causes an additional decrease in total utility.

Randomising

When students randomise, they apply to programme a with probability β_i and to programme b with complementary probability $1 - \beta_i$, where $\beta_i \in (0, 1)$ for all i. The parameter β_i may be different for some or all the students and is independent of the exam scores.

Proposition 4. *If students randomise between applying to a and applying to b when* $\mathbb{E}\, u_i(t_i) = \mathbb{E}\, u_i(b_i) = 0$, *results of Proposition 3 do not change.*

Proof. The proof is in the appendix. □

1.4.2 Student types

Our model assumes that all students have the same type τ. We relax this assumption by allowing two types and continuum types.

Two types of students

Assuming two types of students τ_1 and τ_2 increases the number of estimated cutoff scores to four, $s^e_{a,1}$, $s^e_{a,2}$, $s^e_{b,1}$ and $s^e_{b,2}$, two for each type. The values of $s^e_{a,1}$, $s^e_{a,2}$, $s^e_{b,1}$ and $s^e_{b,2}$ are the same as ones described in Corollary 1 given student's type.

Proposition 5. *1) If students who prefer a more popular programme a of both types overestimate their probability of admission into a, total utility increases. 2) If students who prefer a more competitive programme*

a of both types underestimate their probability of admission into a and overestimate the probability of admission into b, total utility decreases. 3) If students who prefer a more competitive programme a of one or both types underestimate their probability of admission into both a and b, the total utility comparison is ambiguous.

Proof. The proof is in the appendix. □

Continuum types of students

Now we assume that students' types τ_i are drawn from a distribution with cdf $G(\tau)$, $\tau \in [\tau_{min}, \tau_{max}]$. The estimated cutoff scores $s^e_{a,i}$ and $s^e_{b,i}$ are student specific. In a case of unrestricted reports, they solve the same system of equations (1.3) as in the initial case with only one type; students apply to both programmes, thus, distribution of types has no effect on the resulting allocation.

When reports are restricted, for each student the optimal strategy is determined as described in Proposition 2. However, the total utility outcomes depend on the parameters and realisations of the random variables; we plan to estimate the total utility in simulations rather than pursue further theoretical results.

1.4.3 Three programmes

With 3 programmes a, b and c, students have six possible preferences $\{(abc), (acb), (bac), (bca), (cab), (cba)\}$. Denote by α_{xy} the ratio of students who prefer x to y and y to z, by $\alpha_{x.}$ the ratio of students whose first choice is x. Also denote student i's first, second and third ranked

programmes by t_i, m_i and b_i . Students' utilities are:

$$u_i(p) = \begin{cases} 1 & \text{if admitted into } p = t_i \\ 2/3 & \text{if admitted into } p = m_i \\ 1/3 & \text{if admitted into } p = b_i \\ 0 & \text{if not admitted} \end{cases} \tag{1.4}$$

The previous year's cutoff scores are s_a, s_b and s_c, all students are of the same type τ. To extend the algorithm we repeat *Step 2* by updating the waiting lists until there are no new applications.

The utility function 1.4, similarly to it's two-programme counterpart, is linear in the programme's ranking on each student's preference list. Similarly, the valuations of the middle and bottom programmes m_i and b_i would be different in a case with non-linear utility. We could assume that they follow a different utility function or that they are arbitrary. These assumptions are a subject of future research.

Unrestricted reports

As a baseline, consider a process with unrestricted applications. Denote the cutoff scores that result from the allocation process as s_a^*, s_b^* and s_c^*. Proposition 1 holds for any number of programmes, so the students report their preferences truthfully.

The final matching is shown in Figure 1.6 and the cutoff scores s_a^*, s_b^* and s_c^* solve:

$$q_a = \alpha_{a\cdot}\Big[1 - F(s_a^*)\Big]$$
$$q_b = \alpha_{b\cdot}\Big[1 - F(s_b^*)\Big] + \alpha_{ab}\Big[F(s_a^*) - F(s_b^*)\Big]$$
$$q_c = \alpha_{c\cdot}\Big[1 - F(s_c^*))\Big] + (\alpha_{ab} + \alpha_{b\cdot})\Big[F(s_b^*) - F(s_c^*)\Big] + \alpha_{ac}\Big[F(s_a^*) - F(s_c^*)\Big]$$

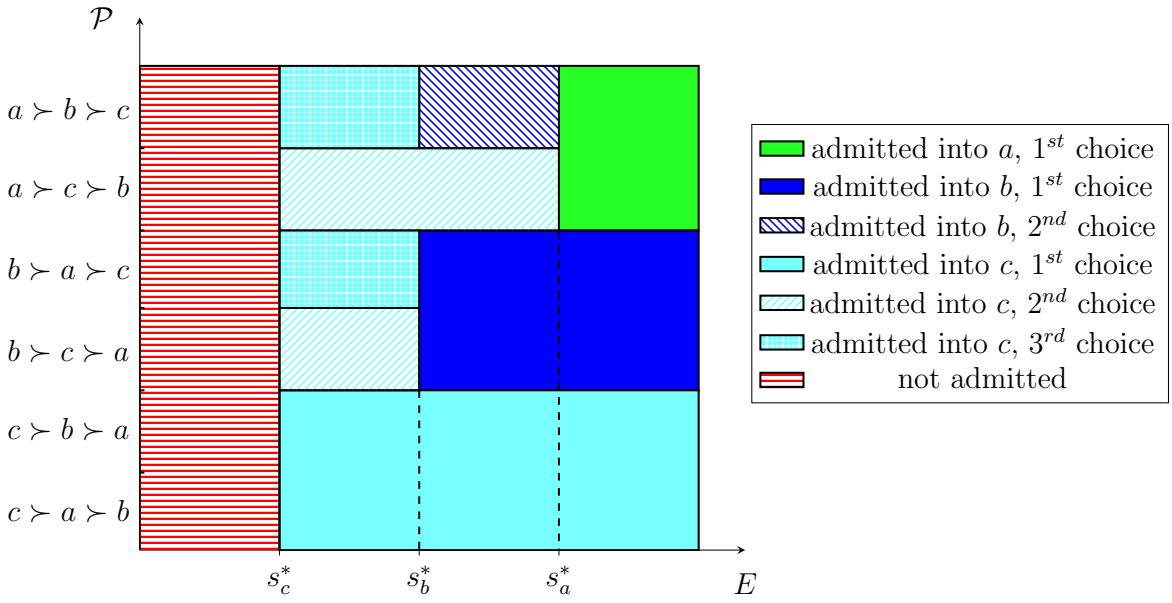


Figure 1.6: Final matching with unrestricted applications

Reports restricted to 1 programme

Now assume that students can only apply to one out of the three programmes. Combining the utilities (1.4) and beliefs (1.2), the expected utility $\mathbb{E}\, u_i(a_i)$ of student i's application a_i depending on his exam score E_i is shown in Table 1.1.

$\mathbb{E}\, u_i(a_i)$	$E_i \geq s_{a_i} + \tau$	$s_{a_i} + \tau > E_i \geq s_{a_i} - \tau$	$s_{a_i} - \tau > E_i$
$a_i = t_i$	1	$\dfrac{E_i - (s_{t_i} - \tau)}{2\tau}$	0
$a_i = m_i$	$\dfrac{2}{3}$	$\dfrac{E_i - (s_{m_i} - \tau)}{3\tau}$	0
$a_i = b_i$	$\dfrac{1}{3}$	$\dfrac{E_i - (s_{b_i} - \tau)}{6\tau}$	0

Table 1.1: Expected utility of an application a_i of student i

A student optimally applies to a programme that yields the maximum expected utility. After estimating cutoff scores with three programmes, there are either five or six cutoff scores. In the scenario depicted in Figure 1.8, we show that the total utility may increase as a result of restricting the applications. The functional form the cutoff score s^e estimated by a student i depends on whether he is deciding between t_i

and m_i, t_i and b_i or m_i and b_i as well as whether the student expects guaranteed alternative. For instance, s_1^e (see Figure 1.8) results from comparing programmes $t_i = a$ and $m_i = b$ by a student who prefers $a \succ b \succ c$ and is not sure about being admitted into ether of the programmes. Cutoff score s_4^e is the lowest score of a student who prefers $a \succ c \succ b$ at which he is willing forgo guaranteed admittance to m_i for a chance at t_i. Although both compare first and second choices, estimated cutoffs differ.

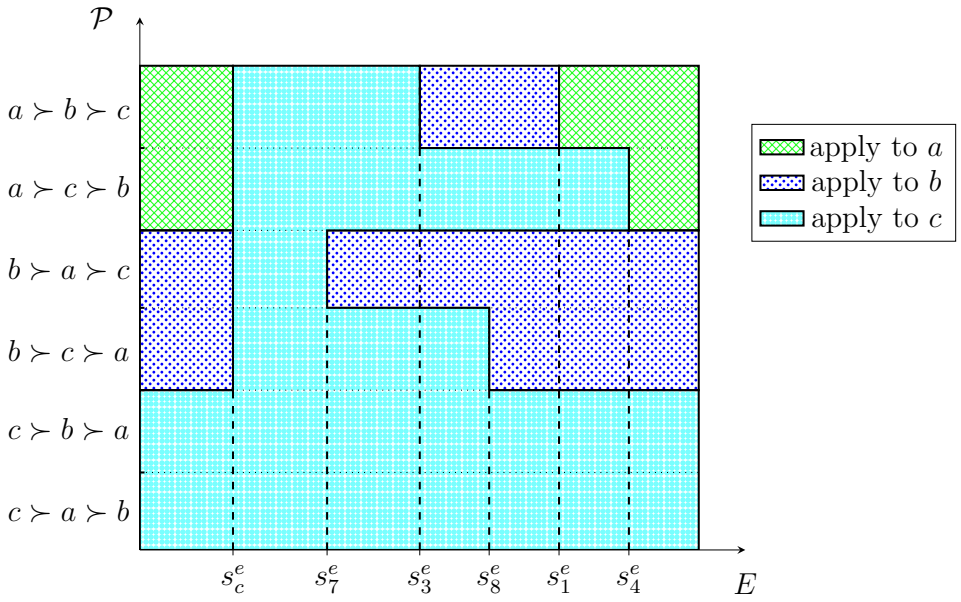
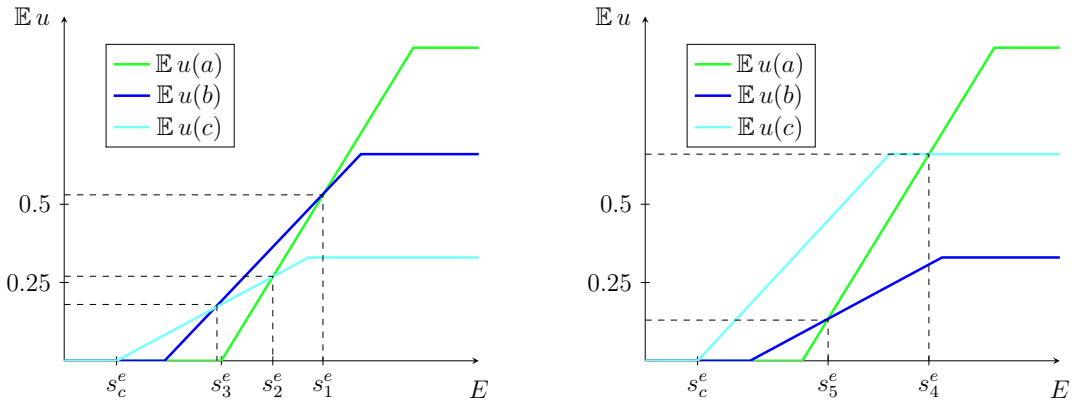


Figure 1.7: Optimal student's strategy

Proposition 6. *In equilibrium with three programmes, students with each preference apply to one programme according to their estimated cutoffs as shown in Figure 1.7.*

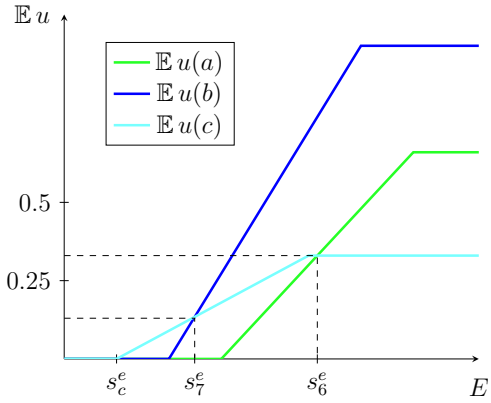
Proof. The proof is in the appendix. □

Figure 1.7 illustrates the optimal strategy for each student.

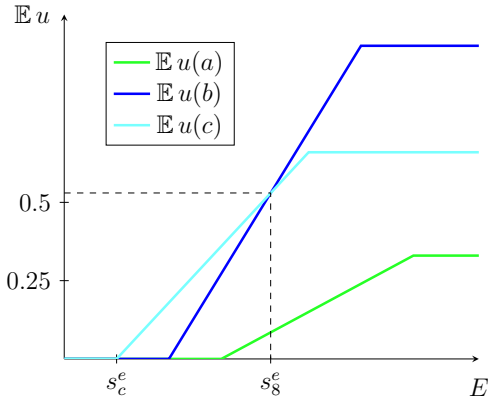


(a) $a \succ b \succ c$

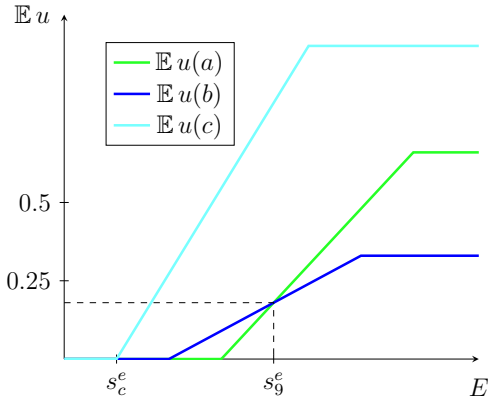
(b) $a \succ c \succ b$



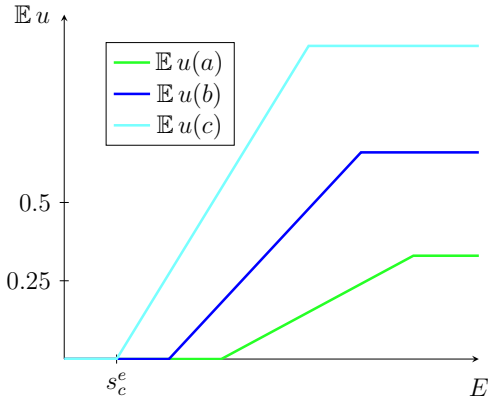
(c) $b \succ a \succ c$



(d) $b \succ c \succ a$



(e) $c \succ a \succ b$



(f) $c \succ b \succ a$

Figure 1.8: Student's expected utility comparison depending on the preference profile

Reports restricted to 2 programmes

When reports are restricted to two programmes, the application a_i of a student i is $a_i = (t_i', m_i')$, possibly different from (t_i, m_i). A student may be admitted to t_i', m_i' or remain unmatched. The probability of being rejected by t_i' is $Pr(\text{rejected by } t_i'|E_i) = 1 - Pr(\text{accepted to } t_i'|E_i) =$

$1 - Pr(t'_i|E_i)$. The expected utility of an application a_i of a student i is:

$$\mathbb{E}\,u(a_i) = \mathbb{E}\,u(t'_i, m'_i) = Pr(t'_i|E_i) \cdot u_i(t'_i) + (1 - Pr(t'_i|E_i)) \cdot Pr(m'_i) \cdot u_i(m'_i)$$

Students can can maximise $\mathbb{E}\,u$ using the Marginal Improvement Algorithm (MIA) [17]. MIA starts with an empty application list. At each step, it adds a programme that increases $\mathbb{E}\,u(a_i)$ by the largest amount among those not yet selected to the list. Once the two programmes that form an optimal application are selected, each student submits the report ordering the programmes in his preference order.

Proposition 7. *In equilibrium with three programmes and two applications, students apply to programmes as shown in Figure 1.9 listing the programmes in the preference order.*

Proof. The proof is in the appendix. □

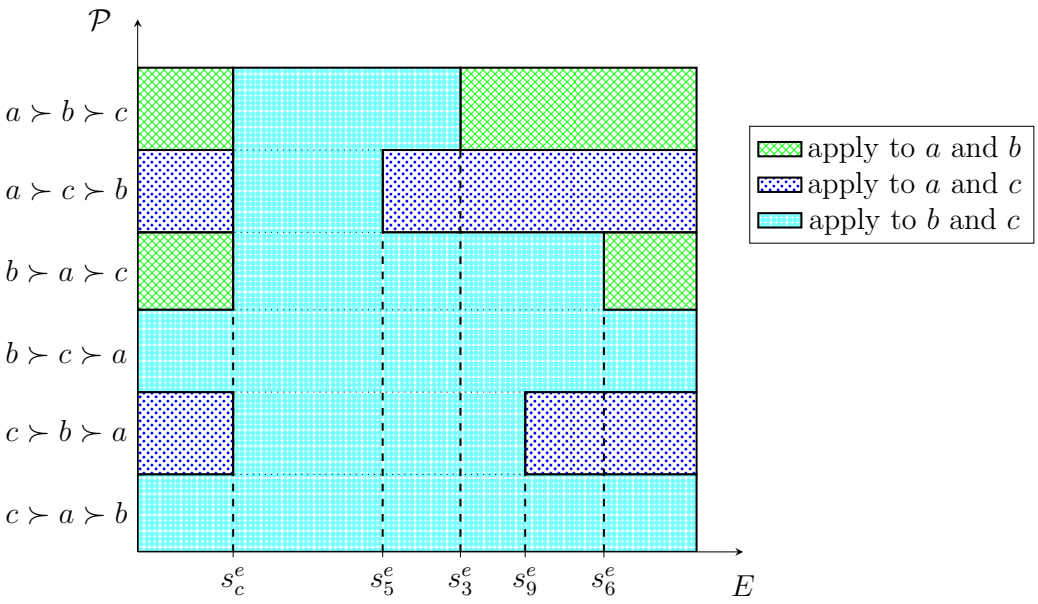


Figure 1.9: Optimal student's strategy

1.4.4 Welfare

Based on the students' optimal strategies, we show some examples of increases in the total utility with restricted applications.

Example 1. ***Reports restricted to one programme.*** *If the students who prefer $a \succ b \succ c$ overestimate their probability of admission into a while the students who prefer $a \succ c \succ b$ do not make a mistake, and students who prefer b to c overestimate their probability of admission into b, the total utility increases.*

As can be seen in Figure 1.10, some high scoring students were not allocated to any programmes. In the baseline, students who prefer $a \succ b \succ c$ and have the exam score in the interval $[s_b^*, s_a^*)$ are allocated to programme b. They, however, applied to a and were not admitted into any programme leaving some spaces in b vacant and lowering the actual cutoff score from s_b^* to s_b'. Those spaces were reallocated to students with the same preference and those who prefer b to any other programme, increasing the total utility. Other unmatched students in the baseline are allocated to c, their least preferred programme. Their seats now belong to students who rank c not only third but also first and second, further increasing total utility.

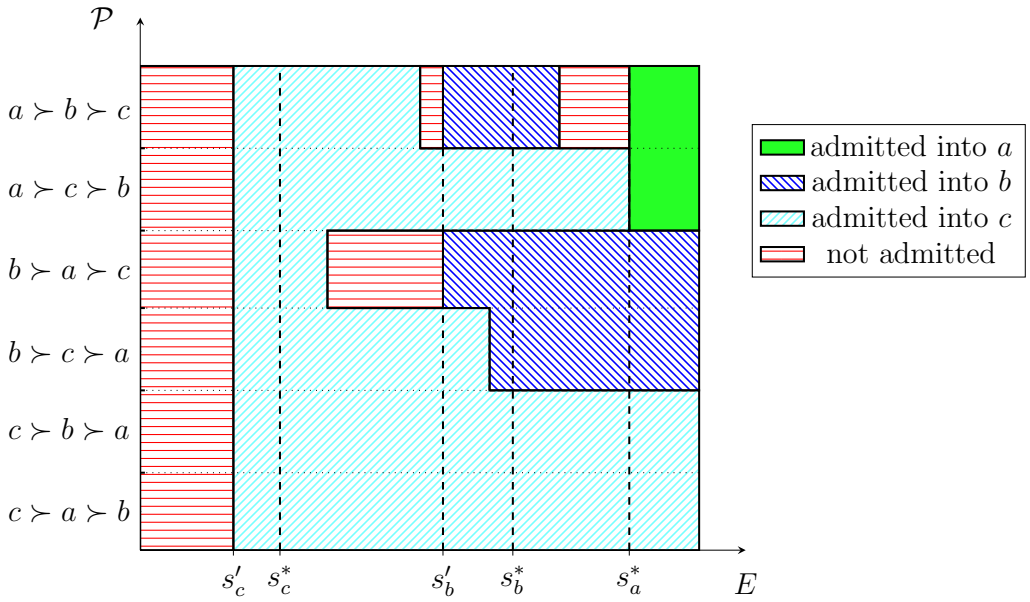


Figure 1.10: Example of an increase in total utility (application to one programme)

Example 2. ***Reports restricted to two programmes.*** *If the students who prefer $a \succ b \succ c$ overestimate their probability of admission*

into a and b, the total utility increases.

Overestimation leads to some high-scoring students that prefer $a \succ b \succ c$ to be unmatched (Figure 1.11). Other students misestimate the probability of admission into a and b as well, but it has no effect on their final allocation as the second programme on the report compensates for the mistake. As a result, high-scoring students that are assigned to their least preferred programme in the baseline case are now unmatched, and the seats are reallocated to students with different preferences. Depending on the accuracy of the estimations of s_c^* the seats may be matched to either students with all types of preferences (as shown in Figure 1.11) or only to those students who do not rank it last in their preferences (this happens if students' estimated utility is zero and they simply apply to their two most preferred programmes). As a result of reallocation the total utility increases.

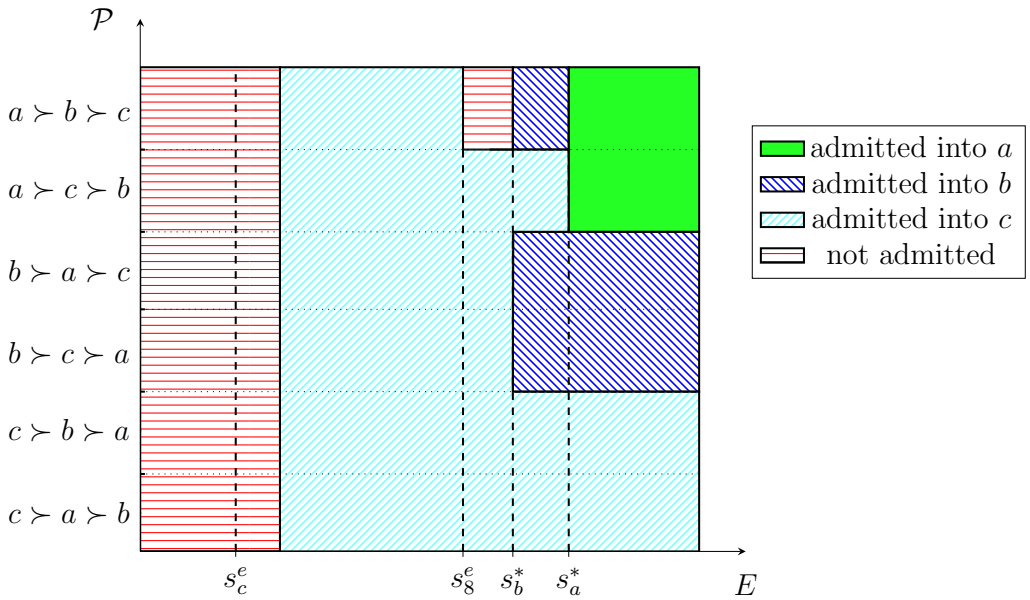


Figure 1.11: Example of an increase in total utility (application to two programmes)

The examples show that our main result holds in a case of three programmes. In other situations, the total utility may also decrease or remain the same.

1.5 Numerical Experiment

In order to verify the theoretical results and to show the effects of the parameters of the model, we simulate the data. Like in the theoretical analysis, we compare the allocation under restricted reports to a baseline case when students submit their full preferences. If student's allocation changes as a result of truncating his reports, we call such student *reallocated*. We investigate the change in the total utility and the number of *reallocations*, that is, the number of students who are reallocated.

To simulate the data and the allocation process we need to make assumptions on the following:

- The number of the students
- The number of the programmes
- Programmes' capacities
- Distribution of the students' exam scores
- Students' preferences
- Programmes' cutoff scores

According to the Universities and Colleges Admissions Service (UCAS) [43], the organisation that processes most[6] of the applications for British universities, in the years 2006-2018, roughly $70\% - 77\%$ of applicants were accepted to full-time undergraduate programmes. Accordingly, we assume capacities that result in roughly $60\% - 80\%$ admission rate. We also investigate a case with considerable lower admissions of 20%. Analysing the SAT scores in 2019 [9] and SAT Subject Tests' scores in

[6]Around a third of undergraduate full-time education admission in Scotland happens outside of the UCAS, which amounts to roughly $2 - 3\%$ of all applications. "For people living in England, Wales, and Northern Ireland, UCAS covers the overwhelming majority of full-time undergraduate provision." [43]

2017-2019 [8] using Shapiro-Wilk test, we find that the SAT test scores are normally distributed, however, out of nine SAT Subject Tests only the scores in the U.S. History form the normal distribution. We hence consider two distributions of students' scores: uniform and truncated normal. Normal or close to normal distribution would be expected when an exam the students are passing is compulsory. It is truncated because there is a minimum and a maximum possible score. Due to self-selection bias[7], however, the distribution may resemble uniform or even have increasing density. The uniform distribution has only two parameters, min and max values, which arise naturally. Where self-selection bias is not present, normal distribution of scores is assumed.

In our numerical experiments, we assign 1000 students to two and then to four colleges. Programmes' capacities are chosen to result is 20% and $60-80\%$ admission. Students' exams are distributed randomly between 0 and 100. We use two different distributions of scores as discussed above. Students' preferences are also generated randomly assuming that they are equally likely to prefer each of the colleges. In a simulation, previous year's cutoff scores are unknown, so we set them using the cutoffs that arise from the baseline case for the current year s_a^* and s_b^*. We also consider distorted previous year's cutoffs.

Two programmes

We start by analysing the case with two colleges. We verify that the theoretical findings in Proposition 3 can be confirmed numerically. We start by assuming that the cutoff scores of the programmes in the previous year were the same as this year's s_a^* and s_b^*. According to Corollary 1, when maximising their expected utility, students will either estimate the cutoff scores correctly or underestimate them. In the former case

[7]Where only the average score matters or exam is required by the programme, the students choose to pass exams in subjects they are good at, which affects the distribution of grades.

students' welfare will not be affected, in the latter, according to Proposition 3, the total utility will increase since students overestimate their admission probabilities. As student's type τ increases, the students become more confident, which further increases the total utility.

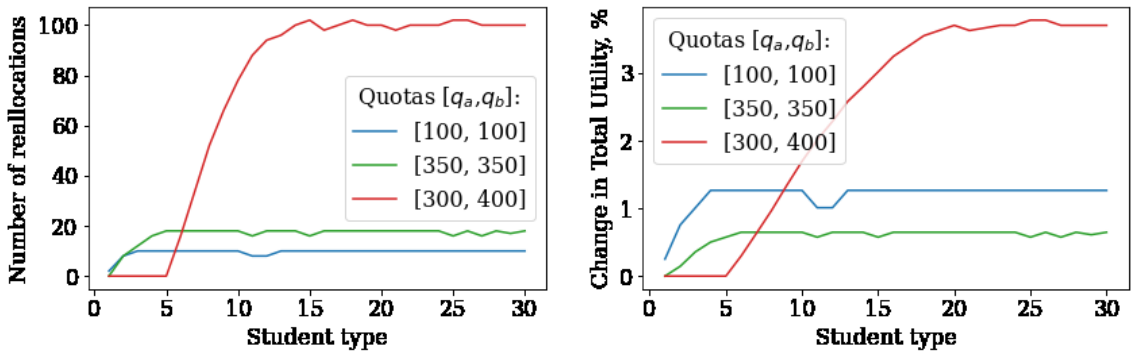


Figure 1.12: Number of reallocations and change in total utility with normally distributed exam scores

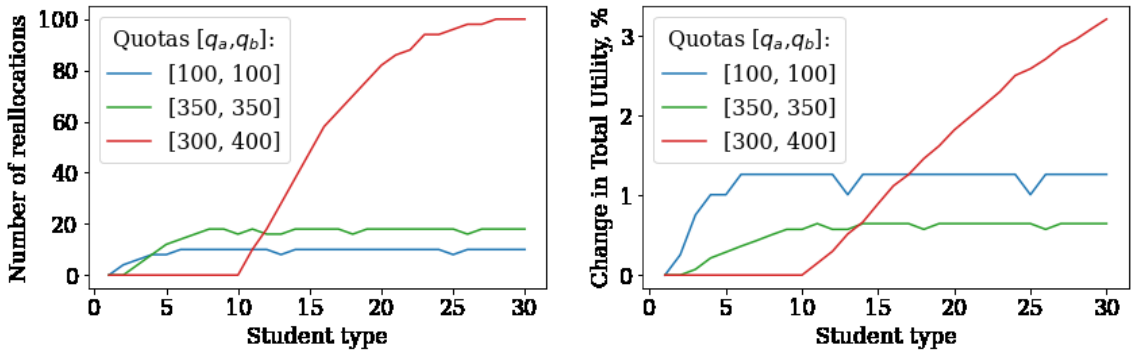


Figure 1.13: Number of reallocations and change in total utility with uniformly distributed exam scores

Figures 1.12 and 1.13 depict the result of the simulations. The number of reallocations (on the left-hand side) shows a number of students who as a result of the preference lists' truncation are allocated a different programme compared to the baseline case. The change in total utility (on the right-hand side) demonstrates the corresponding percentage change in utility. When quotas are the same, the cutoff scores

are very close to each other[8]. This means that an increase in the type will increase the total utility almost immediately. When the quotas are different, so are the baseline cutoff scores. The cutoff scores will be misestimated when the type is big enough. The graphs support our theoretical findings for the first case of Proposition 3. We now turn to the second case.

Our model predicts a decrease in total utility when the cutoff score of the more competitive programme is overestimated. To simulate this scenario, we assume that the previous year's cutoff scores for this programme exceeded the current value s_a^*. We observe that until the type of students τ reaches a certain value, the change in total utility is negative. After reaching the value of approximately 20, the change becomes positive. At this type, the misestimation corrects for the difference in the last year's and current baseline cutoffs.

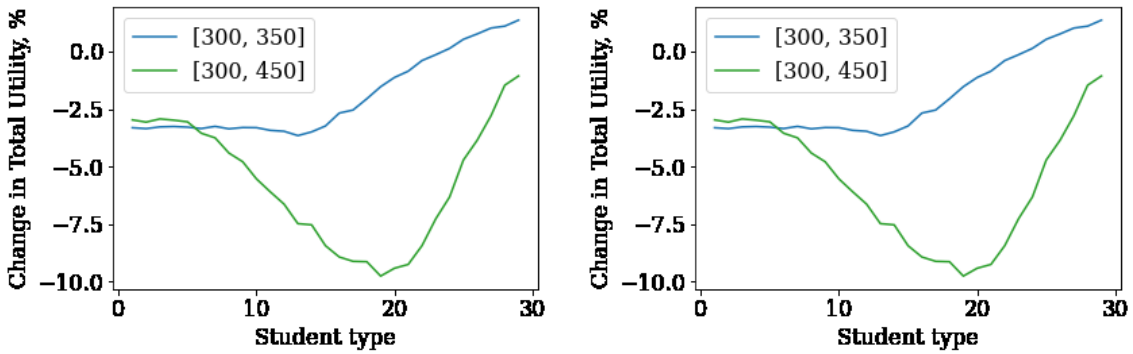


Figure 1.14: Number of reallocations and change in total utility with normally distributed exam scores and distorted previous year's cutoff scores

The results of the simulations for the case with two programmes are in line with theoretical results of our model, so we now consider an example with more colleges.

[8]They may not be identical as the simulation is randomised. When the students are simulated to prefer each programme with equal probability, we may see, for example, 503 students who prefer a and 497 who prefer b.

Four programmes

Our theoretical model only discusses the outcomes of allocation processes in presence of two or three colleges. We simulate the admission process with four colleges, all of which are equally likely to be preferred to other ones. Our goal is to check that the total utility may still increase as a result of report's truncation. We choose two sets of quotas: one makes the process symmetric ($q_a = q_b = q_c = q_d = 200$), and the other makes the first two colleges more competitive ($q_a = q_b = 100, q_c = 200, q_d = 300$). In both cases the total number of seats is 800 per 1000 students, which means that 80% of all students are admitted. We assume that the previous year's cutoff scores are the same as the baseline ones in the current year, hence, we expect the total utility to increase (at least for small values of τ). This is exactly what we observe in Figure 1.15. We also notice that as τ increases, the change in the total utility starts to decline and eventually reaches negative values. This leads us to believe that as the misestimations increase, some of them negate the effect of others.

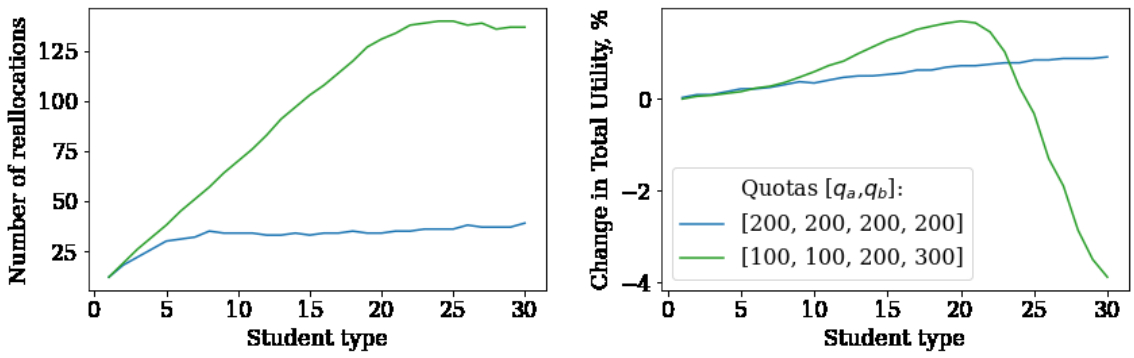


Figure 1.15: Number of reallocations and change in total utility with uniformly distributed exam scores, four colleges

In this paper, we run the experiments to check that the theoretical and numerical results coincide. As the number of colleges increases, analysing results theoretically becomes unfeasible. For large number of

colleges we plan to devise simulations that will describe how the number of allocations and the total utility respond to changes in the parameters of the model.

1.6 Conclusion

We show that, contrary to expectation, restricting the length of the applications may increase welfare when students judge their admission probability heuristically. When high-scoring students misestimate their admission probabilities, their seats may be reassigned to lower-scoring students who obtain higher utility from those seats. We also observe an increase in welfare when students choose among two or three programmes. Ability to apply to two programmes out of three sometimes results in the same allocation as the baseline case but other times increases or decreases total utility. We run some numerical experiments and find that the theoretical findings are in line with the simulated outcomes.

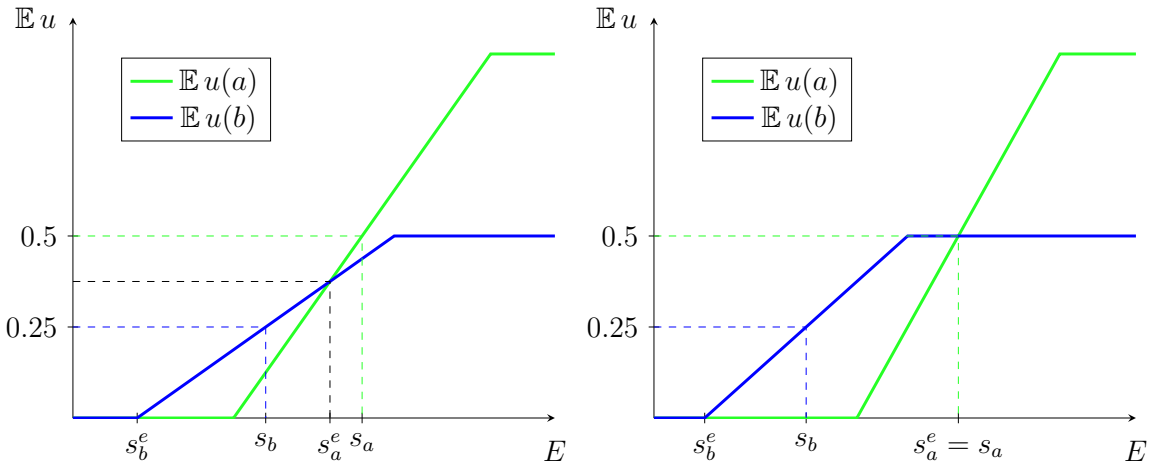
Our model allows for several extensions. First, the utility function may be non-linear. The evaluations of the programmes may follow a different function or be arbitrary. The utilities of programmes that are ranked the same by different students may not be same. Second, the number of programmed may be larger. It would be interesting to analyse how the total utility changes depending on the fraction of programmes that can be included in the application. Third, students' exam scores may be correlated with preferences. More popular programmes usually provide better quality of education. However, students with lower exam scores may find it too hard to study in such programmes and prefer other programmes instead. The change in computations of welfare and further comparisons, which would significantly complicate theoretical analysis, is unlikely to affect the qualitative results. Additionally, students may

be indifferent among some programmes. Finally, the admission process can be decentralised or based on a different allocation procedure.

1.7 Appendix. Proofs of propositions

Proposition 2. *In equilibrium, students who prefer a more competitive programme apply to their preferred programme when their exam scores are larger than or equal to s_a^e or smaller than or equal to s_b^e and to the other programme if the score is in the interval $\left(s_b^e, s_a^e\right)$. Values of the estimated cutoff scores s_a^e and s_b^e depend on the parameters of the model. Students who prefer a less competitive programme apply to their preferred programme.*

Proof. Students are determining the optimal strategy by maximising the expected utility of the application. The expected utility of students who prefer a more competitive programme is shown in Figure 1.16. There are only two cases as, by assumption, $s_a > s_b$. When student's exam score is above s_a^e application to programme a yields higher expected utility than application to programme b, the opposite is true when the score is below s_a^e and above s_b^e. Finally, if the student's exam score is below s_b^e or is equal to s_a^e or s_b^e, then he is indifferent between applying to both programmes and, by assumption, applies to his preferred programme a.



(a) $s_a - \tau < s_b < s_a$ (b) $s_b < s_a - \tau$

Figure 1.16: Student's expected utility comparison, student prefers programme a

Students who prefer a less competitive programme b always have weakly larger expected utility when applying to their most preferred programme, thus, they always apply to b . The expected utilities are depicted in Figure 1.17.

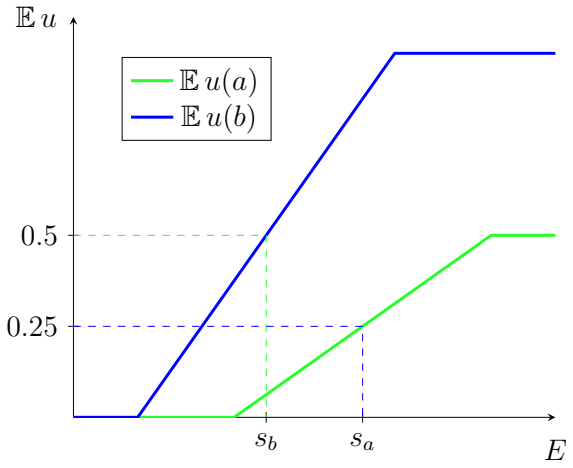


Figure 1.17: Student's expected utility comparison, student prefers programme b

□

Corollary 1.

Proof. Student that prefer programme a apply to programme a if:

	$(a) s_b \leq s_a - \tau$	(b) $s_a - \tau \leq s_b \leq s_a$
$\mathbb{E}\,u_i(a) > Eu_i(b)$	$\frac{E_i-(s_a-\tau)}{2\tau} \geq \frac{E_i-(s_b-\tau)}{4\tau}$ $2E_i - 2s_a + 2\tau \geq E_i - s_b + \tau$ $E_i \geq 2s_a - s_b - \tau$ Thus, $s_a^e = 2s_a - s_b - \tau$	$\frac{E_i-(s_a-\tau)}{2\tau} \geq \frac{1}{2}$ $E_i - s_a + \tau \geq \tau$ $E_i \geq s_a$ Thus, $s_a^e = s_a$
$\mathbb{E}\,u_i(a) = \mathbb{E}\,u_i(b) = 0$	$E_i \leq s_b - t$ Thus, $s_b^e = s_b - t$	

□

Proposition 3. *1) If students who prefer a more competitive programme a overestimate their probability of admission into a, total utility increases. 2) If students who prefer a more competitive programme a underestimate their probability of admission into a and overestimate the*

probability of admission into b, total utility decreases. 3) If students who prefer a more competitive programme a underestimate their probability of admission into both a and b, the total utility comparison is ambiguous.

Proof. Students are allocated to programmes a and b. Students who prefer b only apply to their first preference, thus, only students who prefer a are allocated to a. They each yield utility $u_i = 1$. It follows that the changes in total utility result from differences in final allocations to programme b, namely, on the ratio of students who prefer b among those admitted to b.

1) The seats in b are reallocated from students who prefer a with exam scores $Sc \in [s_a^e, s_a^*]$ and $Sc \in [s_b^*, s_b'']$ (if $s_b^* < s_b''$) to students who prefer b. Thus, the utility of students either remains the same or, as a result of reallocation, decreased from 1/2 to 0 for one student and increases from 0 to 1 for another, therefore, increasing the total utility.

2) All students with scores in $[s_b', s_a^e]$ are accepted into b as well as the ones that prefer b and have higher scores. If $s_b' \leq s_b^*$ then more students would be admitted into b than in the baseline case as $s_a^e > s_a^*$, but the capacities did not change. It follows, that $s_b' > s_b^*$ and among students matched to b there are now less students who prefer b, thus, the total utility decreases.

3) As students apply to their favourite programme when the expected utilities of applying to each programme are the same, only student who prefer a are admitted into a, therefore, the fraction of students who prefer programme a among the students admitted into b needs to be compared to the baseline case. The allocation with larger number of such student will result in a lower total utility since students who prefer a do not yield as much utility from being admitted into b as students who prefer b.

In the baseline case, there are $St_{baseline} = \big(F(s_a^*) - F(s_b^*)\big)\alpha_a$ students

who prefer a and are admitted into b, in the restricted reports' case there are $St_{restricted} = \big(F(s_a^e) - F(s_b^e)\big)\alpha_a$ of them. If $St_{baseline} < St_{restricted}$, the welfare increases and if $St_{baseline} > St_{restricted}$, the welfare decreases. Note that when students who prefer a more competitive programme a underestimate their probability of admission into both a and b, $s_a^e > s_a^*$ and $s_b^e > s_b*$. This means that the estimated cutoff scores with truncated reports are greater than in the baseline case, and every student who prefers a but applies to b will be admitted into b.

If the distribution of scores was uniform, comparing the distances between the cutoffs would be enough to arrive at an unequivocal conclusion. If $s_a^* - s_b^* = s_a^e - s_b^e$, $St_{baseline}$ would be equal to $St_{restricted}$, having no effect on welfare. $s_a^* - s_b* > s_a^e - s_b^e$ would imply that more students who prefer a are admitted to b in the baseline case, thus, the total utility would increase with emergence of restrictions. The opposite would be true if the sign changed to the opposite strict inequality.

The distribution of exam scores, however, is not necessarily uniform. Due to self-selection bias[9] and depending of the difficulty of any specific exam, distribution of scores may be unpredictable. We can determine the welfare implications by computing $St_{baseline}$ and $St_{restricted}$ and comparing them to each other. Theoretically, as a result of restricting reports, both increase and decrease in total utility are possible in this case.

□

Proposition 4. *If students randomise between applying to a and applying to b when* $\mathbb{E}\, u_i(t_i) = \mathbb{E}\, u_i(b_i) = 0$, *results of Proposition 3 do not change.*

Proof. 1) An increase in total utility is due to reallocation of spaces

[9] Apart from compulsory exams, students tend to choose to pass the exams in the subjects they are good at. This leads to various distribution of scores.

resulting from the misestimation of the cutoff scores. High-scoring students, who prefer a and could be admitted into b, apply to a and remain unmatched. The now vacant seats in b are given to lower-scoring students who applied to b. As long as students who prefer b receive some of these seats, the total utility will increase as a result of the reallocation. Since $\beta_i \in (0, 1)$ for all students i, students who prefer b will apply to it with positive probability and receive some of the seats. Thus, the utility will still increase.

Note that the total utility cannot decrease in this case. In the worst scenario where all the seats are reallocated to students who prefer a, the total utility remains unchanged. This scenario, however, is not allowed as $\beta_i \in (0, 1)$.

2) A decrease in total utility is a result of high-scoring students who prefer and could be admitted into a applying and being admitted into b, replacing students who prefer b and, thus, reducing the utility yielded from the reallocated seats. The seats in a that are vacant because of the mistake in applying are matched to lower-scoring students who applied to a. Under the initial assumption, all the seats were given to students who prefer a and receive maximum utility from the reallocation, but the total utility decreased. When students randomise, some of the seats are allocated to students who prefer b, which means that under $\beta_i \in (0, 1)$ the total utility is lower than under the initial assumption, hence, it decreases even further compared to the baseline case. □

Proposition 5. *1) If students who prefer a more popular programme a of both types overestimate their probability of admission into a, total utility increases. 2) If students who prefer a more competitive programme a of both types underestimate their probability of admission into a and overestimate the probability of admission into b, total utility decreases. 3) If students who prefer a more competitive programme a of one or both*

types underestimate their probability of admission into both a and b, the total utility comparison is ambiguous.

Proof. 1,2) The effect on the total utility is qualitatively the same as when all students are of the same type.

3) The fraction of students who prefer programme a among the students admitted into b needs to be compared to the baseline case. Denote the fraction of students of type t_1 as γ. In the baseline case, there are $St_{baseline} = \Big(\big(F(s_a^*) - F(s_b^*)\big)\gamma + \big(F(s_a^*) - F(s_b^*)\big)\big(1-\gamma\big)\Big)\alpha_a$ students who prefer a and are admitted into b, in the restricted reports' case there are $St_{restricted} = \big(F(s_{a,1}^e) - F(s_{b,1}^e)\big)\alpha_a$ of them. If $St_{baseline} < St_{restricted}$, the welfare increases and if $St_{baseline} > St_{restricted}$, the welfare decreases. □

Proposition 6. *In equilibrium with three programmes, students with each preference apply to one programme according to their estimated cutoffs as shown in Figure 1.7.*

Proof. Students are expected utility maximizers, so this result follows directly from Figure 1.8. □

Proposition 7. *In equilibrium with three programmes and two applications, students apply to programmes as shown in Figure 1.9 listing the programmes in the preference order.*

Proof. This result follows directly from Figure 1.8 and Marginal Improvement Algorithm. □

Chapter 2

Fair team formations with allocations of equal size

Abstract

We study fair team formations with allocations of equal size. We find that this case differs from a general case when every agent is allocated different number of indivisible goods. We update the notions of approximate fairness EF1, EFX, Pro1 and ProX as well as consider lexicographic and minimal envy-freeness. We find that most of the fairness properties are independent when the market is large (3 or more agents and more than 3 goods allocated to each agent). We show that Round Robin and generalised Round Robin are EF1, Pro1 and mEF, hence, all there fairness notions can be guaranteed in an allocation. These rules, however, are not necessarily efficient. The efficient Nash Max, Utalitarian Max and Leximin allocation mechanisms may not satisfy any of the fairness properties.

2.1 Introduction

Fair division is a problem that people are faced with on a regular basis. When splitting the rent and bills or allocating goods or jobs, we want to be fair to our friends, colleagues and employees. Although most of us recognise unfair behaviour, defining fairness in a unique way is not

always easy. We may split the rent equally or based on the room sizes, allocate tasks randomly or based on preferences and other criteria. Even when we agree on which division would be the fairest, the nature of the goods may prevent us from achieving it.

In the simpler problems, goods are divisible. Every good can be cut into infinitely many pieces and distributed between the agents. In reality, most goods are not divisible or are only divisible to a certain extent. We can think of money as being divisible, however, paying someone a half of a penny is not possible. We could cut a loaf of bread into as many pieces as we want, but we rarely distribute crumbs between guests or family members. It is therefore important to consider indivisible goods.

There are two branches of research that study fairness when goods are indivisible. The first suggests compensating the agents who receive less goods or goods of poorer quality. The compensations require transferable utilities or monetary transfers. In many cases, however, such transfers are not feasible or legal. Think of patients who need organ transplants, but cannot legally purchase the organs. When an organ becomes available, it is matched to a recipient based on a priority list that is constructed depending on the patients' circumstances. Monetary transfers for changing the place on the list are highly undesirable. The second approach, therefore, does not allow transfers. The goods have to be matched to agents in the fairest way with minimal disparity. We focus our attention on division of indivisible goods without transfers.

We consider a 'team formation problem', namely, a problem of dividing nk items between n agents such that each agents gets the same number k of goods. The agents have heterogeneous cardinal preferences over the goods. We assume that the valuations of the objects are additive, that is, the valuation of the bundle is equal to the sum of the valuations of the individual items in the bundle. The valuations do not

have to be positive. The aim of the division is reaching a *fair* allocation. Examples of fair team formation problems include assigning players to teams of equal size, allocating tasks or shifts to workers, students to supervisors, etc.

We study fairness using three different notions: efficiency, envy-freeness (EF) and proportionality (Pro). Efficiency states that the allocation cannot be improved for any agent without harming other agents. Envy-freeness suggest that there is no envy, that is, every agent prefers his allocation to allocations of other agents. Proportionality states that every agent receives at least $1/n$-th of the total utility of all of the available items combined.

There is extensive research on fair division. We can guarantee fairness when the goods are divisible, however, indivisibility makes the EF and Pro unattainable. Consider a division of 1 valuable object and $n-1$ equally worthless objects between n agents. The worthless objects will not reach $1/n$-th of total utility, and their owners will envy the person who receives the valuable object. Since EF and Pro are not feasible, we consider 'approximate' fairness by weakening the notions.

In the literature, we find common ways of relaxing the EF and Pro properties: "up to one good" and "up to any good". The former is commonly denoted with '1' (EF1/Pro1) and allows the agent to change his or other's bundle up to one good in *the most favourable way* before comparing his bundle to that of his peers. The latter is denoted with 'X' (EFX/ProX) and states that *any* alteration up to one good profitable to the agent must result in the desirable outcome. 'X' is clearly a stronger property than '1' as it considers *any profitable* rather than *the most favourable* change. To formalise the alteration of the bundles, an agent is usually permitted one of the following: (i) removing one item from another agent, (ii) adding a copy of an item to his allocation

without taking it away from another agent or (iii) disregarding one item belonging to a given agent, any other agent or the agent himself. Since our feasible allocations result in all agents receiving equal number of items, we adapt the notions of 'approximate' fairness by allowing an *exchange* of a single good, or 'single exchange'. The exchange does not take place in reality, rather it is a thought experiment conducted by the agent prior to evaluating his bundle.

We also consider notions of envy-freeness that are applicable to the case where each agent receives equal number of goods. These properties do not rely on exchanges, but compare the allocated bundles. Lexicographic envy-freeness (lexEF) prohibits the bundles dominating each other and minimal envy-freeness (mEF) states that the best item of any agent is better than the worst objects of other agents. Both properties are ordinal and do not require cardinal valuations.

2.1.1 Motivation

Fair division of indivisible goods has been considered in the literature, however, allocating equal number of goods to each of the agents has not been studied. Since the general and special cases have some significant differences, we are interested in the applications of the special case.

Real-life examples of allocating equal number of goods to agents include allocating players to teams, students to courses, professors to students for supervision and workers to shifts.

In many universities, students are allowed to choose which courses to study from a list. The total number of credits received by all students to achieve a degree is the same, and the courses are mostly worth the same number of credits. As a result, each student needs to study equal number of courses. Normally, students indicate which classes they would like to attend, but the allocation happens in different ways. In some universities

mechanisms are unclear or even undefined (if all of the courses have roughly the same demand), in others students are enrolled on the 'first come, first served' basis or according to their rating.

In problems of smaller sizes the allocations are often performed heuristically. In a relatively small department, doctors may have to take night shifts a few times a month. A provisional schedule is prepared by one member of the medical team and takes into account doctors' availability and preferences. It can then be altered by the doctors and finalised. If there are no volunteers, the unpopular shifts (ones that fall on public holidays, for example) are allocated to doctors in turn. Often a doctor who joined the department most recently would get the first unpopular shift.

Slightly bigger problem concerns allocating PhD students to Teaching Assistant (TA) jobs on different courses. In some cases the TAs are assigned to their supervisor's courses, however, big undergraduate courses need significantly more tutorials than smaller honours or postgraduate classes. Therefore, TAs can often express their interest in teaching specific courses as well as supply some additional information. Such information may include preferred semester of teaching, the number of hours per week and availability throughout the week. This year, opting out of face-to-face teaching also became possible. The applications are received by the School, and one of the members of the admin staff aggregates it trying to create the best possible allocation. The process is time-consuming and requires a lot of back and forth communication and negotiations. The resulting outcome may not be efficient as human factor plays a role is determining the heuristic approach and assigning jobs.

Where the allocation problem is larger or requires clear rules, drafting may be used. Drafting is a common way of recruiting new players

to teams in North America. It is not surprising that the richer teams can afford to hire more talented and expensive players. This, in turn, generates larger profits and increases the gap between the richer and poorer teams, making the former unbeatable. To remove the disparity to some extent and allow all teams to compete equally, drafting was introduces by National Football League (NFL) in 1936 [24]. The main goal was to stop teams from recruiting all the star players, thus, improving the competition. Eligible players participate in drafting process, they are generally young players representing high school teams or teams in junior leagues. The players are picked by teams in the major leagues in multiple rounds. In each round, each of the teams chooses a player that it can then sign. The picking order differs from league to league. National Football League uses the reverse order of the results from the previous season. Each of the 32 teams participates in 7 rounds of drafting, the choosing order remains unchanged throughout [42]. National Basketball Association (NBA)[32] and National Hockey League (NHL)[19] use lotteries assigning larger probabilities of being at the beginning of the line to teams that performed worse. A team that is clearly not doing well in a season can improve its position in the drafting order by losing the last few games on purpose when the order is deterministic. Randomisation removes the incentive to under-perform as coming last does not guarantee that the team will be the first to draft.

Our model closely resembles the drafting examples in which every team gets to pick the same number of players, and the players do not have a say in which team chooses him/her. Teams are 'agents' with distinct preferences, and players are 'goods' that are allocated to the agents.

The paper is structures in the following way. Chapter 2.2 covers the literature on the topic. Chapter 2.3 presents the model. Chapters 2.4 and 2.5 define and discuss properties and assignment rules. We present

the results on connections between the properties in Chapter 2.6 and show which assignment rules satisfy which of the properties in Chapter 2.7. We discuss the results and future work in Chapter 2.8. Our results are summarised in the Appendix.

2.2 Literature

Fair division literature started with the study of divisible goods. Steinhaus [41] studies the cake-cutting problem and introduced an algorithm that allowed dividing the cake between n players. Every player is guaranteed a fair share, that is, every player receives at least $1/n$-th of the cake's value according to his valuation of the pieces. We call this property *proportionality*. Interestingly, in this setup envy-freeness implies proportionality, the inverse is not true. Brams and Taylor [13] devised an algorithm that is envy free, however, the number of cuts that are made increases rapidly with the number of agents. Kurokawa et al.[27] bound the number of cuts when preferences are piece-wise linear. They also obtain a negative result: there are no finite ways of cutting a cake that are efficient, that is, satisfy Pareto optimality.

While in the problem with divisible goods Pro and EF can be reached, the indivisible goods often violate these properties. We mentioned an example of dividing one valuable and $n-1$ cheap goods between n agents. In order to study fairness in this set up, one needs to relax the notions of fairness to ones that are attainable.

The recent literature on fair divisions with indivisible goods lies in the intersection of economics and computer science. The authors not only study the properties of allocation rules, but also their computational complexity. Our paper is theoretical, hence, we focus on the known theoretical results. Caragiannis et al. [16] study the properties of max-

imum Nash welfare procedure which maximises the product of utilities of all agents. They show that in addition to being fair when the goods are divisible, it is also EF1. Kurokawa et al. [28] focus on existence of MMS guarantee. In more detail, they verify if every agent receives a weakly better bundle than what he can guarantee by partitioning the set of items and choosing the worst bundle (the number of subsets in the partition is the same as the number of players). They show that although reaching MMS guarantee is not always possible, allocations with 2/3 of MMS guarantee always exist.

Azevedo and Budish [3] propose a new way of analysing incentive compatibility. *Strategy-proofness in the large* implies that truth-telling is optimal within a small neighbourhood in large markets. Lee [29] proves that in large one-to-one markets do not have significant incentives to manipulate the allocation. Budish [15] studies the compatibility of fairness, efficiency and strategy-proofness. He presents Approximate Competitive Equilibrium from Equal Incomes Mechanism and shows that it guarantees approximate $(N+1)$ - Maximin Share guarantee (A-MMS) and Envy Bounded by a Single Good (EF1), and is strategyproof in the large. EF1 incorporates a comparison of the bundle of agent i to that of agent j without one of the goods. In our paper, we alter this definition as exactly k objects must be allocated to all agents, hence, removing a good from j while keeping the original bundle of i is unnatural.

Plaut and Roughgarden [35] study envy-freeness up to any good (EFX) and show existence of EFX allocations in some settings. Bouveret, Sylvain and Lemaître [12] study five properties and how they are connected to each other. The properties under considerations are max-min fair-share (MFS), proportional fair-share (Pro), envyfreeness (EF), competitive equilibrium from equal incomes (CEEI) and min-max fair-share (mFS). They show that these criteria can be ordered according to

their strength and that CEEI $\Rightarrow$ EF $\Rightarrow$ mFS $\Rightarrow$ Pro $\Rightarrow$ mFS.

The study of fair division and allocation mechanisms has many practical implications. It can help people allocate goods and tasks as well as split rent and bills. A website called *Spliddit*[1] is available to everyone, and uses the latest results in fair division literature to achieve best possible allocations.

Our work contributes to the literature as we study the fairness properties in a natural setting. Despite seemingly small difference from the general case, we find that some of the established links do not hold, and mechanisms stop satisfying certain properties. In our set up many of the notions of envy-freeness are independent, and EF1 does not imply Pro1. Maximum Nash welfare procedure is not necessarily EF1 anymore. Moreover, allocations that maximise the product and sum of utilities of all agents or maximise the minimum utility among the agents may not satisfy any of the fairness notions. Round Robin (RR) procedure, in contrast, is EF1, Pro1 and minimal EF (mEF). We cannot guarantee lexicographic EF (lexEF) when using RR or generalising the procedure (gRR), in addition, neither RR nor gRR can guarantee efficiency.

When studying the allocation procedures, we restrict our attention to division with positive valuations of the goods by all agents. Although, the properties of allocations allow affine transformation of utilities, the assignment mechanisms need to be adapted to the cases where some (or all) of the distributed items are 'bads' rather than 'goods'. In general, properties of the mechanisms may change in presence of bads or mixed goods [10, 11]. We expect that some changes may persist when the number of items allocated to all agents is the same, and plan to examine this in future papers.

[1] http://www.spliddit.org

2.3 Model

There is a set of agents $N = \{1, ..., n\}$, and a set of indivisible goods A with nk goods. The goods are to be divided between the agents, and each agent receives k items from A. We assume that $n \geq 2$ and $k \geq 2$. We only consider deterministic allocations and do not allow monetary transfers.

Agents have cardinal additive utilities over goods and sets of goods $u_i : A \to \mathbb{R}$, where for any $S \subset A$, $u_i(S) = \sum_{a \in S} u_i(a)$.

Z_i is a set of objects allocated to agent i. A feasible allocation is an ordered equipartition of A, $(Z_1, ..., Z_n)$, that is, $\bigcup_{i=1}^{n} Z_i = A$ and $Z_i \cap Z_j = \emptyset$, $|Z_i| = k$ for any agents $i \neq j$.

The set of all feasible allocations is denoted by Π.

An assignment rule is $f = (f_1, ..., f_n) : (u_1, ..., u_n) \longmapsto \Pi$.

We use $a \succeq_i b$ and $u_i(a) \geq u_i(b)$ interchangeably when agent i weakly prefers item a to item b. We also write $Z_i \setminus \{a\}$ as $Z_i - a$ and $Z_i \cup \{b\}$ as $Z_i + b$.

In our model, affine transformations of utilities are allowed without loss of generality. Consider $v_i(\cdot) = \alpha_i u_i(\cdot) + \beta_i$, where $\alpha_i \in \mathbb{R}_+$ and $\beta_i \in \mathbb{R}$. Since the utilities are additive and each agent i is assigned a set Z_i with exactly k elements, $v_i(Z_i) = \alpha_i u_i(Z_i) + k\beta_i$, hence, the utility of the allocation is re-scaled by the same multiplier α_i and by a constant $k\beta_i$. If under $u_i(\cdot)$ agent prefers Z_i to Z_i' or yields more than average utility from Z_i, then the same is true under $v_i(\cdot)$. As a result, the properties that we study (Pareto Optimality, Proportionality and Envy-Freeness) are invariant with respect to any affine changes to utilities. This allows us to normalise agents' utility functions. We will often assume that the total utility of all goods is equal to 1 and that the individual utilities of all goods are positive, that is, $u_i(a) \geq 0$ and

$u_i(A) = \sum_{a \in A} u_i(a) = 1$ for all $i \in N$. This is in contrast to the general model where considering goods, bads and mixed goods yields different results, so such transformations cannot be applied.

Many of the examples used in the proofs contain goods with identical utilities or agents who are indifferent between goods, however, since all of the examples are satisfied with strict inequalities, they are still valid in the neighbourhood of the utility vectors. It follows that our results hold generally.

2.4 Properties: definitions

Deterministic allocations without monetary transfers pose serious restrictions on providing meaningful fairness guarantees to all agents. Although we use the classical definition of Pareto Optimality, other desirable properties, such as envy-freeness and proportional shares become unfeasible. We weaken these notions slightly and consider 'approximate' guarantees when a property holds after a minor change to the allocation.

We denote allocations of agents i and j by Z_i and Z_j before the exchange and by Z_i' and Z_j' after the *single exchange*. The single exchange consists of agent i giving one of his items to agent j and receiving one the j's items in return.

2.4.1 Pareto Optimality

Pareto Optimality (PO). An allocation is Pareto Optimal if it is impossible to make some agents better off without making some other agents worse off. Since Π is finite, PO allocation always exists.

2.4.2 Envy-Freeness

We start by providing the definition of envy-freeness common to the literature.

Envy-Freeness (EF): for any two agents $i, j \in N$, $u_i(Z_i) \geq u_i(Z_j)$.

Envy-freeness states that any agent prefers his bundle to those of his peers. Agents compares all bundles according to their own preferences. They do not compare their utility to the utilities that other agents receive from their bundles. In the spirit of the general model, we consider envy-freeness up to one good. Since every agent is allocated exactly k goods, we compare the utilities after a single exchange rather than remove items from either of the bundles.

EF1: for any two agents $i, j \in N$,

> *either* $u_i(Z_i) \geq u_i(Z_j)$
> *or* there is $a \in Z_j$ and there is $b \in Z_i$ s.t. $u_i(Z_i + a - b) \geq u_i(Z_j - a + b)$ (we can think of the exchange as the most favourable to i, where a is the best object for i in Z_j, and b is the worst object for i in his own allocation).

We also consider envy-freeness up to any good, which means that any profitable exchange with another agent eliminates envy.

EFX: for any two agents $i, j \in N$,

> *either* $u_i(Z_i) \geq u_i(Z_j)$
> *or* for any $a \in Z_j$ and any $b \in Z_i$ it follows that $u_i(Z_i + a - b) \geq u_i(Z_j - a + b)$ as long as $u_i(b) < u_i(a)$.

In addition, we define minimal and lexicographic envy-freeness. These notions follow the spirit of minimal guarantees. Minimal envy-freeness suggest that any agent prefers his best item to the worst items of his

peers. Lexicographic envy-freeness does not allow any bundle to dominate others. It is only applicable when agents receive exactly k goods.

Minimal Envy-Freeness (mEF): for any two agents $i, j \in N$,

$\underline{either}$ $u_i(Z_i) \geq u_i(Z_j)$

$\underline{or}$ $\max_{a \in Z_i} u_i(a) > \min_{b \in Z_j} u_i(b)$, that is, agent i prefers his best item to the worst item (according to preferences of i) of j .

Lexicographic Envy-Freeness (lexEF). Consider two potential allocations S and T, where $S, T \subset A$ and $|S| = |T| = k$. We say that allocation S dominates (or 'lex-dominates') allocation T for agent i if there exists a bijection $f : S \to T$ such that $u_i(a) \geq u_i(f(a))$ for all $a \in S$, and at least one inequality is strict.

lexEF: for any two agents $i, j \in N$, Z_j does not dominate Z_i for agent i.

Note that mEF and lexEF are purely ordinal properties, they only depend on the ordinal preferences of an agent, and not on the cardinal utility functions. Note also that mEF and lexEF seem weaker than EF1 and EFX as they do not require comparing the utilities of the whole allocation. However, they do not allow a single exchange in contrast to EF1 and EFX, and compare existing allocations instead. This results in independence of the two types of notions, as we show in our paper.

2.4.3 Proportionality

Similarly, we provide the definition of Proportionality and Proportionality up to one and up to any good.

Proportionality (Pro): For any agent $i \in N$, $u_i(Z_i) \geq \frac{1}{n} u_i(A)$.

Approximate proportionality, like EF1 and EFX, make use of single exchanges. Allocation is Pro1 if every agent can receive at least 1/n-th

of the total utility after a single exchange. The item that each agent receives after a single exchange may be assigned to any other agent in the initial allocation.

Pro1: For any agent $i \in N$,

> *either* $u_i(Z_i) \geq \frac{1}{n}u_i(A)$
> *or* there are $a \notin Z_i$ and $b \in Z_i$ such that $u_i(Z_i + a - b) \geq \frac{1}{n}u_i(A)$ (we can think of a as the best object for i outside of the assigned set Z_i, and b as the worst object for i in his own allocation).

Allocation is ProX if every agent receives at least 1/n-th of the total utility after any profitable single exchange.

ProX: For any agent $i \in N$,

> *either* $u_i(Z_i) \geq \frac{1}{n}u_i(A)$
> *or* for any $a \notin Z_i$ and for any $b \in Z_i$ it follows that $u_i(Z_i + a - b) \geq \frac{1}{n}u_i(A)$ as long as $u_i(b) < u_i(a)$.

An alternative definition of fairness up to one good in our case may allow agent i to replace one of his objects by an object belonging to j without removing it from j. This would improve Z_i while keeping Z_j constant. Both bundles would still consist of k goods, however, a duplicate of one of the goods will be created. We do not consider this property in this paper for two reasons: i) we do not want to allow for creation of duplicates (even if they are imaginary), ii) this notion is stronger than EF1 above. Although we find that EF1 can be guaranteed, it does not hold for the rules that are known to be EF1 in a general case with different number of goods assigned to the agents.

2.5 Assignment rules: definitions

We consider five assignment rules and their properties in our paper.

1. Round Robin (RR). Agents pick best available objects one by one, according to an exogenous ordering $(i_1, \ldots, i_n)$ in k rounds (the ordering is the same for all rounds).
2. Generalised Round Robin (gRR). Agents pick best available objects one by one, according to an ordering $\pi = (i_1, \ldots, i_{nk})$. Every agent appears once in a section of the ordering $(i_{n(r-1)+1}, \ldots, i_{nr})$ where $r = 1, \ldots, k$. Compared to RR, in gRR the picking order may differ in every round.
3. Leximin (LM). Among all allocations in Π, choose the ones that maximise utility of the least happy agent, among those - the ones which maximise penultimate lowest utility, etc. Note that LM allocations are the maximal elements of the leximin ordering, which first re-arranges the coordinates of the vectors of utilities in ascending order and then compares them by the first different coordinate.
4. Nash Max (NM). Among all allocations in Π, choose the ones that maximise the product of utilities of all agents.
5. Utilitarian Max (UM). Among all allocations in Π, choose the ones that maximise the sum of utilities of all agents.

Note that LM, NM and UM may result in non-unique allocations, and are therefore correspondences.

All of the above allocation rules are well-known and well-studied. RR and gRR algorithms rely on ordinal preferences and do not need any adjustments when the preferences are described by a utility function.

LM, NM and UM may result in different allocations after a transformation of the utility functions. NM is not sensitive to scale, that is, multiplying all utilities by the same positive factor will not change the allocation, however, adding the same number to all utilities may affect the result. The opposite is true for LM and UM, which means that some form of normalisation may be necessary to make sure that all agents are

treated equally. Moreover, additional clarifications are required when applying NM in case some of the individual utilities are negative. If the number of agents with negative utility is odd, then the product of utilities will become negative, and an increase in total utility may be achieved through a decrease in individual utility of an agent whose evaluation of his bundle is positive. This is the first paper on this topic, so we will not focus on these issues and assume that all the goods have positive valuations when applying the rules. We will show that even if that is true, none of the properties are guaranteed to hold. We plan to extend the research to negative utilities in future.

2.6 Compatibility of properties

We start by investigating the relations between different properties. We have two goals. Firstly, we want to know if some of the properties imply others. Secondly, we want to know if some properties are *compatible*, that is, if there exists an allocation that satisfies all of these properties. In the general case, properties form a hierarchy. In our case, when values of n and k are small, there are some implications and equivalences. As n and k increase, logical independence becomes more common.

EF1 holds whenever each agent receives two goods as a result of the allocation. We therefore will not say that EF1 follows from any other property when $k = 2$ because it simply always holds in that case.

Proposition 8. *Any allocation is EF1 if* $k = 2$.

Proof. Consider any two agents i and j with two items allocated to each. According to u_i, goods allocated to i yield x_1 and x_2 and goods allocated to j are worth y_1 and y_2. Let $x_1 > x_2$ and $y_1 > y_2$.

If $x_1 + x_2 \geq y_1 + y_2$, then agent i does not envy agent j. If $x_1 + x_2 <$

y_1+y_2, after a single exchange he will have the better items from each of the allocated bundles. The total utility of i's altered bundle, x_1+y_1, will exceed that of j's, x_2+y_2, which follows from term-by-term comparison. Hence, the allocation is EF1. □

Now let $k = 3$. Minimal envy freeness states that the best object of agent i is preferred to the worst object of agent j. If the allocation is mEF, then it will also be EF1. The inverse is not necessarily true. A single exchange can improve i's utility enough to close the gap between the bundles even if the initial allocation provided agent i with good that he considers worse than any of the goods allocated to j.

Proposition 9. *mEF implies EF1 if $k = 3$, but EF1 does not imply mEF.*

Proof. Denote element of the set of goods of agent i as a_i, then $Z_i = \{a_i^1, a_i^2, \ldots, a_i^k\}$. Without loss of generality, we label all goods of each agent in a descending order according to u_i. Then for any j, $u_i(a_j^1) \geq u_i(a_j^2) \geq \cdots \geq u_i(a_j^k)$. Following this notation,

$$u_i(a_i^1) \geq u_i(a_i^2) \geq u_i(a_i^3) \tag{2.1}$$

$$u_i(a_j^1) \geq u_i(a_j^2) \geq u_i(a_j^3) \tag{2.2}$$

According to mEF,

$$u_i(a_i^1) \geq u_j(a_i^3) \tag{2.3}$$

Let $a = a_j^1$ and $b = a_i^3$, we then compare $u_i(a_i^1+a_i^2+a_j^1)$ to $u_i(a_j^2+a_j^3+a_i^3)$. From (2.1), (2.2) and (2.3) follows that:

$$u_i(a_i^2) \geq u_i(a_i^3)$$

$$u_i(a_j^1) \geq u_i(a_j^2)$$

$$u_i(a_i^1) \geq u_i(a_j^3)$$

Adding them up, we get $u_i(a_i^1 + a_i^2 + a_j^1) \geq u_i(a_j^2 + a_j^3 + a_i^3)$, hence, EF1 holds.

Now, consider an allocation which is EF1, but not mEF. Let $u_i(b) = 1$ for any $b \in Z_i$, $u_i(a_j^1) = 4$ and $u_i(a) = 2$ for any $a \in Z_j \setminus a_j^1$. mEF does not hold since $1 < 2$. $EF1$ holds as

$$u_i(a_i^1 + a_i^2 + a_j^1) > u_i(a_j^2 + a_j^3 + a_i^3)$$
$$1 + 1 + 4 > 2 + 2 + 1$$
$$6 > 5$$

□

When more goods are allocated to agents, mEF and EF1 are independent. mEF relies on ordinal utilities of the goods in the allocated bundles, while EF1 compares the cardinal utilities after a single exchange. We show that mEF and EF1 may hold simultaneously, one at a time or not at all.

Proposition 10. *If $k > 3$, mEF is logically independent of EF1; mEF and EF1 are not necessarily satisfied.*

Proof. Using the same notation as in Proposition 8,

$$u_i(a_i^1) \geq u_i(a_i^2) \geq \cdots \geq u_i(a_i^k) \tag{2.4}$$
$$u_i(a_j^1) \geq u_i(a_j^2) \geq \cdots \geq u_i(a_j^k) \tag{2.5}$$

Let $a = a_j^1$ and $b = a_i^3$. If EF1 holds, then

$$u_i(Z_i + a - b) \geq u_i(Z_j - a + b)$$
$$u_i(a_i^1 + \cdots + a_i^{k-1} + a_j^1) \geq u_i(a_j^2 + \cdots + a_j^k + a_i^k)$$

The above inequality may be true or false depending on the value of each good. Consider four examples:

Example 3. Let $u_i(b) = 10$ for any $b \in Z_i$ and $u_i(a) = 5$ for any $a \notin Z_i$. mEF holds and the allocation is EF1.

Example 4. Let $u_i(b) = 5$ for any $b \in Z_i$ and $u_i(a) = 10$ for any $a \notin Z_i$. mEF does not holds and the allocation is not EF1 as $k > 3$ (the exchange cannot compensate for the difference in bundles' utilities).

Example 5. Let $u_i(a_i^1) = 6$, $u_i(a_i^l) = 3$ for any $a_i^l \in Z_i \setminus a_i^1$ and $u_i(a) = 5$ for any $a \notin Z_i$. mEF holds since $u_i(a_i^1) = 6 > u_i(a_j^k) = 5$ for any $j \in N \setminus i$.
The original allocation is not $EF1$ since $u_i(Z_i + a - b) < u_i(Z_j - a + b)$ at $k > 3$. EF1 holds if

$$u_i(a_i^1 + \cdots + a_i^{k-1} + a_j^1) > u_i(a_j^2 + \cdots + a_j^k + a_1^k)$$
$$6 + 3(k-2) + 5 > 5(k-1) + 3$$
$$3.5 > k$$

Since we are considering a case when $k > 3$ and k is an integer, EF1 does not hold.

Example 6. Let $u_i(b) = 1$ for any $b \in Z_i$, $u_i(a_j^1) = k + 1$ and $u_i(a) = 2$ for any $a \in Z_j \setminus a_j^1$. mEF does not hold since $1 < 2 < k+1$. $EF1$ holds as

$$u_i(a_i^1 + \cdots + a_i^{k-1} + a_j^1) > u_i(a_j^2 + \cdots + a_j^k + a_1^k)$$
$$1(k-1) + (k+1) > 2(k-1) + 1$$
$$k + 1 > k$$

Based on the examples above, we observe that mEF and $EF1$ can hold simultaneously as well as one can hold while the other does not, hence, they are logically independent. Moreover, both of them may not hold. □

We now examine two ordinal properties: mEF and lexEF. mEF compares one item from i's bundle to one item from j's, while lexEF considers the whole bundles of both agents. It is not surprising that lexEF turns out to be stronger than mEF regardless of the values of k and n.

Proposition 11. *lexEF implies mEF, but mEF does not imply lexEF.*

Proof.

lexEF $\Rightarrow$ mEF. Allocation is lexEF when for any $i, j \in N$, Z_j does not dominate Z_i according to preferences of i. Since there is no domination, either all objects are the same, or there are $a \in Z_i$ and $b \in Z_j$ such that $a \succ_i b$ and $c \in Z_i$ and $d \in Z_j$ such that $c \prec_i d$. Then, either $u_i(Z_i) = u_i(Z_j)$ or $\max_{a \in Z_i} u_i(a) > \min_{b \in Z_j} u_i(b)$. It is possible that both conditions hold if, for example, agent are allocated identical bundles of goods with various utilities.

mEF $\not\Rightarrow$ lexEF. Consider RR allocation assuming that all agents have the same preferences. It is mEF since every object chosen by any agent in the first round is preferred by that agent to any of the k object left for the last round. The allocation is not lexEF as allocation of an agent i who was choosing the objects before j in every round dominates the allocation of j.

□

Although lexEF is stronger than mEF, it is related to EF1 in the same way as mEF.

Proposition 12.

(i) lexEF implies EF1, but EF1 does not imply lexEF when $k = 3$.

(ii) lexEF and EF1 are logically independent $k > 3$.

Proof.

(i) By combining Propositions 9 and 11, we find that lexEF $\Rightarrow$ mEF $\Rightarrow$ EF1 and EF1 $\not\Rightarrow$ lexEF, otherwise it would imply mEF.

(ii) By combining Propositions 10 and 11, we find that EF1 $\not\Rightarrow$ lexEF, otherwise it would imply mEF.

We show that lexEF $\not\Rightarrow$ EF1 by providing an example that is lexEF, but it not EF1.

Example 7. Consider $Z_i = a, c, \ldots, c$ and $Z_j = b, \ldots, b$, where $u_i(a) = x > u_i(b) = y > u_i(c) = z$ and $u_j(a) \geq u_j(b) \geq u_j(c)$. lexEF holds since agent i has both the best and the worst available goods, whereas agents of type j have the medium good. EF1 does not hold whenever

$$\begin{aligned} x + y + (k-2)z &< (k-1)y + z \\ x - z &< (k-2)y - (k-2)z \\ x - y &< (k-2)(y-z) \end{aligned}$$

Let $x - y = 1$ and $y - z = 2$, then EF1 does not hold when $k > 3$.

□

Large number of division problems such as corporate legislation, dissolving partnerships and divorce settlements involve only two parties. It turns out that when $n = 2$, EF1 and Pro1 are equivalent and EF1 is compatible with efficiency. Note that EF1 always holds when $k = 2$, therefore, when two agents are diving four indivisible goods, the resulting allocations will always be EF1 and Pro1.

Proposition 13. *When $n = 2$, EF1 and Pro1 are equivalent.*

Proof. Because there are only two agent, the agent will perform the same single exchange and consider the same $Z_i^{'}$ in both EF1 and Pro1. Not to be envious, the agent needs to have a better bundle from the start or prefer the post-exchange bundle $Z_i^{'}$ to $Z_j^{'}$. Given that there are only two bundles in the partition, the weakly better bundle has a utility of at least $\frac{1}{2}u_i(A)$, hence, EF1 implies Pro1.

Pro1 implies EF1 as if $u(Z_i^{'}) \geq \frac{1}{2}u_i(A)$, then $u(Z_i^{'}) \geq u(Z_i^{'})$ since $A = Z_i^{'} \cup Z_j^{'}$. □

In the general model with no limitations on how many goods are assigned to each agent, EF1 implies Pro1. In contrast to the general case, we find that whenever there are three or more agents EF1 and Pro1 are independent.

Proposition 14. *When $n \geq 3$, EF1 and Pro1 are logically independent.*

Proof.

EF1 $\not\Rightarrow$ Pro1. Assume that there is one agent i, $(n-1)$ identical agents j and three types of goods: a, b and c. Agents j are indifferent between the goods, while $u_i(a) = x > u_i(b) = y > u_i(c) = z$. There are $(n-1)$ items a, $n(k-1)-1$ items b and two items c, and the agents are allocated $Z_i = \{c, c, b, \ldots, b\}$ and $Z_j = \{a, b, \ldots, b\}$.

This allocation satisfies EF1. If i takes a from agent j and gives her c in exchange, their allocations become $Z_i^{'} = \{a, c, b, \ldots, b\}$ and $Z_j^{'} = \{c, b, b, \ldots, b\}$. Since i prefers a to b, and b to c, this single exchange eliminates envy, hence, EF1 holds. Agent j is indifferent between the items, hence, the allocation is EF for j.

This allocation is not necessarily Pro1 for agent i. The best exchange for i is described above, and results in $u_i = x + z + (k-2)y$.

Pro 1 holds if

$$\begin{aligned} u_i(Z_i') &\geq \frac{1}{n}u_i(A) \\ x + z + (k-2)y &\geq \frac{(n-1)x + \Big(n(k-1)-1\Big)y + 2z}{n} \\ x + z + (k-2)y &\geq \frac{(n-1)x + \big(n(k-1)-1\big)y + 2z}{n} \\ nx + nz + n(k-2)y &\geq (n-1)x + \big(n(k-1)-1\big)y + 2z \\ x + (n-2)z &\geq (n-1)y \\ x - y &\geq (n-2)(y-z) \end{aligned}$$

This condition does not necessarily hold. Let, for example, $x = 4$, $y = 3$, $z = 1$, then Pro 1 is not satisfied for any $n \geq 3$. Moreover, if the utility increases linearly, Pro1 will still not hold when $n > 3$.

Pro1 $\not\Rightarrow$ EF1. Since $EF1$ holds when $k = 2$, we consider $k \geq 3$. Assume that there is one agent i, $(n-2)$ identical agents j and an agent k. Assume three types of goods: a, b and c. Agents j and k are indifferent between the goods, while $u_i(a) = x > u_i(b) = y > u_i(c) = z$. There is one item a, $(n-2)k+1$ items b and $2k-2$ items c. The agents are allocated bundles $Z_i = \{c, \ldots, c\}$, $Z_j = \{b, b, \ldots, b\}$ and $Z_k = \{a, b, c, \ldots, c\}$.

Let us first find the values of x, y and z for which Pro1 holds. The best single exchange for agent i results in $Z_i^{'} = a, c, \ldots, c$. To

satisfy Pro1,

$$\begin{aligned} u_i(Z_i^{'}) \geq & \frac{1}{n} u_i(A) \\ x + (k-1)z \geq & \frac{x + \Big((n-2)k+1\Big)y + (2k-2)z}{n} \\ nx + n(k-1)z \geq & x + \Big((n-2)k+1\Big)y + (2k-2)z \\ (n-1)x + (nk-n-2k+2)z \geq & (nk-2k+1)y \\ (n-1)(x-z) \geq & (nk-2k+1)(y-z) \\ (x-z) \geq & \frac{nk-2k+1}{n-1}(y-z) \end{aligned}$$

Note that $\frac{nk-2k+1}{n-1} = k - \frac{k-1}{n-1} < k$ and $\frac{nk-2k+1}{n-1} > 1$ whenever $k > 1$ and $n > 2$, which means that it is always between 1 and k in the case under consideration. Let $x = k+1$, $y = 2$, $z = 1$. It follows, that Pro1 holds.

The allocation,however, is not EF1 as agent i envies agent j even after a single exchange. $Z_i^{'} = \{b, c, \ldots, c\}$ and $Z_j^{'} = \{c, b, \ldots, b\}$, and

$$\begin{aligned} u_i(Z_i^{'}) < & u_i(Z_j^{'}) \\ y + (k-1)z < & z + (k-1)y \\ (k-2)z < & (k-2)y \end{aligned}$$

It follows, that i envies j whenever $k > 2$. Since we are considering $k \geq 3$, EF1 does not hold regardless of the values of x, y and z.

□

The desirable properties of any allocation include fairness and efficiency. Although compatibility of PO with all of the notions of envy-freeness and proportionality is an open question, EF1 and PO are compatible in the market with two agents.

Theorem 1. *If n=2, EF1 and efficiency are compatible when utilities are normalised.*

Proof. Normalise the utilities such that $\sum_{a\in A} u_i(a) = 1$ for $i = 1, 2$. Allocate the bundles based on leximin mechanism, that is, if the allocation results in $u_1 < u_2$, then agent 1 gets the largest possible u_1. This allocation is efficient since one of the agents receives the $\max\min u_i$.

There are three possibilities:

1. $u_1 < 1/2, u_2 < 1/2$. Both agents benefit from exchanging their allocated goods, so the original allocation was not leximin.
2. $u_1 > 1/2, u_2 > 1/2$. Both agents do not envy each other, hence, EF holds.
3. $u_1 < 1/2, u_2 > 1/2$. Let agent 1 exchange his worst good for the best good of agent 2. Assume that the resulting utility $u_1(S_1^{'}) < 1/2$, note that $u_1(S_1^{'}) > u_1(S_1) = x$. Since the initial allocation (S_1, S_2) was leximin, it must be that in any allocation other than (S_1, S_2) one of the agents receives utility inferior to x.
   If $u_1(S_1^{'}) < 1/2$ and $u_2(S_2^{'}) < 1/2$, then, $u_1(S_2^{'}) > 1/2$ and $u_2(S_1^{'}) > 1/2$. Both agents prefer $(S_2^{'}, S_1^{'})$ to the initial allocation, thus, the initial allocation was not leximin. *Contradiction.*
   If $u_1(S_1^{'}) < 1/2$ and $u_2(S_2^{'}) > 1/2$, then both $u_1(S_1^{'}) > x$ and $u_2(S_2^{'}) > x$. The initial allocation was not leximin. *Contradiction.*
   It follows that our initial assumption was incorrect, and $u_1(S_1^{'}) > 1/2$. Consequently, the initial allocation is EF1.

□

We consider six ways of evaluating fairness: four notions of envy-freeness and two notions of proportionality. We find that most of them are independent when enough agents are present in the market and enough goods are allocated. With exception of EFX implying EF1 and

ProX implying Pro1 (by definition) as well as lexEF being stronger than mEF, we show that the properties can hold or not hold independently of each other. We do not consider whether combination of a few properties implies another one in this paper.

Apart from the strength of the fairness properties, we would like to know if they are compatible. In the next section we show that RR satisfies EF1, Pro1 and mEF, hence, the three are compatible and can be guaranteed.

2.7 Mechanisms' Properties

We now study the allocation mechanisms and their properties. We consider five rules: RR, gRR, LM, NM and UM.

Round Robin is a rule that is common and simple to use. Agents pick the best of the remaining objects one-by-one until there are no more objects left. We show that RR is EF1, Pro1 and mEF, however it is not efficient. It also fails EFX and ProX.

Proposition 15. *RR is not necessary PO.*

Example 8. Consider an example with $n = 2$ and $k = 3$. Denote the two players by i and j and goods by a, b, c, d, e and f. Assume that both players prefer $a \succ b \succ c \succ d \succ e \succ f$, and that player i chooses first. Then, following the Round Robin assignment rule, goods a, c, e are allocated to i and b, d, f are allocated to j. If $u_i(b)+u_i(d) > u_i(a)+u_i(e)$ and $u_j(a) + u_j(e) > u_j(b) + u_j(d)$, then this allocation is not PO. Below is a table with numerical values that satisfy this condition.

RR	a	b	c	d	e	f	$u(Z)$
i	$\frac{10}{32}$	$\frac{9}{32}$	$\frac{6}{32}$	$\frac{5}{32}$	$\frac{2}{32}$	0	$\frac{18}{32}$
j	$\frac{12}{40}$	$\frac{10}{40}$	$\frac{8}{40}$	$\frac{5}{40}$	$\frac{4}{40}$	$\frac{1}{40}$	$\frac{16}{40}$

Table 2.1: RR allocation

PI	a	b	c	d	e	f	$u(Z)$
i	$\frac{10}{32}$	$\frac{9}{32}$	$\frac{6}{32}$	$\frac{5}{32}$	$\frac{2}{32}$	0	$\frac{20}{32}$
j	$\frac{12}{40}$	$\frac{10}{40}$	$\frac{8}{40}$	$\frac{5}{40}$	$\frac{4}{40}$	$\frac{1}{40}$	$\frac{17}{40}$

Table 2.2: Pareto Improvement (PI)

The total utility of agent i is $u_i(a,c,e) = \frac{10}{32} + \frac{6}{32} + \frac{2}{32} = \frac{18}{32}$. Agent j receives $u_j(b,d,f) = \frac{10}{40} + \frac{5}{40} + \frac{1}{40} = \frac{16}{40}$. If i receives goods b,c,d and j get goods a,e,f, then the total utilities are $u_i(b,c,d) = \frac{9}{32} + \frac{6}{32} + \frac{5}{32} = \frac{20}{32} > \frac{18}{32}$ and $u_j(a,e,f) = \frac{12}{40} + \frac{4}{40} + \frac{1}{40} = \frac{17}{40} > \frac{16}{40}$. The new allocation is a Pareto Improvement (shown in the PI table above), hence, RR does not satisfy PO.

We now check if the RR allocations satisfy envy-freeness up to one and up to any good.

Proposition 16. *RR is EF1, but not necessarily EFX.*

Proof. According to EF1, for any agents i, j there is $a \in Z_j$ and $b \in Z_i$ such that $u_i(Z_i + a - b) \geq u_i(Z_j - a + b)$. EFX establishes the same relation for *any* $a \in Z_j$ and $b \in Z_i$ s.t. $u_i(a) > u_i(b)$.

Let a_i^l be an object chosen by agent i in the l^{th} round. Consider any two agents i and j. Consider two possibilities for the picking order in the Round Robin rule:

i chooses before j. If i chooses the objects before j, then the allocation is EF. At any round l, $u_i(a_i^l) > u_i(a_j^l)$, hence, $u_i(Z_i) > u_i(Z_j)$ (follows from additivity).

j chooses before i. If i chooses the objects after j, then in the first round j may choose an object that is most preferred by i, however, at any round $l \in 1..k-1$, $u_i(a_i^l) > u_j(a_j^{l+1})$. This implies that $u_i(a_i^1, \ldots, a_i^{k-1}) \geq u_i(a_j^2, \ldots, a_j^k)$. If $u_i(a_i^k) > u_i(a_j^1)$, then the allocation is EF. If the opposite is true, then it satisfies EF1 as for

$a = a_j^1$ and $b = a_i^k$ it follows that $u_i(Z_i + a - b) \geq u_i(Z_j - a + b)$ by additivity.

When it comes to EFX, any profitable exchange must guarantee envy-freeness. Let agents have the same valuations of objects with a fixed increment ε. Then j's bundle is better than i's by at least $k\varepsilon$, however, the smallest profitable exchange will improve i's utility by ε and decrease j's utility by ε, meaning that $u_i(Z_j^{'}) - u_i(Z_i^{'}) = (k-2)\varepsilon > 0$, hence, EFX does not hold whenever $k > 2$. If $k = 2$, let a, b, c, d be the best choices for i from A ($u_i(a) > u_i(b) > u_i(c) > u_i(d)$), and let the allocations of i and j be $Z_i = \{b, d\}$ and $Z_j = \{a, c\}$. One of the profitable exchanges for i results in $Z_i^{'} = \{b, c\}$ and $Z_j^{'} = \{a, d\}$. If, for example, $u_i(a) = 5, u_i(b) = 3, u_i(c) = 2$ and $u_i(d) = 1$, then EFX does not hold.

□

Similarly to envyfreeness, proportionality up to one good is guaranteed by RR, however, ProX may not hold.

Proposition 17. *RR is Pro1, but not necessarily ProX.*

Proof.

Pro1. We will use the same notation as in Proposition 16: a_i^l is an object chosen by agent i in the l^{th} round. In every round of RR, agent i chooses the best available item, hence, $u_i(a_i^l) > u_i(a_j^{l+1})$. When comparing his bundle for Pro1, agent i will either find that Pro holds or exchange a_i^k for the best item belonging to the other agents a^{best}. Then, $u_i(Z_i^{'}) = u_i(a_i^1) + \cdots + u_i(a_i^{k-1}) + u_i(a^{best})$. Compare it to the average

utility:

$$u_i(Z_i') > \frac{1}{n} u_i(A)$$

$$u_i(a_i^1) + \cdots + u_i(a_i^{k-1}) + u_i(a^{best}) > \frac{1}{n} \sum_{\substack{j=1,\ldots,n \\ l=1,\ldots,k}} u_i(a_j^l)$$

$$n\Big(u_i(a_i^1) + \cdots + u_i(a_i^{k-1}) + u_i(a^{best})\Big) > \sum_{l=1,\ldots,k} u_i(a_1^l) + \cdots + u_i(a_n^l)$$

If $u_i(a^{best}) < u_i(a_i^k)$, then agent i received his k top choices from A. This implies Pro.

If $u_i(a^{best}) > u_i(a_i^k)$, then for every object in Z_i chosen in the first $k-1$ rounds, the corresponding utility is present on both sides on the inequality, for the k^{th} round we use $u_i(a^{best}) > u_i(a_i^k)$. For every object not in Z_i chosen in the 2^{nd} round or later, $u_i(a_i^l) > u_i(a_j^{l+1})$. Finally, $u_i(a^{best}) > u_i(a_{j\neq i}^1)$. Hence, the overall inequality holds and the allocation is Pro1.

ProX. Consider n agents with the same preferences, let agent i be the i^{th} in the picking order each round. The preferences are a strict order and the associated utilities are linear[2], that is, the vector of utilities of all elements in A is $(0, 2, \ldots, nk)$. At every round $l = 1, \ldots, k$, i^{th} agents picks the best remaining element with a corresponding utility $(k-l)n + (n-i+1)$[3]. Let us check if ProX holds for the last agent.

Consider agent k, his utility $u(Z_k)$ is:

$$0+(n+1)+\cdots+((k-1)n+1) = n(1+\cdots+(k-1))+k-1 = \frac{nk(k-1)}{2}+k-1$$

When considering ProX for this agent, exchange his item chosen in any but the last round with an item chosen by agent $(k-1)$ in the same round. This will increase his utility by the smallest possible amount 1,

[2]Except for the worst object's utility, which is valued at 0 rather than 1.

[3]The value of the worst object will not follow this formula.

hence, $u(Z_k^{'}) = \frac{nk(k-1)}{2} + k$. If ProX holds, then

$$\begin{aligned} u(Z_k^{'}) &\geq \frac{1}{n}u(A) \\ \frac{nk(k-1)}{2} + k &\geq \frac{1}{n}(\frac{nk(nk+1)}{2} - 1) \\ \frac{nk(k-1)}{2} + k &\geq \frac{1}{n}\frac{nk(nk+1)}{2} - \frac{1}{n} \\ nk(k-1) + 2k &\geq k(nk+1) - 2/n \\ nk^2 - nk + 2k &\geq nk^2 + k - 2/n \\ k &\geq nk - 2/n \end{aligned}$$

It follows, that ProX does not hold whenever $k < nk - 2/n$. This paper examines allocations with $k \geq 2$ and $n \geq 2$, the inequality holds for any pair of such k and n. Hence, the allocation is not ProX.

□

Next, we turn to the minimal guarantees. RR results in allocations that are mEF, but not lexEF.

Proposition 18. *RR is mEF, but not necessarily lexEF.*

Proof. For any agent i, an item chosen in the first round of RR is better than any item chosen in the last round, hence, mEF holds.

If all agents have identical preferences, then the allocation Z_j of the agent j who was choosing the items before agent i in each round will dominate Z_i, hence, the allocation will not be lexEF. □

In order to eliminate unfair advantage of the first chooser, RR can be generalised to a rule where the picking order is determined before every round of choosing. This does not guarantee that the order will be different every time, but it does shuffle the list in vast majority of cases. Despite this, gRR may still fail PO and lexEF and only guarantees the same properties as RR.

Proposition 19. *gRR is EF1, Pro1, mEF, but not necessarily PO or lexEF.*

Proof. EF1 Consider and agent i. Under gRR i will get a weakly better allocation than if he was the last person in the picking order under RR. In the worst case scenario, i will remain last in every round of gRR, but in most cases he will not pick last in every round. Being last under RR results in the worst possible allocation, but it is still EF1 because RR is EF1 (Proposition 16). Hence, gRR is EF1.

Pro1 Using the same logic, since under gRR an agent cannot receive a bundle that is worse than the allocation of an agent who picks last under RR, and RR is Pro1, so is gRR[4].

mEF Similarly to RR, for any agent i, an item chosen in the first round of gRR is better than any item chosen in the last round, hence, mEF holds.

PO Since the RR picking order is possible, efficiency is not guaranteed.

lexEF Let all agents have identical strict preferences over n best objects and be indifferent between the rest. Then, in the first round of gRR the n best objects will be chosen, and the choice of agent i will dominate the choice of agent j if i picks the object before j in the first round. Since all other objects are worse than the first n and have identical utility, the dominance will remain, hence, the allocation will not be lexEF.

□

We aggregate the above results to show that EF1, Pro1 and mEF are compatible and can be guaranteed.

Theorem 2. *EF1, Pro1 and mEF allocation always exists.*

Proof. Such allocation can be obtain by using RR or gRR. □

[4]Note that the argument that proves that RR is Pro1 does not use the fact that agents' picking order does not change. Hence, we can use the same argument here.

In order to obtain efficiency, we study LM, NM and UM. Despite being efficient, these rules do not guarantee that the resulting allocation is fair. Unfortunately, all of the fairness notions may fail to hold.

Proposition 20. *LM, NM and UM do not necessarily satisfy any of EF1, EFX, mEF, lexEF, Pro1 and ProX.*

Proof. Consider an example with agent i and $n-1$ identical agents j. There are $(n-1)k$ goods a and k goods b such that $u_i(a) = p, u_i(b) = q = u_j(a), u_j(b) = 0$, where $p > q$. Since LM maximizes the minimum utility and NM maximizes the product, the resulting allocation is $Z_i = \{b, b, ..., b\}$ and $Z_j = \{a, a, ..., a\}$ when using both mechanisms. Under UM, as we go from this allocation to one in which agent i and agent j performed a single exchange, the total utility increases by $\Delta U = (p-q) + (0-q) = p - 2q$. By making an additional assumption that $p < 2q$, we make sure that the same allocation is optimal under UM.

EF1 Note that when $k = 2$, any allocation is EF1. Let $k > 2$, check if the allocation is EF1:

$$\begin{aligned} u_i(Z_i + a - b) &> u_i(Z_j + b - a) \\ (k-1)q + p &> (k-1)p + q \\ (k-2)q &> (k-2)p \end{aligned}$$

Since $q < p$, the inequality above does not hold whenever $k > 2$, hence, the allocation is not EF1.

EFX Since all agents are allocated k units of the same good, EFX and EF1 become identical. It follows that EFX does not hold whenever $k > 2$.

mEF $\max_{x \in Z_i} u_i(x) = u_i(b) = q < \min_{x \in Z_j} u_i(x) = u_i(a) = p$, hence, mEF does not hold.

lexEF According to the preferences of agent i, every element in Z_j is

superior to every element in Z_i, that is, Z_j dominates Z_i, and lexEF does not hold.

Pro1 Pro1 holds whenever

$$\begin{aligned} u_i(Z_i + a - b) &> \frac{1}{n} u_i(A) \\ (k-1)q + p &> \frac{kq + (n-1)kp}{n} \\ n(k-1)q + np &> kq + (n-1)kp \\ (nk - n - k)q &> (nk - k - n)p \end{aligned}$$

Since $q < p$, the inequality above does not hold whenever $nk - n - k > 0$. By re-writing this condition, we find that Pro1 does not hold whenever $n > 1 + \frac{1}{k-1}$ or, alternatively, $k > 1 + \frac{1}{n-1}$. These are satisfied when $k > 2$ and $n \geq 2$ or $k \geq 2$ and $n > 2$. We conclude that Pro1 does not hold when $k \geq 2$, $n \geq 2$, and at least one of the inequalities is strict.

ProX Since all agents are allocated k units of the same good, ProX and Pro1 become identical. It follows that ProX does not hold when $k \geq 2$, $n \geq 2$, and at least one of the inequalities is strict.

□

2.8 Conclusion and Future Work

We examined a fair division model in which every agent is allocated equal number of goods. We adjusted fairness notions to suit our model and analysed relations between them. We also verified if allocations resulting from RR, gRR, LM, NM and UM satisfy the fairness and optimality conditions.

We found that allocating equal number of goods to the agents differs from a general case of fair division with indivisible goods. Most of the properties become logically independent, most notably, EF1 does not

imply Pro1. Allocation rules can also lose some of the desired properties. RR is fair up to one good but not efficient, while efficient LM, NM and UM are not fair. When two agents divide the goods, EF1 is equivalent to Pro1 and is compatible with efficiency.

There are several direction in which our model can be studied further. Firstly, there are alternative ways of defining fairness up to one good (as discussed in Section 2.4) as well as additional notions of minimal guarantees (for example, MMS). Secondly, combinations of our properties may imply others. Thirdly, we do not yet know if efficiency and our fairness notions are compatible when $n > 2$. The common allocation procedures are either efficient or fair, but this does not imply incompatibility. When considering the allocation rules, we assumed that the valuations were positive. We might get additional results for 'bads' instead of goods. Finally, we plan to consider the properties and allocation rules on specific preference domains. In this paper, the preferences are additive and otherwise arbitrary. We might be able to guarantee better results when the utilities are identical, identical ordinally but not cardinally or dichotomous.

2.9 Appendix. Summary of results

We summarise the results of the paper in the table below. *Yes* indicated that an allocation rule satisfies the property, i.e., the property holds for *all* allocations that result from the rule. If the rule does not guarantee that a property will hold for the resulting allocation, we write *No*.

	EF1	EFX	mEF	lexEF	Pro1	ProX	PO
EF1	$k = 2 \Rightarrow$ EF1 (8)						
EFX	EFX $\Rightarrow$ EF1 (by definition)						
mEF	mEF $\Rightarrow$ EF1, EF1 $\not\Rightarrow$ mEF when $k = 3$ (9) mEF $\not\Leftarrow$ EF1 when $k > 3$ (10)						
lexEF	lexEF $\Rightarrow$ EF1, EF1 $\not\Rightarrow$ lexEF when $k = 3$ (12) lexEF $\not\Leftarrow$ EF1 when $k > 3$ (12)		lexEF $\Rightarrow$ mEF, but mEF $\not\Rightarrow$ lexEF (11)				
Pro1	Pro1 $\Leftrightarrow$ EF1 when $n = 2$, $k \geq 3$ (13) Pro1 $\not\Leftarrow$ EF1 when $n \geq 3$, $k \geq 3$(14)						
ProX						ProX $\Rightarrow$ Pro1 (by definition)	
RR	Yes (16)	No (16)	Yes (18)	No (18)	Yes (17)	No (17)	No (15)
gRR	Yes (19)		Yes (19)	No (19)	Yes (19)		No (19)
LM	No (20)	No (20)	No (20)	No (20)	No (20)	No (20)	Yes
NM	No (20)	No (20)	No (20)	No (20)	No (20)	No (20)	Yes
UM	No (20)	No (20)	No (20)	No (20)	No (20)	No (20)	Yes

Table 2.3: Summary or results

Chapter 3

Student allocation to equal number of elective courses

Abstract

We consider many-to-many allocation of students to elective courses. Each student must attend equal number of courses, and all courses admit exact number of students. We show that DA mechanism may violate the quotas and design two modifications that ensure that the quotas are respected. We show that both modifications may create additional opportunities for manipulation. We simulate the admissions and show that, on average, the number of profitable manipulations increases by no more than 0.4%. When considering the number of manipulable markets, we find that roughly 20% and 18% of the markets are manipulable under our two modifications.

3.1 Introduction

Matching literature focuses on allocating goods to agents. Goods and agents are matched in one-sided or two-sided markets. One-sided markets imply that agents have preferences over goods, but the goods do not have preferences over agents and have no effect on the resulting allocation. In two-sided markets, both sides comprise disjoint sets of agents who have preferences over the agents on the other side of the market. The well-known marriage problem is set in a two-sided market with men and women on each side. They have preferences over each other, and the goal is to marry the largest number of couples in the 'best' way. In

the marriage problem, each man marries at most one woman, and each woman marries at most one man, hence, the one-to-one matchings are analysed. A natural extension of this problem considers one-to-many or many-to-many allocations. Such allocations are common in education and work environments. Consider a problem of matching professors to students for supervision, where each student is supervised by one professor, or allocating workers to shifts (or tasks) with each workers having multiple shifts and each shift consisting of multiple workers.

We study many-to-many matching in a two-sided market with identical sizes of the allocated sets. We base our study on a simple example: college students choose elective courses. All students have to study the same number of electives in a semester in order to complete a degree, and the number of seats in each course is limited. Similar problem arises when allocating workers to tasks. Each full-time employee works the same number of hours per week, and each task requires exact number of hours to be completed. We therefore want to make sure that the employees work to the full capacity, and that the tasks are completed efficiently. We study the outcomes of applying the deferred acceptance (DA) mechanism to such problems.

In their paper, Gale and Shapley [20] introduced the DA allocation mechanism and showed that it has two desirable properties: 1) it results in a stable matching and 2) proposers in the DA mechanism receive the best possible stable allocation. This strong positive result holds in a marriage problem when men and women submit their full list of strict preferences omitting only those women and men, respectively, whom they find unacceptable. The central planner then uses the proposed rule to create pairs (man, woman). Every person is assigned no more than one partner. Stability states that there is no pair who prefer to be together rather than with the people they were assigned to. Existence of stable matching and minimal guarantee sparked extensive research into the properties of the DA and other allocation rules in different markets.

One of the natural generalisations of the original DA mechanism applies to many-to-one or many-to-many markets. In such markets stable *and* strategyproof mechanisms do not exist, hence, the research explores specific domains where both properties do hold or studies the degree to which stability or strategy-proofness is violated. In this paper, we

follow the latter path. We show that the standard DA mechanism cannot always be used when each students is assigned an exact number of courses. We develop two modifications of the procedure that satisfy the requirements of our market and investigate the strategy-proofness of those modifications.

The paper is structured as follows: Section 3.2 discusses the related literature, Section 3.3 formally introduces the model and algorithms under consideration. Section 3.4 presents our results and Section 3.5 concludes.

3.2 Literature

Existence of a simple mechanism that guarantees stable allocation inspired research into assignment mechanisms that satisfy certain properties. The positive result of Gale and Shapley [20] was eventually followed by a negative result by Roth [39]: in two-sided one-to-one matching markets no allocation is both stable and strategyproof. A few years later, Roth [38] proved that one-to-many college admission problem is not the same as the marriage problem. Stability and truth-telling are still not compatible, but also there may be an allocation that all colleges prefer to college-optimal stable outcome.

Although telling the truth is not necessarily a dominant strategy, Roth and Peranson [40] show that in one-to-many matching markets there are not many opportunities for manipulation. Immorlica and Mahdian [23] observe similar results in markets with one-to-one matching showing that the truth-telling is a dominant strategy with high probability when other agents report their preferences truthfully. Kojima and Pathak [26] highlight that colleges may manipulate the admissions by misrepresenting either their preferences or quotas. The latter is not an issue in one-to-one matching markets as all quotas equal one. They investigate not only the frequency of existence of profitable manipulations in the one-to-many allocations, but also examine equilibrium strategies. They show that in large market truthful reports are an approximate equilibrium. All of the above results apply when the submitted preference lists are restricted. Under the complete preferences the results do not necessarily hold. In our paper, students report their full preferences,

hence, manipulations may be more frequent.

Aziz et al. [4] show that when all participants submit full preference lists, DA mechanism cannot be manipulated by the proposers with unit capacity in one-to-many (or one-to-one) markets. If the capacity of the proposer exceeds one, finding manipulations is FTP-complex. They simulate the student-proposing and college-proposing DA allocations and find that, on average, colleges can manipulate the allocations in roughly 37% and 70% of the markets, respectively.

Biro et al. [7] consider college admissions with lower quotas. Based on Hungarian system, where a programme may not open if there are not enough students, they require that the number of allocated students is between a lower and an upper bounds. They show that stable one-to-many matching may not exist and that the problem of verifying the existence of such allocation is NP-complete.

Many-to-many markets introduce additional challenges. In contrast to one-to-one markets, we need to be able to compare groups of allocated agents rather than individual agents. Given complete preferences over individual items, responsive preferences [38] imply that if two groups differ in one item, then the group with the preferred item is the better one. Substitutable preferences are a generalisation of responsive preferences. Under substitutable preferences, an agent that is chosen from a bigger set must also be chosen from a smaller set of available items. Roth [37] analysed stability under the two types of preferences. He showed that in many-to-two markets with substitutable (and, hence, responsive) preferences set of stable allocations is not empty. However, when the preferences are responsive the stable allocations may not be in the core or be Pareto optimal.

In our paper we consider many-to-many markets, and allocate exact number of students to exact number of courses. Effectively, we have a market in which lower and upper quotas coincide. Our work in close to Aziz et al. [4] as we too analyse performance of DA mechanism under responsive preferences. We, however, examine a different type of market and find that manipulations are more rare.

3.3 Model

There is a set of students $S = \{s_1, ..., s_n\}$, and a set of courses $C = \{c_1, \ldots, c_m\}$. Each course has a quota and can only admit up to q students. Each student must study k courses in order to graduate, hence, we only consider allocations in which each student is allocated to exactly[1] k courses. We assume that $nk = mq$, in other words, all courses are filled to their full capacity once the allocation is completed.

Each student has preferences $P(s_i) = P_i$ over the courses, each course has preferences $P(c_j) = P_j$ over students. Since each student is allocated to multiple courses, and each course admits multiple students, we need to extend the preferences to compare *sets* of courses and students respectively. We assume that preferences are *responsive*, namely, if two sets of courses (students) differ in one element, then the set containing a preferred course (student) is the preferred one.

All students and courses find each other acceptable, that is, no student would rather stay unmatched than take a course and vice versa.

3.3.1 Why we cannot use original DA

Before discussing the allocations resulting from the student-proposing DA algorithm, let us formally define it in the many-to-many markets.

DA

Step 1. Each student applies to his top k choices in P_i. Colleges put the best q applicants on the waiting lists of each course $j \in C$ and reject the rest. If there are less than q applicants, then all of them are put on the waiting list of the course, and no one is rejected.

⋮

Step m. Each student that was rejected at the previous step applies to his next t choices in P_i, where t is the number of courses the student was rejected from. Student may apply to less courses if

[1] This assumption is in line with standard practices as students need to gain a fixed number of academic credits (can be converted to number of hours taught) to graduate, and only that number of credits is covered by the tuition fee. Hence, students are generally not allowed to take additional courses.

there are not enough courses left on the preference list. For each course j, the best to date q applicants are put on the waiting list and the rest are rejected.

⋮

Termination. The algorithm terminates when no new offers are made. All students on the waiting lists are admitted in the respective courses.

Applying the deferred acceptance mechanism does not necessarily yield the desired results. The final allocation may leave some students allocated to less than k courses. Consider an example with three students and three courses, where each student needs to study two courses and each course admits two students.

Example 9. *Let* $N = \{s_1, s_2, s_3\}$ *and* $C = \{c_1, c_2, c_3\}$, $k = q = 2$. *We list preferences of the students and courses below in a descending order:*

Students	$P(s_1) : c_1, c_2, c_3$	$P(s_2) : c_1, c_2, c_3$	$P(s_3) : c_2, c_1, c_3$
Courses	$P(c_1) : s_1, s_2, s_3$	$P(c_2) : s_1, s_2, s_3$	$P(c_2) : s_3, s_1, s_2$

At the first step of the DA mechanism, all students apply to their top choices. Below, we show courses which put students on the waiting lists in green, the rejections are shown in red and in black are the courses that students did not apply to at the current step.

$$s_1 : c_1, c_2, c_3$$

$$s_2 : c_1, c_2, c_3$$

$$s_3 : c_2, c_1, c_3$$

The third student was rejected from both courses that he applied to. Next, he applies to the third course and is admitted.

$$s_1 : c_1, c_2, c_3$$

$$s_2 : c_1, c_2, c_3$$

$$s_3 : c_2, c_1, c_3$$

At this point students do not make any new applications and the procedure terminates. Clearly, the third student is only admitted into the

third course. We could also say that he is admitted twice by the same course. This happens because the original DA algorithm was aimed at one-to-one matching. We have a many-to-many set up in which the courses cannot be repeated in any of the students' allocations.

To make sure that each student is allocated to k courses and never twice to the same course, we need to modify the algorithm. Re-allocation of each student i who was not assigned to enough courses will be at the expense of the students who ranked higher than i in the courses that i did not get.

3.3.2 Modified DA

We want to preserve the properties of the DA while making sure that the students are allocated to exactly k courses. To that end, we start by allocating students using the original DA mechanism, and then alter the resulting allocation only if it does not satisfy our requirements.

We call the courses that are not filled to their full capacity and the students who were allocated to less than k courses *underdemanded*. We denote the set of underdemanded courses by C^U and the set of underdemanded students by S^U. We call the complements of these sets *demanded* and denote them by $C^D = C \setminus C^U$ and $S^D = S \setminus S^U$.

We consider two modifications of the the original DA. Both use priority lists of students and alter DA in the spirit of serial dictatorship. Whenever a student is allocated to less than k courses, he can only be matched to an additional course if a student on that course gives up his seat. Our first modification bans a student from attending a demanded course and re-runs the DA. As a result, a spot appears on one of the demanded courses that an underdemanded students takes. Our second modification forces a demanded student to attend one of the underdemanded courses. As a result, such student can now apply to less courses (as there is no need to apply to the course he is forced to attend, but k is still the same), and one seat at a demanded course becomes vacant. In both procedures, we need to decide which student(s) will be forced to vacate his (their) seat(s). We do this based on priority. We assume that there is an exogenous priority list which orders the students from low to high priority. Demanded student with lowest priority will be the

first to give up his seat.

Define the first modification of the DA procedure.

ModDA1

PART 1. Run the DA algorithm. Call the existing allocation μ.

PART 2. If $|\mu(s_i)| = k$ for every $s_i \in S$, then terminate. If not, repeat the following steps until the modified allocation μ' is s.t. $|\mu'(s_i)| = k$ for every $s_i \in S$:

Stage 1. Denote the current allocation by μ. Pick a student according to the picking order (described below). Ban this student from his least preferred demanded course in $\mu(s)$, which did not admit all underdemanded students, $S^U \not\subset \mu(c)$.

Stage 2. Run the DA algorithm. Students are not allowed to apply to courses they were previously banned from. Call the updated allocation μ'. Update the sets of underdemanded students and courses.

Note that this modification prevents student(s) from applying to certain courses. Banning may cause a chain reaction, and the banned student will not necessarily end up with an underdemanded course.

Our second modification does not ban students from applying. Instead, it forces the demanded students to pick underdemanded courses.

ModDA2

PART 1. Run the DA algorithm. Call the existing allocation μ.

PART 2. If $|\mu(s_i)| = k$ for every $s_i \in S$, then terminate. If not, repeat the following steps until the modified allocation μ' is s.t. $|\mu'(s_i)| = k$ for every $s_i \in S$:

Stage 1. Denote the current allocation by μ. Pick a student according to the picking order (described below). Force this student to attend an underdemanded course that he was not assigned to and likes the most.

Stage 2. Run the DA algorithm. Students have to apply to the courses they were forced to attend. Courses cannot reject such students. Update the sets of underdemanded students and courses.

To decrease a number of underdemanded students and underdemanded courses, we pick a student who has to vacate a seat at a demanded course or has to attend an underdemanded course, which in turn leads to him giving up a seat at a demanded course.

Picking order

We pick a student who:

- was allocated to k courses at the previous iteration of the algorithm, that is, $s \in S^D$
- is not assigned to all underdemanded courses, $C^U \cap \mu(s) \neq C^U$
- has the *lowest priority* on the priority list among all students who satisfy the first two criteria

Priority list can have a significant effect on strategy-proofness. We find that the same manipulation may be profitable given one priority list, but harmful when the priorities are different. In our paper, we assume that the priorities are fixed and exogenous. We expect that endogenous priorities may be useful in reducing manipulabity of the allocation procedures, however, we do not study endogenous priorities in this paper.

The DA mechanism and its modifications are well-defined[2] and deterministic, hence, they result in a unique allocation for any given allocation problem[3].

3.4 Strategy proofness

First, we analyse the underdemanded students. These students are least preferred by the courses compared to their peers. This affects their power when trying to affect the outcome of the allocation procedure.

Lemma 1. *Every underdemanded student has been rejected from all the courses that he is not allocated to under DA.*

[2]We can always compare individual courses (students) according to student's (course's) preferences.

[3]An allocation problem is determined by n, m, p, q, students' and colleges' preferences over each other and the exogenous list of priorities

Proof. The DA assignment rule does not terminate until there are no more new application. A student s who has been tentatively accepted by less than k courses keeps applying to the courses in his preference list until: 1) he is put on a waiting list by k courses OR 2) he has applied to all courses on the preference list. Since we assume that students have complete preferences, underdemanded student s applied to all programmes by the time the procedure terminated, hence, he must have been rejected by all the courses that did not admit him. □

Proposition 21. *Under DA, each underdemanded student is allocated to all underdemanded courses.*

Proof. A course only rejects a student if the number of applications to the course exceeds the quota. Underdemanded courses do not reach the quota by the end of the allocation process, hence, each student who applies to the course is accepted. According to Lemma 1, each underdemanded student has been rejected by all the courses he is not assigned to. An underdemanded course cannot reject students, therefore, each underdemanded student is assigned to all underdemanded courses. □

Above propositions imply that the number of underdemanded students and courses cannot be too large.

Proposition 22. *Under DA, the number of underdemanded courses is less than k, and the number of underdemanded students is less than q, that is, $|C^U| < k$ and $|S^U| < q$.*

Proof. Since every underdemanded student is assigned to all underdemanded courses, $|S^U| \geq k$ would imply that the student is not underdemanded. Same logic applies if the number of underdemanded students exceeds $q - 1$. □

In our paper, we focus on manipulability. We start by showing that underdemanded students cannot affect the DA. This result is known in the literature.

Proposition 23. *Reports of underdemanded students do not affect the outcome of the DA mechanism.*

Proof. In their paper, Klijn and Yazıcı [25] examine the Rural Hospital Theorem in the many-to-many setup. They show that the underdemanded students are allocated to the same set of courses in any stable matching given substitutable and weakly separable preferences. Responsive preferences are a subset of these, and DA mechanism results in a stable matching, hence, underdemanded students are assigned to the same set of courses regardless of their reports. □

In contrast to underdemanded students, demanded students may be able manipulate the DA matching by misreporting their preferences.

When it comes to modifications, sets of underdemanded courses and students are updated every time DA is re-run. In presence of multiple underdemanded students, one becomes demanded faster than the rest. This implies that initially underdemanded students may be able to manipulate the modified algorithms at some point.

Proposition 24. *Demanded students may be able to manipulate the DA allocation.*

Proof. Consider an example with $n = 4$, $m = 4$, $k = 2$ and $q = 2$. Let students and colleges have the preferences described below:

Students' preferences	
s_1	c_1, c_2, c_4, c_3
s_2	c_1, c_2, c_3, c_4
s_3	c_3, c_4, c_1, c_2
s_4	c_2, c_3, c_1, c_4

Courses' preferences	
c_1	s_2, s_4, s_1, s_3
c_2	s_1, s_4, s_2, s_3
c_3	s_2, s_3, s_4, s_1
c_4	any

At each step of the DA mechanism, students apply to their top two courses that they have not been rejected from. The list of applicants to each course is shown in the columns of the table below. When the number of applications to a course exceeds two, the course rejects the least preferred applicants keeping two on the waiting list. The students who were rejected apply to the next course(s) on their preference list. We summarise the application process below (rejections are highlighted in red, new applications are blue unless they are immediately rejected):

DA	c_1	c_2	c_3	c_4
Step 1	s_1, s_2	s_1, s_2, s_4	s_3, s_4	s_3
Step 2	s_1, s_2	s_1, s_4	s_2, s_3, s_4	s_3
Step 3	s_1, s_2, s_4	s_1, s_4	s_2, s_3	s_3
Step 4	s_2, s_4	s_1, s_4	s_2, s_3	s_1, s_3

The resulting matching then is:

	s_1	s_2	s_3	s_4
Students' allocations	c_2, c_4	c_1, c_3	c_3, c_4	c_1, c_2

We observe that student s_1 is matched to courses c_2 and c_4. However, his application to c_2 starts a chain reaction which results in him losing a seat at his most preferred course c_1. If instead student s_1 applies to c_1 and c_4 from the start, there will be no rejections, and the resulting allocation will be an improvement for s_1. The reported preferences of student s_1 are then, for example,

$$P'(s_1) : c_1, c_4, c_2, c_3,$$

and the resulting DA procedure terminates after the first step:

DA manipulation	c_1	c_2	c_3	c_4
Step 1	s_1, s_2	s_2, s_4	s_3, s_4	s_1, s_3

By misreporting his preferences, student s_1 improved his allocation. □

When introducing the modifications to the allocation procedure, we may affect the manipulability. When studying the possible manipulations, we are mostly interested in the existence of the manipulations that are beneficial:

only under DA. Such manipulations suggest that the modification of the initial algorithm removes some manipulation opportunities, hence, inducing more honest reporting.

only under ModDA. This would suggest that the modification is open to new manipulations compared to the original DA.

Proposition 25. *Both ModDA1 and ModDA2 may make some manipulations that were profitable under DA ineffective or harmful to the manipulator, hence, modifying DA may decrease the number of profitable manipulations.*

Proof. We will demonstrate this using an example with $n = 5$, $m = 5$, $k = 2$ and $q = 2$. Let student s_1 be the last on the priority list and let the preferences be as shown below:

Students' preferences	
s_1	c_1, c_2, c_3, c_4, c_5
s_2	c_2, c_3, c_4, c_1, c_5
s_3	c_1, c_4, c_5, c_1, c_2
s_4	c_2, c_4, c_1, c_3, c_5
s_5	c_3, c_4, c_2, c_1, c_5

Courses' preferences	
c_1	s_2, s_3, s_4, s_1, s_5
c_2	s_1, s_4, s_2, s_3, s_5
c_3	s_1, s_2, s_3, s_4, s_5
c_4	s_1, s_2, s_3, s_4, s_5
c_5	s_1, s_2, s_3, s_4, s_5

First we run DA algorithm. It ends with one of the students being allocated to only one course, so we implement both modifications keeping in mind that s_1 has the lowest priority, hence, is chosen by both modifications to give up a seat.

DA	c_1	c_2	c_3	c_4	c_5
Step 1	s_1, s_3	s_1, s_2, s_4	s_2, s_5	s_3, s_4, s_5	
Step 2	s_1, s_3	s_1, s_4, s_5	s_2, s_5	s_2, s_3, s_4	
Step 3	s_1, s_3, s_4, s_5	s_1, s_4	s_2, s_5	s_2, s_3	
Step 4	s_3, s_4	s_1, s_4	s_1, s_2, s_5	s_2, s_3	s_5
Outcome	s_3, s_4	s_1, s_4	s_1, s_2	s_2, s_3	

ModDA1 prevents s_1 from applying to the least preferred course that he is currently allocated to, c_3. This means that Steps 1-3 after the ban are the same as under DA, at Step 4 s_1 applies to c_4 instead of c_3.

ModDA1	c_1	c_2	c_3	c_4	c_5
Step 1	s_1, s_3	s_1, s_2, s_4	s_2, s_3	s_3, s_4, s_5	
Step 2	s_1, s_3	s_1, s_4, s_5	s_2, s_5	s_2, s_3, s_4	
Step 3	s_1, s_3, s_4, s_5	s_1, s_4	s_2, s_5	s_2, s_3	
Step 4	s_3, s_4	s_1, s_4	s_2, s_5	s_1, s_2, s_3	s_5
Step 5	s_3, s_4	s_1, s_4	s_1, s_2	s_2, s_3	s_1, s_5

ModDA2 forces s_1 to apply to the the underdemanded course c_5. As a result, s_1 is unable ao apply to s_2.

ModDA2	c_1	c_2	c_3	c_4	c_5
Step 1	s_1, s_3	s_2, s_4	s_2, s_3	s_3, s_4, s_5	s_1
Step 2	s_1, s_3	s_2, s_4, s_5	s_2, s_3	s_3, s_4, s_5	s_1
Step 3	s_1, s_3, s_5	s_2, s_4	s_2, s_3	s_3, s_4, s_5	s_1
Step 4	s_1, s_3	s_2, s_4	s_2, s_3	s_3, s_4, s_5	s_1, s_5

Now assume that s_1 decides to manipulate the DA mechanism. By applying to c_2, he lost a seat at his favourite course c_1. To avoid this, he reports c_3 as his favourite keeping the rest of the preferences the same:

$$P'(s_1) : c_3, c_1, c_2, c_3, c_4.$$

As a result, both modification will prevent s_1 from attending c_1 as shown below.

DA with manipulation	c_1	c_2	c_3	c_4	c_5
Step 1	s_1, s_3	s_2, s_4	s_1, s_2, s_5	s_3, s_4, s_5	
Step 2	s_1, s_3, s_5	s_2, s_4, s_5	s_1, s_2	s_3, s_4	
Step 3	s_1, s_3	s_2, s_4	s_1, s_2	s_3, s_4	s_5

ModDA1 with manipulation	c_1	c_2	c_3	c_4	c_5
Step 1	s_1, s_3	s_1, s_2, s_4	s_2, s_5	s_3, s_4, s_5	
Step 2	s_1, s_3	s_1, s_4, s_5	s_2, s_5	s_2, s_3, s_4	
Step 3	s_1, s_3, s_4, s_5	s_1, s_4	s_2, s_5	s_2, s_3	
Step 4	s_3, s_4	s_1, s_4	s_2, s_5	s_1, s_2, s_3	s_5
Step 4	s_3, s_4	s_1, s_4	s_2, s_5	s_1, s_2	s_3, s_5

ModDA2 with manipulation	c_1	c_2	c_3	c_4	c_5
Step 1	s_3	s_2, s_4	s_1, s_2, s_5	s_3, s_4, s_5	s_1
Step 2	s_3, s_5	s_2, s_4, s_5	s_1, s_2	s_3, s_4	s_1
Step 3	s_3, s_5	s_2, s_4	s_1, s_2	s_3, s_4	s_1, s_5

Now we compare the outcomes of the allocation procedures. Clearly, a manipulation that improved the allocation of s_1 when the courses

were allocated with DA procedure is not effective when the algorithm is modified; ModDA1 results in the same allocation regardless of the manipulation, and ModDA2 makes it harmful to s_1:

Outcome comparison	Allocation of s_1	Manipulation result
DA	c_2, c_3	
DA with manipulation	c_1, c_3	Beneficial
ModDA1	c_2, c_4	
ModDA1 with manipulation	c_2, c_4	Ineffective
ModDA2	c_1, c_5	
ModDA2 with manipulation	c_3, c_5	Harmful

□

The above proposition suggests that it may be possible to modify the DA in such a way that the resulting outcome induces more thruth telling. We now consider both aforementioned modificaitons.

3.4.1 ModDA1

We start by showing that the first modifications, ModDA1, allows for beneficial manipulations that do not improve the outcome of the manipulator under DA.

Proposition 26. *There are manipulations that are profitable under ModDA1 but not under DA, hence, modifying DA to ModDA1 may increase the number of profitable manipulations.*

Proof. We show this using an example with $n = 4$, $m = 4$, $k = 2$ and $q = 2$. Let students and colleges have the preferences described below:

Students' preferences	
s_1	c_1, c_2, c_3, c_4
s_2	c_1, c_3, c_2, c_4
s_3	c_2, c_3, c_1, c_4
s_4	c_4, c_2, c_1, c_3

Courses' preferences	
c_1	s_2, s_3, s_1, s_4
c_2	s_1, s_3, s_4, s_2
c_3	s_2, s_3, s_4, s_1
c_4	any

First run the DA procedure:

DA	c_1	c_2	c_3	c_4
Step 1	s_1, s_2	s_1, s_3, s_4	s_2, s_3	s_4
Step 2	s_1, s_2, s_4	s_1, s_3	s_2, s_3	s_4
Step 3	s_1, s_2	s_1, s_3	s_2, s_3, s_4	s_4

Student s_4 is rejected from all courses but c_4. We therefore need to apply the modification of DA to make sure that all students are allocated to exactly two courses. Assume that s_3 has the lowest priority. He was allocated to courses c_2 and c_3, and he is banned from the course that he prefers least out of the two. Hence, under the modification s_3 is banned from applying to c_3. Given the ban, we run the DA algorithm again:

ModDA1, s_3 has lowest priority	c_1	c_2	c_3	c_4
Step 1	s_1, s_2, s_3	s_1, s_3, s_4	s_2	s_4
Step 2	s_2, s_3, s_4	s_1, s_3	s_1, s_2	s_4
Step 3	s_2, s_3	s_1, s_3	s_1, s_2, s_4	s_4
Step 4	s_2, s_3	s_1, s_3	s_2, s_4	s_1, s_4

Now consider a manipulation in which the first student submits the following preferences:

$$P'(s_1) : c_1, c_4, c_2, c_3.$$

The DA mechanism terminates in one step:

DA manipulation	c_1	c_2	c_3	c_4
Step 1	s_1, s_2	s_3, s_4	s_2, s_3	s_1, s_4

Note that since the DA results in all students receiving exactly two courses to study, the modification is not going to be used given the manipulation.

Now compare the outcomes of the DA and modified DA when the students tells the truth as well as manipulates:

	s_1	s_2	s_3	s_4
DA allocation	c_1, c_2	c_1, c_3	c_2, c_3	c_4
ModDA1 allocation	c_2, c_4	c_1, c_3	c_1, c_2	c_3, c_4
DA and ModDA1 allocation with manipulation	c_1, c_4	c_1, c_3	c_2, c_3	c_2, c_4

Clearly, s_1 has no incentive to misrepresent his preferences when DA is used since under DA he receives his top two choices. Under ModDA1, s_3 loses his seat at c_3 and applies to c_1. Course c_1 finds s_1 least desirable at this step, and s_1 is rejected by his favourite course. In order to prevent this from happening when all students get assigned to the same number of courses, s_1 should give up c_2. Then, he will be able to study c_1.

Consider different priority orders. Even if student s_4 has the lowest priority, he will not be banned from any of the courses, so we only have two cases to consider.

When s_1 has the lowest priority, he is banned from c_2. He therefore applies to c_1 and c_3 at the start of ModDA1:

ModDA1, s_1 has lowest priority	c_1	c_2	c_3	c_4
Step 1	s_1, s_2	s_3, s_4	s_1, s_2, s_3	s_4
Step 2	s_1, s_2	s_3, s_4	s_2, s_3	s_1, s_4

If s_2 has the lowest priority, then he is banned from c_3 and applies to c_1 instead:

ModDA1, s_2 has lowest priority	c_1	c_2	c_3	c_4
Step 1	s_1, s_2	s_1, s_3, s_4	s_3	s_2, s_4
Step 2	s_1, s_2, s_4	s_1, s_3	s_3	s_2, s_4
Step 3	s_1, s_2	s_1, s_3	s_3, s_4	s_2, s_4

When s_1 has the lowest priority, he receives the same allocation as when he manipulates. However, when s_2 has the lowest priority, s_1 is matched to the top two courses according to his preferences. Manipulating in this case would be harmful.

□

Corollary 2. *A manipulation may improve the allocation of an agent given a fixed priority order, but harm him as priorities change.*

Given the priority list, ModDA1 may create additional ways of manipulating the outcome of the allocation. Hence, we consider the second modification.

3.4.2 ModDA2

In order to decrease the number of manipulations, we study another modification of the DA algorithm.

Proposition 27. *There are manipulations that are profitable under ModDA2 but not under DA, hence, modifying DA to ModDA2 may increase the number of profitable manipulations.*

Proof. Let $n = 4$, $m = 4$, $k = 2$ and $q = 2$. Let students and colleges have the preferences described below:

Students' preferences	
s_1	c_1, c_2, c_3, c_4
s_2	c_2, c_4, c_1, c_3
s_3	c_2, c_1, c_3, c_4
s_4	c_3, c_4, c_1, c_2

Courses' preferences	
c_1	s_3, s_2, s_1, s_4
c_2	s_1, s_3, s_4, s_2
c_3	s_3, s_2, s_4, s_1
c_4	s_2, s_1, s_4, s_3

We start by running the DA algorithm:

DA	c_1	c_2	c_3	c_4
Step 1	s_1, s_3	s_1, s_2, s_3	s_4	s_2, s_4
Step 2	s_1, s_2, s_3	s_1, s_3	s_4	s_2, s_4
Step 3	s_2, s_3	s_1, s_3	s_1, s_4	s_2, s_4

Since all students are allocated to exactly two courses, DA procedure terminates without calling upon the modification. Student s_1 receives his second and third choices, and he realises that there is a possible manipulation when s_2 is the lowest on the priority list. Let student s_1 report his preferences as

$$P'(s_1) : c_1, c_2, c_4, c_3.$$

Given the new reported preferences, DA will result in:

DA	c_1	c_2	c_3	c_4
Step 1	s_1, s_3	s_1, s_2, s_3	s_4	s_2, s_4
Step 2	s_1, s_2, s_3	s_1, s_3	s_4	s_2, s_4
Step 3	s_2, s_3	s_1, s_3	s_4	s_1, s_2, s_4
Step 4	s_2, s_3, s_4	s_1, s_3	s_4	s_1, s_2
Step 5	s_2, s_3	s_1, s_3, s_4	s_4	s_1, s_2

The manipulation forces the fourth student out of the second course. Since s_4 is low in preferences of other courses, he is allocated to only one course. We then need to use the modification to resolve this issue. Given the above assumption about s_2 having the lowest priority, we force him to attend c_3. We will indicate the students who cannot be removed from a course by using green colour:

ModDA2 manipulation, s_2 has lowest priority	c_1	c_2	c_3	c_4
Step 1	s_1, s_3	s_1, s_2, s_3	s_2, s_4	s_4
Step 2	s_1, s_3	s_1, s_3	s_2, s_4	s_2, s_4

When s_1 or s_3 have the lowest priority, the resulting allocation also improves as a result of manipulation:

ModDA2 manipulation, s_1 has lowest priority	c_1	c_2	c_3	c_4
Step 1	s_1, s_3	s_2, s_3	s_1, s_4	s_2, s_4

ModDA2 manipulation, s_3 has lowest priority	c_1	c_2	c_3	c_4
Step 1	s_1	s_1, s_2, s_3	s_3, s_4	s_2, s_4
Step 2	s_1, s_2	s_1, s_3	s_3, s_4	s_2, s_4

We are not considering the case when s_4 has the lowest priority because it will be the same as one of the above. Student s_4 is underdemanded, therefore, he does not fit the criteria according to which a student is chosen and forced to attend an underdemanded course. Student who is forced to attend c_4 will be chosen out of $\{s_1, s_2, s_3\}$.

Under DA, the manipulation worsens the outcome of student s_1. However, under ModDA2, s_1 makes sure that the modification is implemented. As a result, his allocation improves regardless of the priorities. □

In this section, we find that if the mechanism of course allocations is changed from DA to ModDA1 or ModDA2, the number of profitable manipulations may increase or decrease. In order to see how many manipulations are profitable and how often they arise, we run simulations.

3.4.3 Simulations

We showed that both our modifications may increase manipulability. We now analyse the frequency with which students can manipulate the allocation before and after the modification of the procedure. We generated the preferences randomly, each preference profile being equally likely. We then checked if any of the students can improve his allocation by reporting any preference profile other than the truthful one. We ran 1000 iterations of the simulations when $n = m = 4$ and $k = q = 2$ to examine the average values across different markets with different agents.

We consider all possible untruthful reports of each agent in the market. We find that a very small number of misreported preferences result in an improvement for the agent. Compared to markets where each agent may receive a different numbers of goods, we have a strict requirement that each student is admitted into k courses. This implies that the manipulator cannot receive an extra good, rather a manipulation will result in an exchange.

We study both ModDA1 and ModDA2, and find that the number of profitable misrepresentations of the true preferences does not exceed 20% and 15%, respectively. Under ModDA1, between 0% and 5.4% of the manipulations become ineffective compared to DA, but up to 15% new manipulations become available. On average, 0.15% of all possible manipulation stop working and number of additional manipulations is about 0.35%. Under ModDA2, only 0.07% of all misrepresentations become beneficial.

The results of our simulations are summarised in the tables below. For each simulation, we compute the number of manipulations that are beneficial under both DA and a modification, one of the allocation rules and none of them. We then compute average and median number of possible profitable manipulations to see how manipulable the algorithms are 'on average'. We also record the minimum and maximum number of benefial manipulations to show the results for the best and worst case scenarios. The extremum values arise from individual (hence, different) simulations. Maximum values show the biggest number of profitable manipulations under DA, modification or both. The minimum number of manipulations that are not beneficial allows us to find the upper

bound for the number of manipulations that are profitable in at least on the assignment rules. We present these results as frequencies rather than the number of manipulations.

Misrepresentenation of preferences	Average	Median	Min	Max
Beneficial under DA, but not ModDA1	0.15%	0%	0%	5.4%
Beneficial under both DA and ModDA1	1%	0%	0%	11.9%
Beneficial under ModDA1, but not DA	0.35%	0%	0%	15.2%
Not beneficial under both DA and ModDA1	98.5%	100%	80.4%	100%
Total	100%	100%		

Table 3.1: Fraction of beneficial misreports among all misreports under DA and ModDA1

Misrepresentenation of preferences	Average	Median	Min	Max
Beneficial under DA, but not ModDA2	0.004%	0%	0%	4.2%
Beneficial under both DA and ModDA2	1.3%	0%	0%	14.6%
Beneficial under ModDA2, but not DA	0.07%	0%	0%	8.3%
Not beneficial under both DA and ModDA2	98.6%	100%	85.4%	100%
Total	100%	100%		

Table 3.2: Fraction of beneficial misreports among all misreports under DA and ModDA2

Although the number of beneficial manipulations is small, there is a considerable number of markets that at least one student is motivated and capable of manipulating. When considering the number of markets in which profitable manipulations are possible, we find that roughly 20% are manipulable under ModDA1 and about 18% under ModDA2.

Our results suggest that ModDA2 is a better modification as it allows for less additional beneficial manipulations compared to ModDA1 and it exposes less markets to manipulating. Intuitively, this is something we could expect. ModDA1 bans a student from attending one of the courses, but allows him to apply to other courses that were not underdemanded after the last run or re-run of DA. Hence, the student may try to get admitted into a course that was not underdemanded at

the previous stage and force someone else to take the worse alternative. However, in ModDA2 a popular student is forcefully allocated to an underdemanded course. A pair of (forced student, forced course) is created prior to the re-run of DA and cannot be changed, reducing the number of new beneficial manipulations.

3.5 Conclusions and Future Work

We consider a problem of matching student to elective courses. Each student must be allocated to exact number of courses and each course has equal number of seats. The deferred acceptance mechanism (DA) may result in a student being allocated to the same course multiple times, hence, we develop two modifications and study their strategy-proofness.

The modifications are based on a priority list which specifies which students have to give up their seats in popular courses so that underdemanded students could attend enough classes. We assume that the priority list is exogenous, however, we show that the priorities have an affect on manipulations. The two modifications show roughly the same results in simulations: the average increase in the number of effective manipulations is very small. Our second modifications performs better in reducing the number of markets in which submitting non-truthful preferences may be beneficial.

Future work includes multiple directions. Firstly, we plan to run simulations with larger markets. The biggest challenge in this regard is to balance the market size and computational requirements. Secondly, we are going to examine the possibility of reducing manipulability by making the priority list endogenous. On the one hand, this would imply that the student might be able to affect not only the matching, but the list as well. On the other, it may be possible to design it as to reduce the number of manipulable markets.

Bibliography

[1] Atila Abdulkadiroglu and Tayfun Sönmez. "School choice: A mechanism design approach". In: *The American Economic Review* 93.3 (2003), pp. 729–747.

[2] Atila Abdulkadiroğlu, Yeon-Koo Che, and Yosuke Yasuda. "Resolving conflicting preferences in school choice: The "Boston mechanism" reconsidered". In: *The American Economic Review* 101.1 (2011), pp. 399–410.

[3] Eduardo M Azevedo and Eric Budish. "Strategy-proofness in the large". In: *The Review of Economic Studies* 86.1 (2019), pp. 81–116.

[4] Haris Aziz, Hans Georg Seedig, and Jana Karina von Wedel. "On the susceptibility of the deferred acceptance algorithm". In: *arXiv preprint arXiv:1502.06318* (2015).

[5] Michel Balinski and Tayfun Sönmez. "A tale of two mechanisms: student placement". In: *Journal of Economic theory* 84.1 (1999), pp. 73–94.

[6] Péter Biró. "Student admissions in Hungary as Gale and Shapley envisaged". In: *University of Glasgow Technical Report TR-2008-291* (2008).

[7] Péter Biró et al. "The college admissions problem with lower and common quotas". In: *Theoretical Computer Science* 411.34-36 (2010), pp. 3136–3153.

[8] Colleage Board. "SAT Subject Tests Percentile Ranks: 2017–2019 Graduating Classes". In: *https://secure-media.collegeboard.org/sat/pdf/sat-subject-tests-percentile-ranks.pdf* (2019).

[9] Colleage Board. "SAT Understanding Scores". In: *https://collegereadiness.collegeboard.org/pdf/understanding-sat-scores.pdf* (2019).

[10] Anna Bogomolnaia et al. "Competitive division of a mixed manna". In: *Econometrica* 85.6 (2017), pp. 1847–1871.

[11] Anna Bogomolnaia et al. "Dividing bads under additive utilities". In: *Social Choice and Welfare* 52.3 (2019), pp. 395–417.

[12] Sylvain Bouveret and Michel Lemaıtre. “Characterizing conflicts in fair division of indivisible goods using a scale of criteria”. In: *Autonomous Agents and Multi-Agent Systems* 30.2 (2016), pp. 259–290.

[13] Steven J Brams and Alan D Taylor. “An envy-free cake division protocol”. In: *The American Mathematical Monthly* 102.1 (1995), pp. 9–18.

[14] Sebastian Braun, Nadja Dwenger, and Dorothea Kübler. “Telling the truth may not pay off: An empirical study of centralized university admissions in Germany”. In: *The BE Journal of Economic Analysis & Policy* 10.1 (2010).

[15] Eric Budish. “The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes”. In: *Journal of Political Economy* 119.6 (2011), pp. 1061–1103.

[16] Ioannis Caragiannis et al. “The unreasonable fairness of maximum Nash welfare”. In: *ACM Transactions on Economics and Computation (TEAC)* 7.3 (2019), pp. 1–32.

[17] Hector Chade and Lones Smith. “Simultaneous search”. In: *Econometrica* 74.5 (2006), pp. 1293–1307.

[18] Yeon-Koo Che and Youngwoo Koh. “Decentralized college admissions”. In: *Journal of Political Economy* 124.5 (2016), pp. 1295–1338.

[19] *Draft Rules*. URL: http://www.nhl.com/ice/page.htm?id=86689.

[20] David Gale and Lloyd S Shapley. “College admissions and the stability of marriage”. In: *The American Mathematical Monthly* 69.1 (1962), pp. 9–15.

[21] Isa E Hafalir et al. *College admissions with entrance exams: Centralized versus decentralized*. Tech. rep. SFB 649 Discussion Paper, 2016.

[22] Allan Hernandez-Chanto. “Centralized Assignment of Students to Majors: Evidence from the University of Costa Rica Job Market Paper”. In: (2016).

[23] Nicole Immorlica and Mohammad Mahdian. “Marriage, honesty, and stability”. In: (2003).

[24] Kathryn Johnston et al. “To draft or not to draft? A systematic review of North American sports’ entry draft”. In: *Scandinavian journal of medicine & science in sports* (2021).

[25] Flip Klijn and Ayşe Yazıcı. “A many-to-many ‘rural hospital theorem’”. In: *Journal of Mathematical Economics* 54 (2014), pp. 63–73.

[26] Fuhito Kojima and Parag A Pathak. “Incentives and stability in large two-sided matching markets”. In: *American Economic Review* 99.3 (2009), pp. 608–27.

[27] David Kurokawa, John Lai, and Ariel Procaccia. "How to cut a cake before the party ends". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. 1. 2013.

[28] David Kurokawa, Ariel D Procaccia, and Junxing Wang. "Fair enough: Guaranteeing approximate maximin shares". In: *Journal of the ACM (JACM)* 65.2 (2018), pp. 1–27.

[29] SangMok Lee. "Incentive compatibility of large centralized matching markets". In: *The Review of Economic Studies* 84.1 (2016), pp. 444–463.

[30] Antonio Miralles. "School Choice:The Case for the Boston Mechanism". In: (2008). Boston University Job Market Paper.

[31] B Moldovanu. "Roth, AE, and Sotomayor, MAO: Two-sided Matching-A Study in Game-theoretic Modeling and Analysis (Book Review)". In: *Journal of Economics/Zeitschrift für Nazionalökonomie* 55 (1992), p. 116.

[32] *NBA Draft Lottery: Schedule, odds and how it works*. URL: https://www.nba.com/nba-draft-lottery-explainer.

[33] Joana Pais and Ágnes Pintér. "School choice and information: An experimental study on matching mechanisms". In: *Games and Economic Behavior* 64.1 (2008), pp. 303–328.

[34] Joana Pais, Ágnes Pintér, and Róbert F Veszteg. "College admissions and the role of information: An experimental study". In: *International Economic Review* 52.3 (2011), pp. 713–737.

[35] Benjamin Plaut and Tim Roughgarden. "Almost envy-freeness with general valuations". In: *SIAM Journal on Discrete Mathematics* 34.2 (2020), pp. 1039–1068.

[36] Antonio Romero-Medina. "Implementation of stable solutions in a restricted matching market". In: *Review of Economic Design* 3.2 (1998), pp. 137–147.

[37] Alvin E Roth. "A natural experiment in the organization of entry-level labor markets: Regional markets for new physicians and surgeons in the United Kingdom". In: *The American economic review* (1991), pp. 415–440.

[38] Alvin E Roth. "The college admissions problem is not equivalent to the marriage problem". In: *Journal of economic Theory* 36.2 (1985), pp. 277–288.

[39] Alvin E Roth. "The economics of matching: Stability and incentives". In: *Mathematics of operations research* 7.4 (1982), pp. 617–628.

[40] Alvin E Roth and Elliott Peranson. "The redesign of the matching market for American physicians: Some engineering aspects of economic design". In: *American economic review* 89.4 (1999), pp. 748–780.

[41] Hugo Steihaus. "The problem of fair division". In: *Econometrica* 16 (1948), pp. 101–104.

[42] *The rules of the NFL Draft*. URL: https://operations.nfl.com/journey-to-the-nfl/the-nfl-draft/the-rules-of-the-draft/. (Accessed: 23.11.2021).

[43] UCAS. "UCAS Undergraduate end of cycle data resources 2016 – 2018". In: *https://www.ucas.com/data-and-analysis/ucas-undergraduate-releases/ucas-undergraduate-end-cycle-data-resources* (2018).

[44] Júlia Varga. "The role of labour market expectations and admission probabilities in students' application decisions on higher education: The case of Hungary". In: *Education Economics* 14.3 (2006), pp. 309–327.

[45] Min Zhu. "College admissions in China: A mechanism design perspective". In: *China Economic Review* 30 (2014), pp. 618–631.

Appendix A

Appendix to Chapter 1

Python code used for simulations of college admissions

```
from scipy.stats import truncnorm
import random
import numpy as np
from itertools import groupby
import matplotlib.pyplot as plt
from collections import OrderedDict, Counter
from copy import deepcopy
from datetime import datetime
from statistics import median
import pickle

**************************************************************
*                   Generate and update colleges             *
**************************************************************

def generateColleges(nColleges, cutoffs, quotas):
    colleges = []
    for i in range(nColleges):
        newCollege = {
            "name": i,
            "lastYearCutoff": cutoffs[i],
            "quota": quotas[i]
        }
        colleges.append(newCollege)
    return colleges

def updateColleges(colleges, cutoffs):
# changes the elements in colleges
```

```
    for i, college in enumerate(colleges):
        college["lastYearCutoff"] = cutoffs[i]
    return


**************************************************
*              Generate and update Students                *
**************************************************
def getTruncatedNormal(mean=60, sd=30, low=0, upp=10):
    return truncnorm((low - mean) / sd, (upp - mean) / sd, loc=mean,
                                        scale=sd).rvs(size=1)[0]

def getUniform(low=0, high=100):
    return np.random.uniform(low, high)

def getPreferences(PreferenceProbabilities):
    nOfColleges = len(PreferenceProbabilities)
    preferences = list(np.random.choice(nOfColleges, nOfColleges,
                                        replace=False, p=
                                        PreferenceProbabilities))
    return preferences

def getUtilities(preferences):
    utilities = []
    nOfColleges = len(preferences)
    valUtilities=[]
    utilities = []
    nOfColleges = len(preferences)
    for i in range(0,nOfColleges):
        newUtility = 1 - i/nOfColleges
        valUtilities.append(newUtility)
    utilities = [x[1] for x in sorted(zip(preferences,valUtilities),
                                        key=lambda x: x[0])] #
                                                , reverse=
                                        True)]
    utilities
    return utilities

def generateStudents(nStudents, StudentType,
                                    StudentPreferenceProbabilities):
    from random import randint
    students = []
    for i in range(nStudents):
        newStudent = {
            "examScore": getTruncatedNormal(70, 20, 0, 100) , #
```

```
                                                getUniform() #,
            "type": StudentType,
            "preferences": [],
            "probabilities": [],
            "utilities": [],
            "expectedUtilities": [],
            "applications":[],
            "applications_to_keep":[],
            "continueAllocating": True,
            "allocatedTo": None,
            "perceivedCutoffs": []
        }
        newStudent["preferences"] = getPreferences(
                                            StudentPreferenceProbabilities
                                            )
        newStudent["applications"] = deepcopy(newStudent["preferences
                                            "]) # by default students
                                            can apply everywhere
        newStudent["applications_to_keep"] = deepcopy(newStudent["
                                            preferences"])
        newStudent["utilities"] = getUtilities(newStudent["
                                            preferences"])
        students.append(newStudent)
    students = sorted(students, key=lambda x: x["examScore"], reverse
                                        =True) #otherwise sorting will
                                         be needed during allocation
    return students
def getProbabilities(examScore, studentType, lastYearCutoff):
    probabilities = [(examScore-element+studentType)/(2*studentType)
                                        for element in lastYearCutoff]
    probabilities = [min(1,element) for element in probabilities]
    probabilities = [max(0,element) for element in probabilities]
    return probabilities

def updateStudentsPerceivedCutoffs(students, perceived_cutoffs):
# changes the elements in students
    for student in students:
        student['perceivedCutoffs'] = perceived_cutoffs


def updateStudentsProbAndEu(students):
# changes the elements in students
    for student in students:
        student["probabilities"] = getProbabilities(student["
                                            examScore"],student["type"
```

```
                                        ],student["
                                        perceivedCutoffs"])
        student["expectedUtilities"] = [a * b for a, b in zip(student
                                        ["utilities"], student["
                                        probabilities"])]
    return students

def optimalApplication(student,nApplications):
    tuple_in_order = sorted(zip(student['expectedUtilities'], list(i
                                    for i in range(len(student['
                                    preferences'])))), key =
                                    lambda x: -x[0])
    maxExpectedUtility = tuple_in_order[0][0]
    best_colleges = [x[1] for x in tuple_in_order if x[0]==
                                    maxExpectedUtility]
    if len(best_colleges) == 1:
        return best_colleges
    else:
        preference_order = [student["preferences"].index(x) for x in
                                        best_colleges]
        best_colleges_sorted_by_preference = [x[1] for x in sorted(
                                        zip(preference_order,
                                        best_colleges))]
        return best_colleges_sorted_by_preference[:nApplications]

def updateStudentApplications(students, nApplications):
# changes the elements in students
    for student in students:
        student["applications"] = optimalApplication(student,
                                        nApplications)
        student["applications_to_keep"] = deepcopy(student["
                                        applications"])
        student['continueAllocating'] = True
    return students

**************************************************
*                  Allocation procedure                  *
**************************************************
def getAllocation(students, colleges):
    remainingQuota = [item['quota'] for item in colleges]
    for student in students:
        while student['continueAllocating'] == True:
            college = student["applications"].pop(0)
            if student["applications"]==[]:
                student["continueAllocating"] = False
```

```
            if remainingQuota[college]>0:
                if student["allocatedTo"] is None:
                    remainingQuota[college] -= 1
                    student["allocatedTo"] = college
                elif student["preferences"].index(college)<student["
                                                    preferences"].
                                                    index(student["
                                                    allocatedTo"]):
                    remainingQuota[student["allocatedTo"]]+=1
                    remainingQuota[college] -= 1
                    student["allocatedTo"] = college

**************************************************
*                    Data generation                *
**************************************************
def findCutoffs(colleges, students):
    getAllocation(students,colleges)
    cutoffs = {k: min([x['examScore'] for x in v]) for k, v in
                                        groupby(students, lambda x: x[
                                        "allocatedTo"])}
    del cutoffs[None]
    cutoff_output = OrderedDict(sorted(cutoffs.items(), key=lambda t:
                                        t[0]))
    return [elem[1] for elem in cutoff_output.items()]


def distortCutoffs(cutoffs,distort):
    for key in cutoffs:
        cutoffs[key] += distort
    getAllocation(students,colleges)
    cutoffs = {k: min([x['examScore'] for x in v]) for k, v in
                                        groupby(students, lambda x: x[
                                        "allocatedTo"])}
    cutoffs_to_return = []
    for k in sorted(cutoffs.keys()):
        cutoffs_to_return.append(cutoffs[k][:-1])
    return cutoffs_to_return

def GenerateData(nColleges, quotas, nStudents, StudentType,
                                    StudentPreferenceProbabilities,
                                    distort):
    colleges = generateColleges(nColleges, [0]*nColleges, quotas)
    baseStudents = generateStudents(nStudents, StudentType,
                                        StudentPreferenceProbabilities
                                        )
```

```
    limitedStudents = deepcopy(baseStudents)
    lastYearCutoffs = findCutoffs(colleges, baseStudents)
    updateStudentsPerceivedCutoffs(limitedStudents, lastYearCutoffs)
    updateColleges(colleges, lastYearCutoffs)
    updateStudentsProbAndEu(limitedStudents)
    return colleges, baseStudents, limitedStudents, lastYearCutoffs

**************************************************
*                        Visuals                 *
**************************************************

def findFrontiers(students):
    frontiers = {}
    preferenceProfile = set([tuple(x['preferences']) for x in
                                        students])
    for pref in preferenceProfile:
        studentsWithPreferences = [stud for stud in students if tuple
                                            (stud['preferences']) ==
                                            pref]
#
#         printStudents(studentsWithPreferences)
        currentAllocation = studentsWithPreferences[0]['allocatedTo']
        maxFrontier = 100
        currentPref = tuple()
        currentCutoff = studentsWithPreferences[0]['examScore']
        for st in studentsWithPreferences[1:]:
            if st['allocatedTo'] != currentAllocation:
                currentPref = tuple(st['preferences'])
                frontiers.setdefault(currentPref,[]).append([
                                                    currentCutoff,
                                                    maxFrontier,
                                                    currentAllocation]
                                                    )
                maxFrontier = currentCutoff
                currentAllocation = st['allocatedTo']
            currentCutoff = st['examScore']
        currentPref = tuple(st['preferences'])
        frontiers.setdefault(currentPref,[]).append([0, maxFrontier,
                                            currentAllocation])
    return frontiers

def plotAllocation(students, nColleges):
    coloursTwoColleges={0: 'green', 1: 'blue', None: 'red'}
    coloursThreeColleges={0: 'green', 1: 'blue', 2: 'cyan', None:'red
                                        '}
```

```
    if nColleges == 2:
        colours = coloursTwoColleges
    elif nColleges == 3:
        colours = coloursThreeColleges

    n_points = 1000
    eScore = np.arange(0, 100, 0.1)
    frontiers = findFrontiers(students)
    print(frontiers)
    y_step = 1/len(frontiers)
    current_ymax = 1
    current_ymin = 0
    fig, (ax) = plt.subplots(1, 1, sharex=True)
    y1 = [1]*n_points
    ax.plot(eScore, y1, color='black')
    for i, key in enumerate(frontiers):
        y2 = [1 - (i+1)*y_step]*n_points #1-...to plot from the top
        ax.plot(eScore, y2, color='black')
        current_frontiers = frontiers[key]
        for fr in current_frontiers:
            ax.fill_between(np.linspace(fr[0], fr[1], n_points), y1,
                                                y2, facecolor=colours[
                                                fr[2]], interpolate=
                                                True)
        y1 = y2

def plotEu(students):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    students_who_prefer_0 = [elem for elem in students if elem['
                                        preferences']==[0,1]]
    students_who_prefer_1 = [elem for elem in students if elem['
                                        preferences']==[1,0]]
    Eu_0_prefer_0 = [st["expectedUtilities"][0] for st in
                                        students_who_prefer_0]
    Eu_1_prefer_0 = [st["expectedUtilities"][1] for st in
                                        students_who_prefer_0]
    ax.plot([st["examScore"] for st in students_who_prefer_0],
                                        Eu_0_prefer_0, color='tab:
                                        green')
    ax.plot([st["examScore"] for st in students_who_prefer_0],
                                        Eu_1_prefer_0, color='tab:blue
                                        ')
    for st in students_who_prefer_0:
        if st["expectedUtilities"][0] <  st["expectedUtilities"][1]:
```

```
            print(st["examScore"])
            break
    #ax = fig.add_subplot(2, 1, 2)
    fig2 = plt.figure()
    ax2 = fig2.add_subplot(1, 1, 1)
    ax2.plot([st["examScore"] for st in students_who_prefer_1], [st["
                                    expectedUtilities"][0] for st
                                    in students_who_prefer_1],
                                    color='tab:green')
    ax2.plot([st["examScore"] for st in students_who_prefer_1], [st["
                                    expectedUtilities"][1] for st
                                    in students_who_prefer_1],
                                    color='tab:blue')

def plotChanges(changes, varx, vary, xlabel, ylabel, legend):
    colours = {0: 'blue', 1:'green', 2:'red', 3:'orange', 4:'cyan'}
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    for i, ch in enumerate(changes):
        ax.plot([x[varx] for x in ch], [y[vary] for y in ch], color='
                                        tab:'+colours[i])
    ax.legend(legend)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)



**************************************************
*          Single iteration with allocation graphs          *
**************************************************

def getTotalUtility(students):
    totalUtility = sum([x['utilities'][x['allocatedTo']] for x in
                                    students if not (x["
                                    allocatedTo"] is None)])
    return totalUtility

numColleges = 2
numStudents = 1000
Quotas = [200,600]
stType = 30
prefProbabilities = [0.5,0.5]
cutoffsDistortions = 0
maxApplicationsLimited = 1
```

```
Colleges, BaseStudents, LStudents, cutoffs = GenerateData(numColleges
                                    , Quotas, numStudents,  stType,
                                    prefProbabilities,
                                    cutoffsDistortions)
LimitedStudents = updateStudentApplications(deepcopy(LStudents),
                                    maxApplicationsLimited)
getAllocation(LimitedStudents, Colleges)


print(getTotalUtility(BaseStudents))
plotAllocation(BaseStudents,numColleges)
print(getTotalUtility(LimitedStudents))
plotAllocation(LimitedStudents,numColleges)

**************************************************
*               Computing results, medians                *
**************************************************
AllMedianChanges = []
numColleges = 2 #4
numStudents = 1000
#Quotas = [200,300]
AllQuotas = [[100,100],[350,350],[300,400]] #[[200,200,200,200], [100
                                    ,100,200,300]]
#stType = 5
stMaxType = 20
stTypeStep = 1
prefProbabilities = [0.5,0.5]
cutoffsDistortions = 0
maxApplicationsLimited = 1
numRepetitions = 50


for Quotas in AllQuotas:
    start_time = datetime.now()
    changes = []
    median_changes = []
    baseline = []
    for t in list(range(1, stMaxType+1,stTypeStep)):
        print(t, "/", stMaxType)
        iter_changes = []
        for i in range(numRepetitions):
            Colleges, BaseStudents, LStudents, cutoffs = GenerateData
                                    (numColleges, Quotas,
                                    numStudents, t,
                                    prefProbabilities,
```

```
                                        cutoffsDistortions)
        currBaselineTotalUtility = getTotalUtility(BaseStudents)
        currBaseline = {
            'type': 'baseline',
            'totalUtility': currBaselineTotalUtility
        }
        baseline.append(currBaseline)
        LimitedStudents = updateStudentApplications(deepcopy(
                                        LStudents),
                                        maxApplicationsLimited
                                        )
        getAllocation(LimitedStudents, Colleges)

        numReallocations = 0
        for i, st in enumerate(BaseStudents):
            st2 = LimitedStudents[i]
            if st['allocatedTo']!=st2['allocatedTo']:
                numReallocations += 1

        currTotalUtility = getTotalUtility(LimitedStudents)
        currTotalUtilityChange = currTotalUtility -
                                        currBaselineTotalUtility

        curr_change = {
            'type': t,
            'totalUtility': currTotalUtility,
            'totalUtilityChange': currTotalUtilityChange,
            'TotalPerCentUtilityChange': currTotalUtilityChange*
                                            100/
                                            currBaselineTotalUtility
                                            ,
            'frontiers': findFrontiers(LimitedStudents),
            'reallocations': numReallocations
        }
        iter_changes.append(curr_change)
        changes.append(curr_change)
    av_change = {
        'type': t,
        'totalUtility': median([elem['totalUtility'] for elem in
                                        iter_changes]),
        'totalUtilityChange': median([elem['totalUtilityChange']
                                        for elem in
                                        iter_changes]),
        'TotalPerCentUtilityChange': median([elem['
                                        TotalPerCentUtilityChange
```

```
                                                    '] for elem in
                                                    iter_changes]),
            'reallocations': median([elem['reallocations'] for elem
                                                    in iter_changes])
        }
        median_changes.append(av_change)
    AllMedianChanges.append({'Quotas':Quotas, 'MedianChanges':
                                        median_changes})
    print(datetime.now() - start_time)


with open('.pickle', 'wb') as f:
    pickle.dump(AllMedianChanges, f)


**************************************************************
*                          Plots                             *
**************************************************************

QuotasToPrint = [[100,100],[200,200],[350,350],[300,400],[400,500]]
MedianChangesToPrint = [elem for elem in AllMedianChanges if elem['
                                    Quotas'] in QuotasToPrint]
plotChanges([elem['MedianChanges'] for elem in MedianChangesToPrint],
                                     'type', 'reallocations',
            'Student type', 'Number of reallocations', [elem ['Quotas
                                                    '] for elem in
                                                    MedianChangesToPrint])
plotChanges([elem['MedianChanges'] for elem in MedianChangesToPrint],
                                     'type', '
                                    TotalPerCentUtilityChange',
            'Student type', 'Change in Total Utility, %',[elem ['
                                                    Quotas'] for elem in
                                                    MedianChangesToPrint])
```

Appendix B

Appendix to Chapter 3

Python code used for simulations of ModDA1

```
from scipy.stats import truncnorm
from pprint import pprint
import random
from random import shuffle
import numpy as np
from itertools import groupby, permutations #product
import matplotlib.pyplot as plt
from collections import OrderedDict, Counter
from copy import deepcopy
from datetime import datetime
from statistics import median
import pickle
**************************************************
*                        Data generation                    *
**************************************************
def GenerateStudents(nCourses, nStudents,
                                    limitNumberOfPreferenceProfilesStudents
                                    =None):
    Students=[]
    CoursesIndices = [i for i in range(nCourses)]
    PreferenceProfiles = list(permutations(CoursesIndices))
    if limitNumberOfPreferenceProfilesStudents is not None:
        PreferenceProfiles = PreferenceProfiles[:
                                            limitNumberOfPreferenceProfilesStudents
                                            ]
    preferences = random.choices(PreferenceProfiles, k=nStudents)
    for i in range(nStudents):
        Student={
```

```
            'Preferences': list(preferences[i]),
            'ReportedPreferences':list(preferences[i]),
            'WaitingList': [],
            'RejectionList': [],
            'ForcedCourses':[],
            'boolForcedCourses': False,
            'StillToApply': list(preferences[i])
            #'k'
        }
        Students+=[Student]
    return Students

def GenerateCourses(nCourses, nStudents):
    Courses=[]
    StudentsIndices = [i for i in range(nStudents)]
    PreferenceProfiles = list(permutations(StudentsIndices))
    preferences = random.choices(PreferenceProfiles, k=nCourses)
    for i in range(nCourses):
        Course={
            'Preferences': preferences[i],
            'WaitingList': [],
            'Applications': []
            #'Capacity': q #q[i]
        }
        Courses+=[Course]
    return Courses

def GenerateData(nStudents, nCourses,
                                    limitNumberOfPreferenceProfilesStudents
                                    =None):
    return GenerateStudents(nCourses, nStudents,
                                        limitNumberOfPreferenceProfilesStudents
                                        ), GenerateCourses(nCourses,
                                        nStudents)

**************************************************************
*                          DA and ModDA1                     *
**************************************************************

def AllocationDA(Students, Courses, k, q):
    '''
        k - number of courses each student needs to be allocated to,
        q - quotas at all courses
    '''
    flagApplicationMade=True
```

```
    while flagApplicationMade:
        flagApplicationMade = False
        for i,st in enumerate(Students):
            for _ in range(0,k-len(st['WaitingList'])):
                if not st['StillToApply']:
                      break
                Courses[st['StillToApply'][0]]['Applications'] += [i]
                flagApplicationMade=True
                st['StillToApply'] = st['StillToApply'][1:]
            st['WaitingList'] = []
        for j, course in enumerate(Courses):
            ApplicationsInPrefOrder = [ap for ap in course['
                                                Preferences'] if ap in
                                                 course['Applications'
                                                ]]
            WaitingList = ApplicationsInPrefOrder[:q]
            course['WaitingList'] = deepcopy(WaitingList)
            course['Applications'] = deepcopy(WaitingList)
            RejectionList = ApplicationsInPrefOrder[q:]
            for i_st in WaitingList:
                Students[i_st]['WaitingList']+=[j]
            for i_st in RejectionList:
                Students[i_st]['RejectionList']+=[j]
    return deepcopy(Students), deepcopy(Courses)

def ListUnderDemandedCourses(Courses,q):
    UnderdemandedCourses = []
    for i,c in enumerate(Courses):
        if len(c["WaitingList"])<q:
            UnderdemandedCourses += [i]
    return UnderdemandedCourses

def ListUnderDemandedStudents(Students,k):
    UnderdemandedStudents = []
    for i,s in enumerate(Students):
        if len(s["WaitingList"])<k:
            UnderdemandedStudents += [i]
    return UnderdemandedStudents

def AllocationModDA(DataStudents,DataCourses, Priorities, k, q):
    '''
        Priorities is a list of students ordered from low to high
                                            priority
        k - number of courses each student needs to be allocated to,
        q - quotas at all courses
```

```
'''
Students, Courses = AllocationDA(deepcopy(DataStudents), deepcopy
                                 (DataCourses), k, q)
#check if all the students are already allocated to exactly k
                                 courses
UnderdemandedStudents = ListUnderDemandedStudents(Students,k)
if not UnderdemandedStudents:
    return Students, Courses
#if not, remove the student with lowest priority from his least
                                 preferred course (conditions
                                 apply)
while UnderdemandedStudents:
    for st in Students:
        st['boolForcedCourses']=True
    UnderdemandedCourses = ListUnderDemandedCourses(Courses,q)
    for i in range(len(Priorities)):
        iStudent = Priorities[i] #Student's index in Students
        if iStudent not in UnderdemandedStudents:
            WaitingOverdemandedCourses = [c for c in Students[
                                             iStudent]['
                                             WaitingList'] if c
                                              not in
                                             UnderdemandedCourses
                                             ]
            iCourseRemove = [c for c in Students[iStudent]['
                                             ReportedPreferences
                                             '] if c in
                                             WaitingOverdemandedCourses
                                             ][-1]
            Courses[iCourseRemove]['WaitingList'].remove(iStudent
                                             )
            Students[iStudent]['WaitingList'].remove(
                                             iCourseRemove)
            Students[iStudent]['ReportedPreferences'].remove(
                                             iCourseRemove)
            iForcedCourse = [c for c in Students[iStudent]['
                                             ReportedPreferences
                                             '] if c in
                                             UnderdemandedCourses
                                             ][0]
            Courses[iForcedCourse]['WaitingList']+=[iStudent]
            Students[iStudent]['WaitingList']+= [iForcedCourse]
            Students[iStudent]['ForcedCourses']+= [iForcedCourse]
            break
    for st in Students:
```

```
            st['StillToApply'] = deepcopy(st['ReportedPreferences'])
        for c in Courses:
            c['Applications'] = []
        Students,Courses = AllocationDA(deepcopy(Students), deepcopy(
                                        Courses), k, q)
        UnderdemandedStudents = ListUnderDemandedStudents(Students,k)
    return deepcopy(Students), deepcopy(Courses)

**************************************************************
*                        Manipulations                       *
**************************************************************

def ManipulateDA(Students, Courses, k, q, iManipulator=0):
    '''iManipulator is an index of a student who manipulates'''
    Manipulations = []
    DAStudents, DACourses = AllocationDA(deepcopy(Students), deepcopy
                                         (Courses), k, q)
    DAAllocation = deepcopy(DAStudents[iManipulator]['WaitingList'])

    CoursesIndices = [i for i in range(len(Courses))]
    PreferenceProfiles = list(permutations(CoursesIndices))
    PreferenceProfiles.remove(tuple(Students[0]['Preferences']))

    manipStudents = deepcopy(Students)
    for pref in PreferenceProfiles:
        manipStudents[iManipulator]['Preferences'] = deepcopy(list(
                                        pref))
        manipStudents[iManipulator]['StillToApply'] = deepcopy(list(
                                        pref))
        DAManipStudnets, DAManipCourses = AllocationDA(deepcopy(
                                        manipStudents), deepcopy(
                                        Courses), k, q)
        Manipulation = {
            'Allocation': 'DA',
            'Manipulator': iManipulator,
            'Reported preferences': deepcopy(pref),
            'Students': deepcopy(DAStudents),
            'Courses': deepcopy(DACourses),
            'ManipStudnets': deepcopy(DAManipStudnets),
            'ManipCourses': deepcopy(DAManipCourses)
        }
        Manipulations+=[Manipulation]
    return Manipulations

def ManipulateModDA(Students, Courses, Priorities, k, q, iManipulator
```

```
                                        =0):
    #without manipulations
    Manipulations = []
    ModDAStudents, ModDACourses = AllocationModDA(deepcopy(Students),
                                        deepcopy(Courses), Priorities,
                                         k, q)

    CoursesIndices = [i for i in range(len(Courses))]
    PreferenceProfiles = list(permutations(CoursesIndices))
    PreferenceProfiles.remove(tuple(Students[iManipulator]['
                                        Preferences']))

    manipStudents = deepcopy(Students)
    for pref in PreferenceProfiles:
        manipStudents[iManipulator]['Preferences'] = deepcopy(list(
                                        pref))
        manipStudents[iManipulator]['StillToApply'] = deepcopy(list(
                                        pref))
        ModDAManipStudnets, ModDAManipCourses = AllocationModDA(
                                        deepcopy(manipStudents),
                                        deepcopy(Courses),
                                        Priorities, k, q)
        noForcedMAllocation = deepcopy(ModDAManipStudnets[0]['
                                        WaitingList'])
        ForcedMAllocation = deepcopy(ModDAManipStudnets[0]['
                                        WaitingList'])
        Manipulation = {
            'Allocation': 'ModDA',
            'Manipulator': iManipulator,
            'Reported preferences': deepcopy(pref),
            'Students': deepcopy(ModDAStudents),
            'Courses': deepcopy(ModDACourses),
            'ManipStudnets': deepcopy(ModDAManipStudnets),
            'ManipCourses': deepcopy(ModDAManipCourses)
        }
        Manipulations+=[Manipulation]
    return Manipulations

**************************************************************
*                    Summarising results                     *
**************************************************************

def isImprovement(InitialAllocation, FinalAllocation, Preferences):
    IndecesOutcome1 = sorted([Preferences.index(i) for i in
                                        InitialAllocation])
```

```
    IndecesOutcome2 = sorted([Preferences.index(i) for i in
                                        FinalAllocation])
    if IndecesOutcome1==IndecesOutcome2:
        return 'no change'
    #if different length, assign the worst outside options to the
                                        shoter list
    if len(IndecesOutcome1)<len(IndecesOutcome2):
        IndecesOutcome1 +=[len(Preferences)]*(len(IndecesOutcome2)-
                                            len(IndecesOutcome1))
    if len(IndecesOutcome2)<len(IndecesOutcome1):
        IndecesOutcome2 +=[len(Preferences)]*(len(IndecesOutcome1)-
                                            len(IndecesOutcome2))

    IndecesTupled = list(zip(IndecesOutcome1,IndecesOutcome2))
    WeakComparison  = [x[0]>=x[1] for x in IndecesTupled]#if x[0]>x[1
                                        ], then the new allocation is
                                        better
    nWeaklyDominantPairs  = sum(WeakComparison)
    if nWeaklyDominantPairs ==len(WeakComparison):
        return 'Yes'
    else: return 'No'

def ManupulationResultSummary(Manipilations):
    Summary=[]
    for Manip in Manipilations:
        InitialAllocation = []
        ManipulatedAllocation = []
        iManipulator = Manip['Manipulator']
        ReportedPreferences = Manip['Reported preferences']
        for st in Manip['Students']:
            InitialAllocation += [[i for i in st['WaitingList']]]
        for st in Manip['ManipStudnets']:
            ManipulatedAllocation += [[i for i in st['WaitingList']]]
        #find manipulator's outcomes
        manInitAlloc = InitialAllocation[iManipulator]
        manFinAlloc = ManipulatedAllocation[iManipulator]
        manPreferences = Manip['Students'][iManipulator]['Preferences
                                            ']
        AllocationImproved = isImprovement(manInitAlloc, manFinAlloc,
                                            manPreferences)
        Sum= {
            'Allocation': Manip['Allocation'],
            'Manipulator': iManipulator,
            'ReportedPreferences': ReportedPreferences,
            'InitialAllocation': InitialAllocation,
```

```
            'ManipulatedAllocation': ManipulatedAllocation,
            'Allocation of manipulator improved': AllocationImproved
        }
        Summary+=[Sum]

    return Summary

def ManipulationsComparison(ManipulationsDA, ManipulationsModDA):
    ManDA = ManupulationResultSummary(ManipulationsDA)
    ManModDA = ManupulationResultSummary(ManipulationsModDA)
    comparison = {
        'DA not ModDA': 0,
        'not DA, modDA': 0,
        'neither': 0,
        'both': 0,
        'total':0,
        'Same outcome in both': 0,}
    for i in range(len(ManDA)):
        comparison['total']+=1
        mDA = ManDA[i]
        mModDA = ManModDA[i]
        DAimprovement = mDA['Allocation of manipulator improved']
        ModDAimprovement = mModDA['Allocation of manipulator improved
                                  ']
        if DAimprovement == 'Yes' and ModDAimprovement == 'Yes':
            comparison['both']+=1
            if mDA['ManipulatedAllocation'][mDA['Manipulator']] ==
                                            mModDA['
                                            ManipulatedAllocation'
                                            ][mModDA['Manipulator'
                                            ]]:
                comparison['Same outcome in both'] += 1
        elif DAimprovement == 'Yes':
            comparison['DA not ModDA']+=1
        elif ModDAimprovement == 'Yes':
            comparison['not DA, modDA']+=1
        else: comparison['neither']+=1
    return comparison

**************************************************
*                   Summarising results                    *
**************************************************

nStudents = 4
nCourses = 4
```

```
kCourses = 2 #exact number of courses to be allocated
Quotas = 2
iterations = 1000

limitNumberOfPreferenceProfilesStudents = 3

#DataStudents, DataCourses = GenerateData(nStudents, nCourses,
                                    limitNumberOfPreferenceProfilesStudents
                                    )
start_time_all = datetime.now()
TheComparisons = []
for iter in range(iterations):
    Students, Courses = GenerateData(nStudents, nCourses)
    Comparisons=[]
    for iManipulator in range(nStudents):
        ManipulationsDA = ManipulateDA(deepcopy(Students), deepcopy(
                                        Courses), kCourses, Quotas
                                        , iManipulator)
        Priorities = [i for i in range(nStudents)]
        random.shuffle(Priorities)
        ManipulationsModDA = ManipulateModDA(deepcopy(Students),
                                        deepcopy(Courses),
                                        Priorities, kCourses,
                                        Quotas, iManipulator)
        Comparison = ManipulationsComparison(ManipulationsDA,
                                        ManipulationsModDA)
        Comparisons += [Comparison]

    TheComparison={}
    for key in Comparisons[0].keys():
        TheComparison[key]=(sum([x[key] for x in Comparisons]))
    TheComparisons +=[TheComparison]

SummarisedComparisons={}
for key in TheComparisons[0].keys():
    SummarisedComparisons[key]=[x[key] for x in TheComparisons]

pprint(TheComparison)
pprint(SummarisedComparisons)
```

Python code used for simulations of ModDA2

```
from scipy.stats import truncnorm
from pprint import pprint
```

```
import random
from random import shuffle
import numpy as np
from itertools import groupby, permutations #product
import matplotlib.pyplot as plt
from collections import OrderedDict, Counter
from copy import deepcopy
from datetime import datetime
from statistics import median
import pickle
**************************************************************
*                        Data generation                     *
**************************************************************
def GenerateStudents(nCourses, nStudents, k,
                                    limitNumberOfPreferenceProfilesStudents
                                    =None):
    Students=[]
    CoursesIndices = [i for i in range(nCourses)]
    PreferenceProfiles = list(permutations(CoursesIndices))
    if limitNumberOfPreferenceProfilesStudents is not None:
        PreferenceProfiles = PreferenceProfiles[:
                                            limitNumberOfPreferenceProfilesStudents
                                            ]
    preferences = random.choices(PreferenceProfiles, k=nStudents)
    for i in range(nStudents):
        Student={
            'Preferences': list(preferences[i]),
            'ReportedPreferences':list(preferences[i]),
            'WaitingList': [],
            'RejectionList': [],
            'ForcedCourses':[],
            'StillToApply': list(preferences[i]),
            'k': k[i],
            'current_k': k[i]
        }
        Students+=[Student]
    return Students

def GenerateCourses(nCourses, nStudents, q):
    Courses=[]
    StudentsIndices = [i for i in range(nStudents)]
    PreferenceProfiles = list(permutations(StudentsIndices))
    preferences = random.choices(PreferenceProfiles, k=nCourses)
    for i in range(nCourses):
        Course={
```

```
            'Preferences': preferences[i],
            'WaitingList': [],
            'Applications': [],
            'ForcedStudents':[],
            'q': q[i],
            'current_q': q[i]
        }
        Courses+=[Course]
    return Courses

def GenerateData(nStudents, nCourses, k, q,
                                    limitNumPrProfilesStudents=None):
    '''
        k - list of number of courses each student needs to be
                                            allocated to,
        q - list of quotas at all courses
    '''
    return GenerateStudents(nCourses, nStudents, k,
                                        limitNumPrProfilesStudents),
                                        GenerateCourses(nCourses,
                                        nStudents, q)

**************************************************************
*                          DA and ModDA2                     *
**************************************************************

def AllocationDA(Students, Courses):
    flagApplicationMade=True
    while flagApplicationMade:
        flagApplicationMade = False
        for i,st in enumerate(Students):
            for _ in range(0,st['current_k']-len(st['WaitingList'])):
                if not st['StillToApply']:
                      break
                Courses[st['StillToApply'][0]]['Applications'] += [i]
                flagApplicationMade=True
                st['StillToApply'] = st['StillToApply'][1:]
            st['WaitingList'] = []
        for j, course in enumerate(Courses):
            ApplicationsInPrefOrder =
                [ap for ap in course['Preferences']
                    if (ap in course['Applications'])
                        and (ap not in course['ForcedStudents'])]
            WaitingList = ApplicationsInPrefOrder[:course['current_q'
                                                ]]
```

```
            course['WaitingList'] = deepcopy(WaitingList)
            course['Applications'] = deepcopy(WaitingList)
            RejectionList = ApplicationsInPrefOrder[course['current_q
                                                    ']:]
            for i_st in WaitingList:
                Students[i_st]['WaitingList']+=[j]
            for i_st in RejectionList:
                Students[i_st]['RejectionList']+=[j]
    return deepcopy(Students), deepcopy(Courses)

def ListUnderDemandedCourses(Courses):
    UnderdemandedCourses = []
    for i,c in enumerate(Courses):
        if len(c['WaitingList'])<c['current_q']:
            UnderdemandedCourses += [i]
    return UnderdemandedCourses

def ListUnderDemandedStudents(Students):
    UnderdemandedStudents = []
    for i,s in enumerate(Students):
        if len(s['WaitingList'])<s['current_k']:
            UnderdemandedStudents += [i]
    return UnderdemandedStudents

def AllocationModDA(DataStudents,DataCourses, Priorities):
    '''
        Priorities is a list of students ordered from low to high
                                            priority
    '''
    Students, Courses = AllocationDA(deepcopy(DataStudents), deepcopy
                                        (DataCourses))
    #check if all the students are already allocated to exactly k
                                        courses
    UnderdemandedStudents = ListUnderDemandedStudents(Students)
    if not UnderdemandedStudents:
        return Students, Courses
    #if not, remove the student with lowest priority from his least
                                        preferred course (conditions
                                        apply)
    while UnderdemandedStudents:
        UnderdemandedCourses = ListUnderDemandedCourses(Courses)
        for i in range(len(Priorities)):
            iStudent = Priorities[i] #Student's index in Students
            if iStudent not in UnderdemandedStudents:
                cur_st = Students[iStudent]
```

```
                iForcedCourse = [c for c in cur_st['
                                                    ReportedPreferences
                                                    '] if c in
                                                    UnderdemandedCourses
                                                     and c not in
                                                    cur_st['
                                                    ForcedCourses'] ][
                                                    0]
                Courses[iForcedCourse]['ForcedStudents']+=[iStudent]
                Courses[iForcedCourse]['current_q']-=1
                cur_st['ForcedCourses']+=[iForcedCourse]
                cur_st['current_k']-=1
                break
        for st in Students:
            StillToApply = deepcopy(st['ReportedPreferences'])
            for ForcedCourse in st['ForcedCourses']:
                StillToApply.remove(ForcedCourse)
            st['StillToApply'] = StillToApply
        for c in Courses:
            c['Applications'] = []
        Students,Courses = AllocationDA(deepcopy(Students), deepcopy(
                                            Courses))
        UnderdemandedStudents = ListUnderDemandedStudents(Students)
    return deepcopy(Students), deepcopy(Courses)

**************************************************************
*                        Manipulations                       *
**************************************************************

def ManipulateDA(Students, Courses, iManipulator=0):
    '''iManipulator is an index of a student who manipulates'''
    Manipulations = []
    DAStudents, DACourses = AllocationDA(deepcopy(Students), deepcopy
                                        (Courses))
    DAAllocation = deepcopy(DAStudents[iManipulator]['WaitingList'])

    CoursesIndices = [i for i in range(len(Courses))]
    PreferenceProfiles = list(permutations(CoursesIndices))
    manipStudents = deepcopy(Students)
    for pref in PreferenceProfiles:
        manipStudents[iManipulator]['ReportedPreferences'] = deepcopy
                                            (list(pref))
        manipStudents[iManipulator]['StillToApply'] = deepcopy(list(
                                            pref))
        DAManipStudnets, DAManipCourses = AllocationDA(deepcopy(
```

```
                                        manipStudents), deepcopy(
                                        Courses))
            Manipulation = {
                'Allocation': 'DA',
                'Manipulator': iManipulator,
                'Reported preferences': deepcopy(pref),
                'Students': deepcopy(DAStudents),
                'Courses': deepcopy(DACourses),
                'ManipStudnets': deepcopy(DAManipStudnets),
                'ManipCourses': deepcopy(DAManipCourses)
            }
            Manipulations+=[Manipulation]
        return Manipulations

    def ManipulateModDA(Students, Courses, Priorities, iManipulator=0):
        #without manipulations
        Manipulations = []
        ModDAStudents, ModDACourses = AllocationModDA(deepcopy(Students),
                                            deepcopy(Courses), Priorities)

        CoursesIndices = [i for i in range(len(Courses))]
        PreferenceProfiles = list(permutations(CoursesIndices))
        manipStudents = deepcopy(Students)
        for pref in PreferenceProfiles:
            manipStudents[iManipulator]['ReportedPreferences'] = deepcopy
                                        (list(pref))
            manipStudents[iManipulator]['StillToApply'] = deepcopy(list(
                                        pref))
            ModDAManipStudnets, ModDAManipCourses = AllocationModDA(
                                        deepcopy(manipStudents),
                                        deepcopy(Courses),
                                        Priorities)
            noForcedMAllocation = deepcopy(ModDAManipStudnets[0]['
                                        WaitingList'])
            ForcedMAllocation = deepcopy(ModDAManipStudnets[0]['
                                        WaitingList'])
            Manipulation = {
                'Allocation': 'ModDA',
                'Manipulator': iManipulator,
                'Reported preferences': deepcopy(pref),
                'Students': deepcopy(ModDAStudents),
                'Courses': deepcopy(ModDACourses),
                'ManipStudnets': deepcopy(ModDAManipStudnets),
                'ManipCourses': deepcopy(ModDAManipCourses)
            }
```

```
        Manipulations+=[Manipulation]
    return Manipulations

**********************************************************
*                    Summarising results                 *
**********************************************************
def isImprovement(InitialAllocation, FinalAllocation, Preferences):
    IndecesOutcome1 = sorted([Preferences.index(i) for i in
                                        InitialAllocation])
    IndecesOutcome2 = sorted([Preferences.index(i) for i in
                                        FinalAllocation])
    if IndecesOutcome1==IndecesOutcome2:
        return 'no change'
    #if different length, assign the worst outside options to the
                                        shoter list
    if len(IndecesOutcome1)<len(IndecesOutcome2):
        IndecesOutcome1 +=[len(Preferences)]*(len(IndecesOutcome2)-
                                            len(IndecesOutcome1))
    if len(IndecesOutcome2)<len(IndecesOutcome1):
        IndecesOutcome2 +=[len(Preferences)]*(len(IndecesOutcome1)-
                                            len(IndecesOutcome2))

    IndecesTupled = list(zip(IndecesOutcome1,IndecesOutcome2))
    WeakComparison  = [x[0]>=x[1] for x in IndecesTupled]#if x[0]>x[1
                                        ], then the new allocation is
                                        better
    nWeaklyDominantPairs  = sum(WeakComparison)
    if nWeaklyDominantPairs ==len(WeakComparison):
        return 'Yes'
    else: return 'No'

def ManupulationResultSummary(Manipilations):
    Summary=[]
    for Manip in Manipilations:
        InitialAllocation = []
        ManipulatedAllocation = []
        iManipulator = Manip['Manipulator']
        ReportedPreferences = Manip['Reported preferences']
        for st in Manip['Students']:
            InitialAllocation += [[i for i in st['WaitingList']+st['
                                            ForcedCourses']]]
        for st in Manip['ManipStudnets']:
            ManipulatedAllocation += [[i for i in st['WaitingList']+
                                            st['ForcedCourses']]]
        #find manipulator's outcomes
```

```
            manInitAlloc = InitialAllocation[iManipulator]
            manFinAlloc = ManipulatedAllocation[iManipulator]
            manPreferences = Manip['Students'][iManipulator]['Preferences
                                                    ']
            AllocationImproved = isImprovement(manInitAlloc, manFinAlloc,
                                                     manPreferences)
            Sum= {
                'Allocation': Manip['Allocation'],
                'Manipulator': iManipulator,
                'Reported Preferences': ReportedPreferences,
                'InitialAllocation': InitialAllocation,
                'ManipulatedAllocation': ManipulatedAllocation,
                'Allocation of manipulator improved': AllocationImproved
            }
            Summary+=[Sum]
        return Summary

def ManipulationsComparison(ManipulationsDA, ManipulationsModDA):
    ManDA = ManupulationResultSummary(ManipulationsDA)
    ManModDA = ManupulationResultSummary(ManipulationsModDA)
    comparison = {
        'DA not ModDA': 0,
        'not DA, modDA': 0,
        'neither': 0,
        'both': 0,
        'total':0,
        'Same outcome in both': 0,}
    for i in range(len(ManDA)):
        comparison['total']+=1
        mDA = ManDA[i]
        mModDA = ManModDA[i]
        DAimprovement = mDA['Allocation of manipulator improved']
        ModDAimprovement = mModDA['Allocation of manipulator improved
                                                    ']
        if DAimprovement == 'Yes' and ModDAimprovement == 'Yes':
            comparison['both']+=1
            if mDA['ManipulatedAllocation'][mDA['Manipulator']] ==
                                                mModDA['
                                                ManipulatedAllocation'
                                                ][mModDA['Manipulator'
                                                ]]:
                comparison['Same outcome in both'] += 1
        elif DAimprovement == 'Yes':
            comparison['DA not ModDA']+=1
        elif ModDAimprovement == 'Yes':
```

```
                comparison['not DA, modDA']+=1
            else: comparison['neither']+=1
        return comparison

def ManipulationAveragesAndMedians(SummarisedComparisons):
    for res in SummarisedComparisons

**********************************************************
*                       Simulation                       *
**********************************************************

nStudents = 4
nCourses = 4
kCourses = [2]*nStudents #exact number of courses to be allocated
Quotas = [2]*nCourses
iterations = 10000
AllComparisons = []
AllManipulationsDA = []
AllManipulationsModDA = []
for iter in range(iterations):
    Students, Courses = GenerateData(nStudents, nCourses, kCourses,
                                     Quotas)
    Comparisons=[]
    CurrManipulationsDA = []
    CurrManipulationsModDA = []
    Priorities = [i for i in range(nStudents)]
    random.shuffle(Priorities)
    for iManipulator in range(nStudents):
        ManipulationsDA = ManipulateDA(deepcopy(Students), deepcopy(
                                       Courses), iManipulator)
        CurrManipulationsDA +=[ManipulationsDA]
        ManipulationsModDA = ManipulateModDA(deepcopy(Students),
                                       deepcopy(Courses),
                                       Priorities, iManipulator)
        CurrManipulationsModDA+=[ManipulationsModDA]
        Comparison = ManipulationsComparison(ManipulationsDA,
                                       ManipulationsModDA)
        Comparisons += [Comparison]
    AllManipulationsDA += [CurrManipulationsDA]
    AllManipulationsModDA += [CurrManipulationsModDA]
    TheComparison={}
    for key in Comparisons[0].keys():
        TheComparison[key]=(sum([x[key] for x in Comparisons]))
    AllComparisons +=[TheComparison]
```

```
SummarisedComparisons={}
for key in AllComparisons[0].keys():
    SummarisedComparisons[key]=[x[key] for x in AllComparisons]

for key in SummarisedComparisons:
    SummarisedComparisons[key] = [SummarisedComparisons[key][i]/
                                        SummarisedComparisons['total']
                                        [i]*100 for i in range(len(
                                        SummarisedComparisons[key]))]
TheAverages = {}
for key in SummarisedComparisons:
    TheAverages[key] = sum(SummarisedComparisons[key])/len(
                                        SummarisedComparisons[key])
TheMedians = {}
for key in SummarisedComparisons:
    TheMedians[key] = median(SummarisedComparisons[key])
TheMax={}
for key in SummarisedComparisons:
    TheMax[key] = max(SummarisedComparisons[key])
TheMin={}
for key in SummarisedComparisons:
    TheMin[key] = min(SummarisedComparisons[key])
```