UTRECHT UNIVERSITY

BACHELOR THESIS

School matching in Amsterdam: Top Trading Cycles reconsidered

Author:
Simone VAN BRUGGEN

Supervisor:
Dr. J.M. BROERSEN

A 7.5 ECTS thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Faculty of Humanities

July 4, 2017

Abstract

Simone VAN BRUGGEN

School matching in Amsterdam:
Top Trading Cycles reconsidered

For years, the matching of students to schools in Amsterdam has been a tough challenge. Especially in 2015, when the matching algorithm DA-MTB was used, many families were displeased with the results.

By running simulations using real data from the Amsterdam matching process, the different matching mechanisms can be compared. Boston, DA-MTB, DA-STB and TTC are analysed on their theoretical properties and performance on efficiency and fairness.

The Boston mechanism performs best in placing students at their first choice, but does place many students at a school they do not prefer. DA-MTB maximizes the number of students assigned to a school in their top-5, but assigns the smallest number of students to their first choice. When comparing DA-STB and TTC, they create a similar assignment, but TTC places more students at schools for every rank in the preference list and creates a Pareto efficient assignment. TTC can be adapted to the admission problem in which school-specific priorities exist by applying its assigning process only to a subset of the students. TTC does create instances of justified envy, but given its properties and results seems a better fit to the Amsterdam school problem than DA-STB.

Keywords: school choice, allocation mechanism, Boston mechanism, Deferred Acceptance mechanism, Top Trading Cycles, strategy-proofness, Pareto efficiency, fairness

Contents

1 Introduction

1.1 Introduction

After graduating primary school, students have to make an important decision: which secondary school do they want to attend? In Amsterdam, this choice is even more difficult, since the capacities of popular schools are not sufficient for the number of applicants. This leads to the complicated issue of assigning students to schools, which is one of the most studied problems in market design.

Mechanisms to match students to schools can be as simple as a lottery, but many more sophisticated algorithms exist. A variation on a simple lottery was used in Amsterdam for many years. The *Boston mechanism*[1] assigns students to their school of preference by drawing lots. If a student is not assigned in the first round, she has to pick a preference from the schools that still have capacity and draw lots again. This process continues until every student is assigned. This leads to many students being placed at their favourite school, but being unlucky in the first round drastically restricts a student's choice and often leads to being placed at a school they do not prefer.

In 2015, a mechanism called *Deferred Acceptance* (DA) was implemented. In this algorithm, which works similar to the Boston mechanism, students get placed based on a lottery, which is different for each school. The main difference with Boston is that the actual assigning is deferred until all students are placed. Thus, being chosen in the first round does not guarantee a spot at that particular school. Because of the working of DA, students need to hand in a list of preferences whereas Boston only demands a single choice per round. Handing in an extensive preference list is more demanding for students. For example, they have to visit more information days of schools which is time-consuming.

The results of DA were mixed, as it placed many students at a school in their top-3, but also left a notable part of the students unassigned. Furthermore, there was a considerably large group of students who wanted to switch schools with each other. This even led to a lawsuit. The judge decided that under no circumstances switching should be allowed, since this would lead to strategic choosing of preferences in the future. For many parents, this was hard to grasp, because the switching of places seemed like an obvious thing to do and it is hard to see the consequences of switching without knowledge about the algorithms involved. Furthermore, many families prefer to have some control over the outcome of the matching and do not view strategising as a bad thing. This property of strategy-proofness, which simply means that strategic choices do not lead to a better result, is a preferable property of matching mechanisms as it makes the assignment process more clear. Giving true preferences is the best strategy, which is useful for school boards and less stressful for students

[1] We will often refer to the Boston mechanism as *Boston* for brevity.

since they do not have to worry about their strategy. Therefore, strategy-proofness was a strong requirement from the municipality for the matching mechanism [8].

Because the results of DA led to many complaints, OSVO[2] initially decided to switch back to Boston. This choice led again to protest from parents; they started a petition for 'matching 2.0' [16] which OSVO accepted. Matching 2.0 is a version of Deferred Acceptance in which students have one lottery number for every school. As a result, more students got accepted on their most preferred school. Even though the results of matching 2.0 were received as positive and promising, the mechanism still has its shortcomings. More students than in 2015 got allocated on a school that is not in their top-5 and a substantial number of students ended up at a school they did not desire. Furthermore, parents and students found the system complicated and were not satisfied with the long preference lists they have to come up with. Especially in case of a low lottery number, students have to hand in lenghty preference lists.[3]

Apart from satisfying parents and students, the mechanism also needs to fit requirements imposed by the city counselor and the municipality. For example, they require that the system leads to no strategic behaviour and true preference lists, and gives no reason for students to switch their position in the resulting allocation.

Ideally, there exists an algorithm that manages to both satisfy the wishes of parents and students, and the requirements given by the municipality. As we will see, finding a 'perfect' system is impossible and trade-offs need to be made between desirable properties. The aim of this thesis is to clarify the matching process and the difficulties in finding an ultimate system. We will propose a different algorithm called Top Trading Cycles to solve the matching problem, and we will analyze and compare the various algorithms to provide more insight in their performance.

1.2 Context and A.I.

The matching of students to schools is a classic example of a problem in which market design, game theory and social choice come together. Research about this subject is also relevant for the field of artificial intelligence (A.I.), as it involves studying and formalising human behaviour. The gap between the perfect mathematical world and the real world makes it hard to predict how humans will behave in a game theoretic setting, which ideally consists of rational agents. The behaviour of (intelligent) rational agents is a well-studied subject in artificial intelligence. To understand the way people react to and use a matching process such as in Amsterdam involves a deeper understanding of the human rationale, which is also of great importance in artificial intelligence research.

[2]OSVO (http://www.osvo.nl) is an association of school boards of secondary schools in Amsterdam and is responsible for the central admission of new students.

[3]This topic was also discussed in the Dutch news and led to much criticism: https://www.nrc.nl/nieuws/2016/03/18/schoolkeuze-amsterdam-is-studie-op-zich-1599569-a785880

2 Theoretical background

In this chapter, the different ways of matching students to schools will be analysed in more depth. We will look into the game theoretic properties of the mechanisms involved and discuss the assignment process of these mechanisms.

2.1 Problem statement

Our school allocation problem is focused on assigning a finite set of students $N = \{1...n\}$ to a finite set of schools S, in which every school a has a capacity q_a. Each student i has a preference P_i over the schools in S. N is assigned to S using an *allocation mechanism*.

Definition 2.1.1. (Svensson [15]), An **allocation mechanism** is a mapping f from $\mathcal{U}$ to the set of allocations, i.e. every preference profile is mapped on an allocation of indivisible goods.

An allocation mechanism creates an assignment of students to schools, based on a many-to-one matching of students and schools. From now on, we will use *assignment* and *matching* interchangeably.

2.1.1 Properties

Every allocation mechanism creates different (non-unique) matchings, due to their assignment procedures. By distinguishing several well-studied properties for allocation mechanisms, we can compare these mechanisms and predict the character of their outcome. We discern several properties for allocation mechanisms, which are formalisations of the intuitive features of these mechanisms. As we will see, these properties are desirable but not compatible.

Strategy-proofness

A mechanism is strategy-proof if it cannot be manipulated by false preferences. In such a mechanism, reporting true preferences is the dominant strategy.

Definition 2.1.2. A mechanism is called **strategy-proof** if it is impossible for a user to get a better outcome by not giving their true preference.

If a school allocation mechanism is not strategy-proof, a student can be better off handing in a preference list that differs from her true preference. This is generally viewed as an undesirable quality, because deciding on a preference list would become even more time-consuming: not only would students have to choose the schools of their preference, they would also need to decide on their strategy. Having to choose strategically could also hurt students from disadvantaged backgrounds, who might not get help from their parents in choosing in a strategic way. Moreover, true preference lists are useful for schools and related parties, as these provide insight in the actual numbers concerning applications. Strategic choosing can of course also be viewed more optimistically. Students and parents feel like they have more control of the outcome of the matching if they can end up at a different school when choosing strategically. They don't see the option for strategising as an unwanted property, and favour this choice over the randomness of a lottery number.

Pareto efficiency

A matching is called Pareto efficient[1] if there exists no other matching that Pareto dominates it. For a matching to be Pareto dominated, there has to exist another matching in which the outcome is just as good for all students and is better for at least one student.

Definition 2.1.3. A matching μ **Pareto dominates** a matching ν if every student in μ is at least as good off and there is some student i that is better off in μ than in ν.

Definition 2.1.4. An allocation μ is **Pareto efficient** if there exists no allocation ν that Pareto dominates μ.

Simply put, if a matching is efficient, it assigns as many students as possible to the school they prefer. Clearly, efficiency is a very appealing property for an allocation mechanism. An inefficient matching gives rise to negative reactions among users, since they feel there could have been a better result. Since this type of efficiency is only concerned with the outcome, it is also referred to as *ex-post* efficiency. We also define *ex-ante* efficiency. Ex-ante efficiency is efficiency as it is realised before any lottery is instantiated.

Definition 2.1.5. (Abdulkadiroğlu et al. [1]) An allocation is **ex-ante Pareto efficient** if it is impossible to reallocate probability shares of different schools in such a way that expected utility of some students increases without reducing the expected utility of other students.

Stability

When a students gets assigned to a school a but prefers another school b on which she has higher priority than an assigned student on b, this results in an unwanted outcome. This outcome is called *unstable*. Our intuitive notion of fairness for an assignment corresponds with the more formal definition of stability.

Definition 2.1.6. (Gale and Shapley [14]) An assignment of applicants to colleges will be called **unstable** if there are two applicants α and β who are assigned to colleges A and B, respectively, although β prefers A to B and A prefers β to α.

For a solution to be stable, it should simply not be unstable. In other words, there should be no unmatched student-school pair (i, s) where student i prefers school s to her assignment and school s prefers student i to one or more of its admitted students. Clearly, since we are interested in secondary schools, it makes little sense to talk about schools preferring students[2]. In combination with priorities, we can reword the stability definition to: There should be no unmatched student-school pair (i, s) where student i prefers school s to her assignment and she has higher priority than some other student who is assigned a seat at school s [1].

[1]We will often state that a matching is *efficient*, which refers to *Pareto efficiency*.

[2]A few schools do prefer certain students (e.g. art schools), but they have their own selection of students independently of the central admission.

Fairness

To decide if an assignment is fair, one can measure the occurrences of justified envy.

Definition 2.1.7. (Morrill [10]) A student i is said to have **justified envy** if there is a school a such that i prefers a to her assignment, and i has higher priority at a than one of the students assigned to a.

Definition 2.1.8. An assignment μ is called **fair** if no student in μ has justified envy.

In a stable matching, all occurrences of justified envy are eliminated and the assignment is fair.

2.1.2 Impossibility theorems and trade-offs

Ideally, all of the properties above would be satisfied in an allocation mechanism. However, Roth [13] shows that efficiency and fairness are incompatible. Furthermore, Kesten [6] shows that an allocation mechanism cannot be strategy-proof if it is efficient and free of justified envy. Consequently, there can not exist a Pareto efficient and strategy-proof mechanism that selects an efficient and justified envy-free allocation if such an allocation exists. This means that the properties of strategy-proofness, Pareto efficiency and fairness need to be balanced when choosing an allocation mechanism. Depending on the needs and wishes of the users, an algorithm that satisfies two of these three properties can be chosen.

2.2 Overview algorithms

Each allocation mechanism has its own subset of the aforementioned properties. We will now discuss the different algorithms that have been implemented in Amsterdam in the past years.

2.2.1 Boston

In 2005, Amsterdam started using an centralized application and admission system based on the Boston allocation algorithm. The Boston algorithm runs as follows:

1. Every student hands in one preference. All students get placed at the school of their preference. If the capacity of the school is insufficient for all applicants, a central lottery decides which students are placed.

2. Students that are not yet placed, choose a new preference from schools that still have capacity. In case there are too many applicants again, students get placed or rejected based on their lottery number.

3. Step 2 is repeated until all students are assigned.

The results of the Boston mechanism were mixed. Many students got placed at their first choice, but a considerably large group of students was not placed at a school in their top-3. Furthermore, the Boston algorithm is known not to be strategy-proof. Students expecting to have little chance of being accepted at a certain school might want to apply for another school. As a result, the mechanism is not truth-telling. The allocation is also not ex-post Pareto efficient, since users are likely to misrepresent their preferences and thereby create an assignment that is not efficient. However, Abdulkadiroğlu et al.[1] show that Boston is ex-ante efficient, as it gives students the opportunity to express the intensity of their preferences by deciding whether or not to make a strategic choice.

2.2.2 Deferred Acceptance

Based on a simulation by De Haan et al. [3], the Boston system got replaced by a mechanism based on the Deferred Acceptance (DA) mechanism. Deferred Acceptance is the central concept used in the stable marriage algorithm proposed by Gale and Shapley [14], which is based on men proposing to women to create stable matches. In each round, women defer accepting a proposal until the end of the round, leading to more preferred choices. This algorithm has been widely applied to college application. To use the algorithm for the school admission problem, a lottery has to be used to create a preference order over students for each school. We distinguish two types of lotteries: Single-Tie Breaking (DA-STB) and Multiple-Tie Breaking (DA-MTB). Using Single-Tie Breaking, every student gets one lottery number which is used for every school, while in Multiple-Tie Breaking every student has to draw lots for every school.

The DA algorithm, for both Single-Tie and Multiple-Tie Breaking, runs as follows:

1. Every student hands in her preference list. Every student gets placed on her first choice. If the capacity of the school is not sufficient for the number of applications, a lottery decides which students get rejected and which students are placed provisionally.

2. Rejected students get assigned to the next school on their preference list.

3. In case of insufficient capacity a lottery decides who gets placed. In this lottery, every student who was provisionally placed takes part.

4. Until all students are assigned, repeat step 2 and 3.

What is problematic about using DA for school allocation is the fact that the algorithm is designed for a system in which both parties have a preference, as is the case with marriages and college application. Since only the student preference is important in our case, this leads to inefficient results. This is especially the case for DA-MTB, which the unsatisfactory results of the matching in Amsterdam in 2015 confirmed. Only 74% of the students got assigned to their first choice [3].

3 Top Trading Cycles

Since the results of the DA mechanism are not optimal, we will try to apply a different solution to the school allocation problem. We have seen that a trade-off between strategy-proofness, efficiency and fairness needs to be made. DA is known to be strategy-proof and fair, but efficiency is a desirable property that should not be overlooked. In this chapter, we will introduce the Top Trading Cycles algorithm, a mechanism that is proven to be strategy-proof and efficient.

3.1 Algorithm

Introduced by Shapley and Scarf [14], TTC was designed for the Shapley-Scarf housing market. Every student in this market already owns a house. To improve the allocation, the following procedure is run:

1. Each student points to her favourite house.
2. Every house points to its owner.
3. Since there exists a finite number of students, there must exist a cycle. For every cycle, assign the student to the house she is pointing to and remove the student and house from the set.

Although assigning houses differs from assigning schools, the algorithm can easily be changed to fit the school assignment problem. Since students do not own a school, a school points at the student with the highest priority at the school. A student points at her favourite school. The algorithm continues as described above.

We will look at an example that shows the difference between DA and TTC.

P_i	P_j	P_k	$\succ_a$	$\succ_b$	$\succ_c$
b	a	a	i	j	k
a	b	b	k	k	j
c	c	c	j	i	i

TABLE 3.1: Preferences and priorities

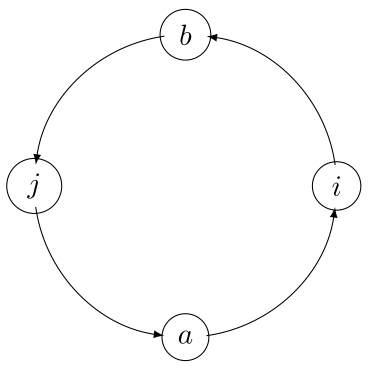

FIGURE 3.1: Cycle during TTC assignment

Solving this assignment using DA leads to the following assignment:

$$\mu = \begin{pmatrix} i & j & k \\ a & b & c \end{pmatrix}$$

Solving the same assignment using TTC, we obtain two cycles. The first one is shown in figure 3.1. After assigning this cycle, only k and c remain to be assigned.
This creates the assignment ν below. Because student i and j are better off in this assignment than in μ and student k is just as well off, ν Pareto dominates μ.

$$\nu = \begin{pmatrix} i & j & k \\ b & a & c \end{pmatrix}$$

This example shows the difference in efficiency between DA and TTC. TTC makes a better outcome possible by trading the priorities of student i and j. Abdulkadirğlu and Sönmez [1] give a general distinction between DA and TTC: when fairness is more important than efficiency, DA is the better choice, and when efficiency is prefered over the elimination of justified envy, TTC should be chosen.

3.2 Objections to TTC

Since it has been shown that TTC is not fair, this would seem like an obvious and acceptable argument to choose a different algorithm. However, this seems to be a minor concern among school boards when choosing an allocation mechanism. The Amsterdam research group rejected TTC [4], because it would be unsuitable to handle school-specific priorities given that these priorities are non-transmittable. For example, the priority of a student who has siblings attending a certain school should not get priority on another school. Morrill [11] has shown that by using an adapted version of TTC this problem can be overcome. Furthermore, De Haan et al. state that without priorities, TTC would be equivalent to a version of DA-STB [4]. This claim does not seem to be supported by the existing literature[1]. It thus seems TTC should not be ruled out and should mainly be chosen or rejected on its mathematical properties.

3.3 Properties

We have argued that TTC is strategy-proof and Pareto efficient. These properties can easily be shown.

Strategy-proofness

When a student gets assigned at the end of step *i*, we could say she leaves the algorithm at step *i*. Since she points at the best available spot, she will get assigned to the best possible school s when reporting her true preferences. Schools that she prefers over s are no longer available in step *i*, as she would otherwise be pointing at those schools. She cannot change the cycles that have formed before step *i* by handing in false preferences, so these prefered schools will have left the algorithm at step *i*, independently of reporting true or false preferences. As a result, she can only harm herself by misrepresenting her preferences and reporting true preferences leads to the best possible outcome.

[1]However, Abdulkadirğlu et al. show that in case of a single priority ordering (i.e. a single lottery) TTC reduces to Random Serial Dictatorship [1], an algorithm which assigns students to schools in order of priority using a central lottery.

Pareto efficiency

A student leaving the algorithm at step 1 is assigned her first choice, and cannot be better off. Subsequently, any student who leaves at step 2 is assigned her top choice among the remaining schools. She can only create a better outcome by harming someone who was assigned in step 1.
As this is the case for each step, no student can be made better off without hurting a student who was assigned in an earlier step. This means that there exists no other assignment that Pareto dominates the outcome, and therefore that the created assignment is Pareto efficient.

3.4 Restricting tradeable priorities

Since we can use any decision rule for the pointing of students and schools, we can adapt our pointing rule for schools to restrict the trading of school-specific priorities. If some school a has capacity q_a and S_a are the q_a highest ranked students at a, we have a (possibly empty) subset of students, $R_a \subset S_a$, who should not be allowed to trade their priority. We obtain a set of unrestricted students $S_a \setminus R_a$. Now we let school a no longer point at the student with the highest priority at a, but at the unrestricted student with the highest priority at a. Restricted students get assigned to their prioritised school at the beginning of the algorithm, to prevent having them distributed randomly in case of insufficient capacity. This effectively eliminates trading of school-specific priorities. Since we can solve this problem, TTC can be applied to the Amsterdam assignment problem in a straightforward fashion.

4 Data modeling

To test the algorithms and ideas introduced in the previous chapters, we will run simulations using real data from the Amsterdam matching process. The results of these simulations using real data can be used to gain insight in how the allocation mechanisms will perform in real life.

4.1 Dataset

The dataset used in the experiments is provided by OSVO. They collected the data of all students who started secondary school in 2016. This was the year DA-STB was first used in the actual assignment of the students. The resulting Excel file contains the anonymized allocations and preferences of these students. It also contains information about priorities, lottery numbers and preselection.

Before using the data for the simulations, several adjustments have been made. School names have been substituted by IDs and the priority column is changed from a string value to a binary value, since we are only interested in whether or not a student has priority at her first school and not in the reason for this priority. Students who are preselected are removed from the dataset, since they are not relevant for our simulations.

4.2 Simulations

The simulations are run in Python, which is chosen both for its ease in handling data sets and modeling our problem. The packages used for carrying out and analysing these simulations are Pandas and Numpy.

Implementing the algorithms is quite straightforward. A minor challenge was modeling the Boston mechanism using the available dataset. Since this dataset contains the results of DA-STB, it contains the preference list students decided on beforehand. Boston lets students decide again on their new preference if they are not selected in the first round. To simulate this as accurately as possible, we use the first available school on the preference list of a student who got rejected as their new choice, since this would probably be the school they would prefer if they could choose again. However, this does not completely cover the side effects that occur in reality, when students can strategically choose a new school after the first round. For now, we will assume that our simulations might differ from reality, but that the difference is small enough to properly compare the algorithms.

To minimize the side effects of the random lottery for students which is used in each algorithm, every simulation is run 100 times with different lottery numbers. The results are based on the average of these simulations.

The implementations of the algorithms can be found in Appendix A.

4.3 Evaluation measures

The algorithms will be measured and compared using the following metrics:

- Efficiency: measured by shares of students placed in one of their most-prefered n schools.
- Fairness: measured by the number of occurrences of justified envy.

We will also measure the number of trades made by students with school-specific priorities in the Top Trading Cycles algorithm. This analysis is omitted for the other algorithms since they do not use the concept of trading priorities. As we are only interested in the results of our simulations and the algorithms are sufficiently fast for our purposes, the speed of our implementations is not examined in the experiments.

5 Results

In this section, we will discuss the performance of the algorithms Boston, DA-STB, DA-MTB and TTC using simulations. The school assignment is simulated using the data and implementation as discussed in chapter 4. First, the ex-post efficiency of the algorithms is compared. This is done by comparing the shares of students placed in one of their most-prefered n schools.

TABLE 5.1: Placement of students on rank of their assigned school as percentage of total number of students

	Boston	DA-STB	DA-MTB	TTC
1st choice	0.865	0.831	0.735	0.844
2nd choice	0.057	0.102	0.165	0.087
3rd choice	0.024	0.028	0.058	0.033
4th choice	0.011	0.013	0.026	0.013
5th choice	0.008	0.008	0.011	0.012
6th choice	0.008	0.007	0.004	0.006
7th choice	0.007	0.006	0.001	0.006
8th choice	0.005	0.005	0.001	0.004

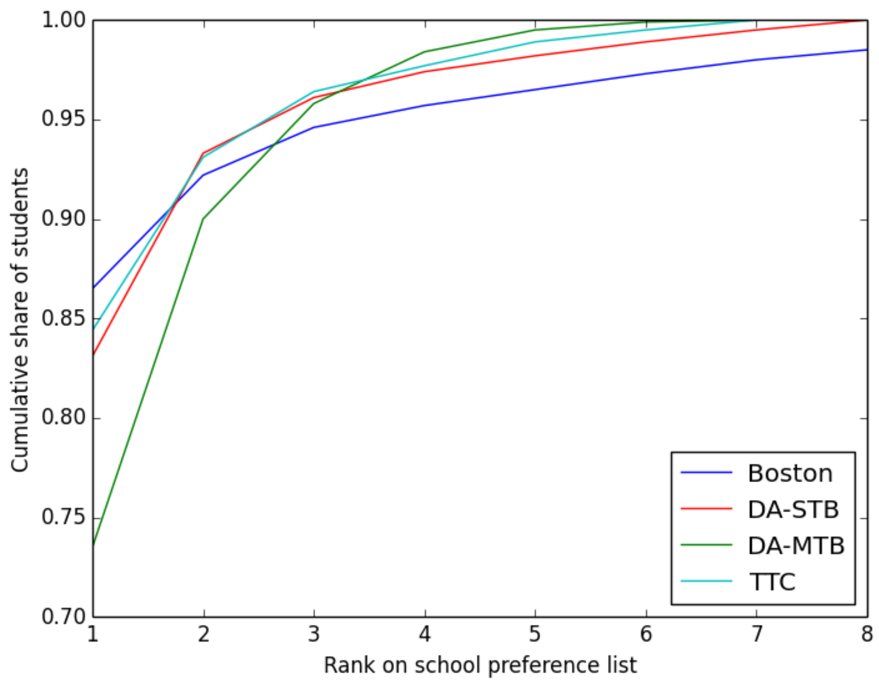

FIGURE 5.1: Distribution of students to schools on preference list

Figure 5.1 shows the cumulative shares for the different algorithms, based on the results from table 5.1. We see that Boston clearly outperforms the other algorithms in placing students at their first choice school, but loses this lead and performs worse than the other mechanisms in placing students in their top-n for $n \geq 2$. Another immediate observation is the low number of students placed at their first choice school by DA-MTB. However, DA-MTB does run best in placing students in a school in their top-n for $n > 3$. TTC and DA-STB perform similar, although TTC places more students at schools for each rank in the preference list than DA-STB. TTC performs 0.62% better than DA-STB in placing students in their top-3. While this seems like a small percentage, this does mean an additional 115 students get placed at one of their top-3 choices in the 2016 Amsterdam matching. Both TTC and DA-STB run worse than Boston for placing students at their first choice, but outperform Boston in top-n for $n \geq 2$. Based on our results, TTC gives the best average result of the algorithms and is a better choice than DA-STB. Depending on specific wishes and preferences, one of the other algorithms is preferable. Boston would be the best fit if one were to maximise the number of students placed at their first choice, whereas DA-MTB is preferable if the highest possible number of students should be assigned to a school in their top-5.

To evaluate the fairness of the algorithms, we measured the number of instances of justified envy. Since there were no instances of justified envy for Boston and both the DA mechanisms in the simulations, we omitted these algorithms from our results table. As we can see in the results in table 5.2, the number of instances of justified envy grows after restricting the school-specific trades. In the unrestricted version of TTC, we consider all students for trading and assigning. When restricting the mechanism, we consider only a subset of these students. Hence, it makes sense to have a larger number of occurrences of justified envy. In total, the occurrences of justified envy increased with 15.9%.

TABLE 5.2: Average number of trades and occurrences of justified envy before and after restricting of trading

	Before restricting	After restricting
School-specific priority trades	31	0
Occurrences of justified envy	29.5	34.2

The biggest concern in not considering TTC turns out not to be a frequent issue: in our simulation, only 2.9% of the students with non-tradeable priorities got assigned to a school where they did not have priority, which is 0.46% of all students. Even though this is a small part of the students, it is still an undesirable effect which we managed to eliminate using the restricted variation of TTC. No instances of trading of school-specific priorities took place, in exchange for a small increase in occurrences of justified envy.

6 Conclusion

6.1 Conclusion

The problem of assigning students to schools has proven to be a difficult challenge, for which unfortunately no ideal solution exists. Our main conclusion would therefore be a safe one, namely that each of the algorithms we have discussed and analysed has its benefits and disadvantages and that there exists no ideal solution to the Amsterdam school admission problem.

However, as we have seen, given certain choices, some mechanisms perform better than others. When fairness is desired, Deferred Acceptance with Single-Tie Breaking seems to be the best choice. DA-STB has good overall results and is a useful compromise between DA-MTB and Boston. If one were to maximise the number of students admitted to their first choice school, the Boston mechanism would lead to the best results. Following from its theoretical properties and the results of our simulation, Boston has quite some disadvantages but clearly outperforms other algorithms in assigning students to the school of their first choice. In terms of politics, the Boston mechanism would correspond with a conservative liberal view of school assignment, in which a great outcome for most students is prefered over having a moderate result for all students, despite harming a small group. Another scenario might be that as many students as possible should be assigned to a school in their top-5, even if this means less students are assigned to their first choice; in this case, Deferred Acceptance with Multiple-Tie Breaking should be chosen. This could be seen as a more social approach to the matching process.

The research on matching procedures in Amsterdam did not even take the Top Trading Cycles algorithm into account. Since our results are promising, it would be very useful for future research and simulations to consider this algorithm. The objection that TTC would not be suitable for the matching in Amsterdam because of the possibility of trading of school-specific priorities can easily be overcome by using the adapted restricted variation of TTC we proposed. The mechanism has some valuable properties, most notably its Pareto efficiency. Based on the results of our simulations, TTC is a very interesting contender for DA-STB. The idea behind TTC might also be easier to grasp for parents and students, as its basic idea is quite intuitive.

Apart from the theoretical limits of these mechanisms, another challenge lies in finding a solution that unites the ideal theoretical world of mathematics and game theory, and the messy everyday world in which these mechanisms are used. In the real world, users are not necessary predictable agents who always behave rational. We cannot expect users not to make strategic choices and give true preferences, simply by using a strategy-proof system. If a user is convinced that they can alter the results of a strategy-proof system, they will do so. Likewise, using a system in which strategising is possible does not mean that people will make strategic choices. They might not receive the best possible outcome, but might not even be aware of this. This kind of behaviour is not something game theorists or, more specifically, researchers of this school admission problem can predict. This (possible) irrationality makes the design of an optimal system even more difficult.

Finally, an important solution to the matching problems in Amsterdam lies in capacity. In the end, since there exists no ideal key to this problem in game theory, there lies a task for school boards and the municipality of Amsterdam to come up with solutions for the growing number of applicants and the insufficient capacities of schools.

6.2 Future research

Given the extensive literature and research in the market design and social choice area, there are many ideas that can still be applied to the school admission problem and specifically tested on the Amsterdam case.

One recommendation would be to further analyze the Top Trading Cycles algorithm. The basic variant, as it is implemented in this thesis, makes more trades than absolutely necessary because some students that have high priority at their most prefered school can be assigned immediately and can skip the trading process. *Clinch and Trade* is a variation of TTC which eliminates unnecessary trades [11].

Other solutions to the school admission problem might also be found in other areas, such as mathematical optimisation [12]. Results from optimisation are less directed at the theoretical properties such as in game theory, but focus on maximising the outcome of an algorithm. Clearly, this might very well be useful in assigning as many students as possible to their prefered schools. Because its focus differs from the game theoretic approach, this may lead to different and interesting results. The school admission problem could be modeled as a bipartite graph consisting of students and schools, in which students and schools are connected by weights of their preferences. Finding the shortest path with minimum cost, using for example the Hungarian Algorithm [7], would lead to a complete assignment with lowest cost in preferences. It would be interesting to investigate how a result like this corresponds to the formal definitions and properties of game theory.

It may also be interesting to study in more detail how the capacities of schools should be altered to minimise rejecting students. Simulations of the matching process in which the capacities of popular schools get extended could be of aid for school boards to see if increasing the capacities is a feasible solution.

Finally, an important part of research lies in bridging the gap between research and the public understanding of the matching process. Questionnaires could be useful in gaining a deeper insight in the public opinion and can lead to a solution to the admission problem that better fits the ideas and views of its users. For example, it would be interesting to ask families which properties of the matching mechanisms they value most. As we have stated in our conclusion, part of the difficulty of finding a solution to the school admission problem is the irrationality of users of the system. Because users decide a great deal of the result of a matching, studies in areas such as sociology and philosophy can be useful in understanding the human rationale and can lead to better predictions of human behaviour in a game theoretical setting.

Bibliography

[1] Abdulkadiroğlu, A. and Sönmez, T. “School Choice: A Mechanism Design Approach”. In: *American Economic Review* 93.3 (June 2003), pp. 729–747.

[2] Abdulkadiroğlu, A. et al. “Minimizing Justified Envy in School Choice: The Design of New Orleans’ OneApp”. In: *NBER Working Paper* w23265 (2017).

[3] De Haan, M. et al. *Schoolkeuze voorgezet onderwijs in amsterdam: Verslag van een simulatiestudie*. 2014. URL: http://www.verenigingosvo.nl/wp-content/uploads/2014/04/RapportSimulaties.pdf.

[4] Gautier, P. et al. “The performance of school assignment mechanisms in practice”. In: (2016).

[5] Gautier, P. et al. *Wie gaat naar welke school – en waarom?* 2015. URL: https://www.onderwijsconsument.nl/dit-de-regie-die-ouders-willen-hebben/.

[6] Kesten, O. “School choice with consent”. In: *Quarterly Journal of Economics* 125 (2010), pp. 1297–1348.

[7] Kuhn, H.W. “The Hungarian Method for the Assignment Problem”. In: *Naval Research Logistics Quarterly* 2 (1955), pp. 83–97.

[8] Kukenheim, S. *Uitkomst Evaluatie en Tevredenheidsonderzoek Matching PO-VO*. 2015. URL: http://www.stichtingvsa.nl/pdf/20151006/bijlage_1_20151005_WH_brief_evaluatie_matching_PO-VO.pdf.

[9] Morrill, T. *Making Efficient School Assignment Fairer*. 2013.

[10] Morrill, T. “Making just school assignments”. In: *Games and Economic Behavior* 92 (2015), pp. 18–27.

[11] Morrill, T. *Two Simple Variations of Top Trading Cycles*. 2014.

[12] Roberts, F.S. and Tesman, B. *Applied Combinatorics*. Chapman and Hall/CRC, 2009.

[13] Roth, A.E. “The Economics of Matching: Stability and Incentives”. In: *Mathematics of Operations Research* 7.4 (1982), pp. 617–628.

[14] Shapley, L. and Scarf, H. “On Cores and Indivisibility”. In: *Journal of Mathematical Economics* 1 (1974), pp. 23–37.

[15] Svensson, L. “Strategy-proof allocation of indivisible goods”. In: *Social Choice and Welfare* 16 (1999), pp. 557–567.

[16] Task group ’Petitie voor matching’. *Geen loting, maar aangepaste matching voor middelbare scholen in Amsterdam*. 2015. URL: http://www.onderwijsconsument.nl/wordpress/wp-content/uploads/Position-paper-voor-aangepaste-matching.pdf.

A Python implementation

Classes

```
class Student(object):
    def __init__(self, id, preferenceList, lotNr, priority, priorityID):
        self.id = id
        self.preferenceList = preferenceList
        self.preferences = list(preferenceList)
        self.lotNr = lotNr
        self.priority = priority
        self.priorityID = priorityID
        self.placed = False
        self.randomAllocation = False
        self.points_to = 0
        self.currentPoint = 0
        self.averageRank = 0

class School(object):
    def __init__(self, schoolID, capacity):
        self.schoolID = schoolID
        self.capacity = capacity
        self.students = []
        self.proposals = []
        self.lottery = []
        self.points_to = 0
        self.currentPoint = 0

class Matching(object):
    def __init__(self, students, schools):
        self.students = students
        self.schools = schools
```

Shared functions

```
def proposeFav(student, schools):
    preference = student.preferenceList[0]
    if not np.isnan(preference):
        school = schools[int(preference)-1].proposals
        student.preferenceList.pop(0)
        if school.capacity > len(school.students:
            school.append(student)
        else:
            return None
    return (schools, student)
```

```python
def randomAssign(notPlaced, schools):
    for s in notPlaced:
        randomSchool = np.random.choice(schools)
        # search new randomschool until school with capacity is found
        while len(randomSchool.students) >= randomSchool.capacity:
            randomSchool = np.random.choice(schools)
        randomSchool.students.append(s)
        s.randomAllocation = True
    return schools
```

Implementation Boston mechanism

```python
def boston(students, schools):
    notPlaced = []
    allStudentsAssigned = False
    i = 1
    while(not allStudentsAssigned and i < 15):
        match = matchingRound(students, schools, notPlaced)
        allStudentsAssigned = True
        numberNotAssigned = 0
        for student in students:
            if not student.placed:
                allStudentsAssigned = False
                numberNotAssigned += 1
        i+=1
    schools = randomAssign(notPlaced, schools)
    return Matching(match.students, match.schools)

def matchingRound(students, schools, notPlaced):
    for school in schools:
        school.proposals = []
    for s in students:
        if not s.placed:
            if s.preferenceList:
                proposal = proposeFav(s, schools)
                while proposal is None:
                    proposal = proposeFav(s, schools)
                    schools = proposal[0]
                    s = proposal[1]
            else:
                notPlaced.append(s)
    for school in schools:
        #add students that have priority
        for s in school.proposals:
            if s.priorityID == school.schoolID and s.priority == 1:
                school.students.append(s)
                s.placedSchool = school.schoolID
                school.proposals.remove(s)
                s.placed = True
        #add students in order of priority until the capacity is reached
        school.proposals.sort(key=lambda x: x.lotNr, reverse=True)
        for i in range(0, school.capacity - len(school.students)):
            if i < len(school.proposals):
                school.students.append(school.proposals[i])
                school.proposals[i].placed = True
        print school.schoolID, len(school.students)
    return Matching(students, schools)
```

Implementation Deferred Acceptance Single-Tie Breaking mechanism

```
def daStb(students, schools):
    notPlaced = []
    allStudentsAssigned = False
    for i in range(0,20):
        match = matchingRound(students, schools, notPlaced)
        allStudentsAssigned = True
    schools = randomAssign(notPlaced, schools)
    return Matching(match.students, match.schools)

def matchingRound(students, schools, notPlaced):
    for school in schools:
        school.proposals = []
    for s in students:
        if not s.placed:
            if s.preferenceList:
                proposal = proposeFav(s.preferenceList, s, schools)
                schools = proposal[0]
                s = proposal[1]
            else:
                notPlaced.append(s)
    for school in schools:
        #add all students back to the proposal-pool
        proposals = school.students + school.proposals
        for student in proposals:
            student.placed = False
        school.students = []

        #add students that have priority
        for s in proposals:
            if s.priorityID == school.schoolID and s.priority == 1:
                school.students.append(s)
                proposals.remove(s)
                s.placed = True

        #add students in order of priority until the capacity is reached
        proposals.sort(key=lambda x: x.lotNr, reverse=True)
        for i in range(0, school.capacity - len(school.students)):
            if i < len(proposals):
                school.students.append(proposals[i])
                proposals[i].placed = True
    return Matching(students, schools)
```

Implementation Deferred Acceptance Multiple-Tie Breaking mechanism

```python
def daMtb(students, schools):
    notPlaced = []
    allStudentsAssigned = False
    studentsToAssign = students
    lotteryArray = range(students[-1].id+1)
    # create random lottery for each school
    for school in schools:
        school.mtbLottery = random.sample(lotteryArray, len(lotteryArray))
    for i in range(0,30):
        match = mtbMatchingRound(studentsToAssign, schools, notPlaced)
        studentsToAssign = match.students

    schools = randomAssign(notPlaced, schools)
    return Matching(match.students, match.schools)

def mtbMatchingRound(students, schools, notPlaced):
    for school in schools:
        school.proposals = []
    for s in students:
        if not s.placed:
            if s.preferenceList:
                proposal = proposeFav(s.preferenceList, s, schools)
                schools = proposal[0]
                s = proposal[1]
            else:
                notPlaced.append(s)
    for school in schools:
        proposals = school.students + school.proposals
        for student in proposals:
            student.lotNr = school.mtbLottery[student.id]
            student.placed = False
        school.students = []

        #add students that have priority
        for s in proposals:
            if s.priorityID == school.schoolID and s.priority == 1:
                school.students.append(s)
                proposals.remove(s)
                s.placed = True

        #add students in order of priority until the capacity is reached
        proposals.sort(key=lambda x: x.lotNr, reverse=True)
        for i in range(0, school.capacity - len(school.students)):
            if(i < len(proposals)):
                school.students.append(proposals[i])
                proposals[i].placed = True
    return Matching(students, schools)
```

Implementation Top Trading Cycles mechanism

```
def ttc(students, schools):
    # initalization
    studentsToAssign = students
    priorityStudents = []
    for student in studentsToAssign:
        if student.priority == 1:
            priorityStudents.append(student)
            studentsToAssign.remove(student)
    for student in priorityStudents:
        priorityAssign(student, schools)
    lotteryArray = range(len(studentsToAssign))
    for school in schools:
        school.lottery = random.sample(lotteryArray, len(lotteryArray))
        school.points_to = studentsToAssign[school.lottery[0]]
    for student in studentsToAssign:
        student.points_to = student.preferenceList[0]
    allStudentsAssigned = False
    notPlaced = []
    while(not allStudentsAssigned):
        notPlaced += point(studentsToAssign, schools)
        for student in studentsToAssign:
            node = findCycle(student, studentsToAssign, schools)
            if(not node is None):
                assignCycle(node, studentsToAssign, schools)
        allStudentsAssigned = True
        for student in studentsToAssign:
            if not student.placed:
                allStudentsAssigned = False
    randomAssign(notPlaced, schools)
    return Matching(students, schools)

def priorityAssign(student, schools):
    school = schools[student.preferenceList[0]-1]
    school.students.append(student)

def point(students, schools):
    notPlaced = []
    for student in students:
        if student.placed:
            continue
        # students keep pointing to same school if it has capacity
        school = schools[int(student.points_to)-1]
        while school.capacity <= len(school.students):
            if student.currentPoint + 1 > (len(student.preferenceList) - 1)
                notPlaced.append(student)
                students.remove(student)
                for school in schools:
                    school.lottery.remove(max(school.lottery))
                break
            else:
                student.currentPoint += 1
                student.points_to = student.preferenceList[student.currentP
                school = schools[int(student.points_to)-1]
    for school in schools:
        while school.points_to.placed and school.currentPoint < len(school.
            school.currentPoint += 1
            nextStudent = school.lottery[school.currentPoint]
            if nextStudent < len(students):
                school.points_to = students[nextStudent]
    return notPlaced

# randomly find cycle in graph and return first student in cycle
def findCycle(firstNode, students, schools):
    cycleFound = False
    node = firstNode
    nodesSeen = []
    while not cycleFound:
        if node.placed:
            return None
        nodesSeen.append(node.id)
        node = nextNode(node, students, schools)
        if node.id in nodesSeen:
            cycleFound = True
    return node

def nextNode(node, students, schools):
    return schools[int(node.points_to)-1].points_to

def assignCycle(node, students, schools):
    nextStudent = node
    startReached = False
    while not startReached:
        school = schools[int(nextStudent.points_to)-1]
        school.students.append(nextStudent)
        nextStudent.placed = True
        nextStudent = school.points_to
        if nextStudent == node:
            startReached = True
```