

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

A Tool for Random Assignments

Dominik Spies

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

A Tool for Random Assignments

Ein Werkzeug für randomisierte Allokationen

Author:	Dominik Spies
Supervisor:	Prof. Dr. Felix Brandt
Advisor:	M. Sc. Johannes Hofbauer
Submission Date:	July 28, 2017

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich,

Dominik Spies

Acknowledgments

I want to thank everyone who helped and supported me in my studies in any way big or small. I am especially thankful to Paola who gave me valuable feedback on this thesis. The same goes for my advisor Johannes Hofbauer, without whom this thesis would not have been possible.

Thank you.

Abstract

There are many situations where a set of indivisible objects needs to be assigned to a group of agents. Common examples are college programmes, social housing or donor organs. What makes these problems tricky is that you cannot solve them by using the free market, since these objects cannot or must not be sold. In this thesis we will look at four different solutions to allocate objects to agents, whilst taking the agents' respective preferences into account, and analyse their axiomatic properties.

Contents

Acknowledgments	iii
Abstract	iv
1. Introduction	1
2. Literature	3
3. Formalisation	5
3.0.1. Preferences	5
3.0.2. Assignments	6
3.0.3. Stochastic Dominance	7
3.1. Axiomatic Properties	8
3.1.1. Optimality	8
3.1.2. Equal Treatment of Equals	9
3.1.3. Envy-Freeness	9
3.1.4. Popularity	10
4. Algorithms	12
4.1. Random Serial Dictatorship	12
4.2. (Extended) Probabilistic Serial	13
4.3. Popular Assignment Rules	15
4.4. Boston Mechanism	16
4.4.1. Naive Boston Mechanism	16
4.4.2. Adaptive Boston Mechanism	16
4.5. Axiomatic Properties	19
5. Software Design	20
5.1. Preferences and Profiles	20
5.1.1. Helper Classes	20
5.1.2. Scenario Files	21
5.2. Allocations	21
5.2.1. Allocation Files	21

5.3. Assignment Strategies	21
5.3.1. Implementations	22
5.4. Testing Axiomatic Properties	23
5.5. The GUI	23
5.5.1. The Main Frame	24
5.5.2. The Allocation Main Frame	25
5.6. Libraries	27
6. Profile Enumeration	28
6.0.1. Anonymity and Neutrality	29
6.0.2. Preference-transformations and Preference-duality	30
6.1. The Extended Profile Enumeration Algorithm	31
7. Profile Sampling	33
7.1. Impartial Culture	33
7.2. Spatial Model	33
7.2.1. Greedy distance clustering	33
7.3. Iterative Joining	34
8. Outlook	35
A. Appendix	36
A.1. Proofs	36
A.2. Adapted Linear Program to Check Optimality	37
A.3. Linear Program for Popular Random Assignments with Equal Treatment of Equals	37
A.4. First Difference Between the Allocation Algorithms with Equal Number of Objects and Agents	38
A.4.1. Profile 3,3 - 2	38
A.4.2. Profile 3,3 - 3	38
A.4.3. Profile 3,3 - 7	39
A.4.4. Profile 4,4 - 52	39
A.5. List of all Profiles Modulo Anonymity and Neutrality	40
A.5.1. Profiles of size $n = 2$	40
A.5.2. Profiles of size $n = 3$	40
List of Figures	43
List of Tables	44

Contents

Bibliography	45
---------------------	-----------

1. Introduction

There are many situations where a set of indivisible objects needs to be assigned to a group of agents. Common examples are college programmes, social housing or donor organs. What makes these problems tricky is that you can not solve these problems by using the free market, since these objects can not or must not be sold. In this thesis we will look at four different solutions to allocate objects to agents, whilst taking the agents respective preferences into account:

- **Random Serial Dictatorship** puts the agents in a random order and lets them pick their most preferred object out of the remaining objects.
- **Extended Probabilistic Serial** turns the problem into an eating competition in which all agents “eat” their top preference and are rewarded with a chance of winning the object relative to the amount that they have eaten of it.
- **Popular Random Assignments** produce solutions that fulfil the democratic idea that there is no majority that would prefer another solution.
- The naive and adaptive versions of the **Boston Mechanism** use random tie-breaking to iteratively match agents to their objects.

After we have looked at the different strategies, we will analyse the solutions that they provide. We will check if agents with equal preferences are treated equally and if there are agents that envy their competitors. We will also find out what constitutes an “optimal solution” and if all of the strategies are able to provide one.

Then we will think about the idea of preferences and find a way to list all possible preference rankings without unnecessary redundancy. We will also spend some time thinking about randomly generating preferences using different models.

In section 2 we start out by looking at the already existing work. Then we will move on to section 3, where we will formalise our problem and many of the terms we will use throughout this thesis. The solution concepts will be formally introduced and discussed in section 4. In section 5 we will have a look at an example implementation of the algorithms in Java and discuss the design decisions of the program. Afterwards

we will look at the problems of preference enumeration and preference sampling in sections 6 and 7 respectively. Lastly, we will look at possible extensions of this work in section 8.

2. Literature

Random Serial Dictatorship, as an algorithm that picks a random ordering of agents and lets them pick their most preferred object, was first discussed by Allan Gibbard in 1977 [Gib77]. This idea, however, proved to have several drawbacks: Aziz et al. [ABB13] showed that it gets computationally difficult to determine the exact chances of agents and objects being matched as problem size increases.

Bogomolnaia et al. [BM01] looked at several different properties of Random Serial Dictatorship and found examples where the algorithm returns results that are suboptimal in the sense that all agents would prefer another allocation over the one returned.

Bogomolnaia et al. then also introduced a new alternative to Random Serial Dictatorship, called Probabilistic Serial. This algorithm can be thought of as a race between the agents to claim the highest chances of winning their preferred objects for themselves. They found several interesting properties of this mechanism. For example that any agent will always either prefer his “cut” over the ones of other agents or at least be indifferent between the two. However, they also found that the properties of Probabilistic Serial were incompatible with strategy-proofness, meaning that agents could swing the outcome in their favour by lying about their true preferences – an “attack” Random Serial Dictatorship was insusceptible to. Probabilistic Serial had several extensions. Most notably one by Katta and Sethuraman [KS06] for situations in which the agents could be indifferent between objects. This Extended Probabilistic Serial algorithm inherited many of the axiomatic properties including the envy-freeness property described above.

Probabilistic Serial was then again extended by Budish et al. [BCKM13] Their approach focused on different constraints one might want to place on the general allocation problem. For example matching students to schools with a certain capacity or with fixed quotas for boys and girls. Their generalisation also kept most of the axiomatic properties of the base mechanism.

One of the most famous allocation mechanisms is the so-called Boston Mechanism that was described by Abdulkadiroğlu and Sönmez [AS03] while discussing the mechanisms of school choice.

Boston schools adopted this mechanism in 1999 but dropped it after researchers re-

vealed several flaws with the mechanism, including being susceptible to manipulation by parents trying to “game the system” and questionable outcomes like students getting rejected by a school in favour of students who have given the school a lower priority than the students that were rejected [KÜ10].

Another very interesting family of object assignments are the so-called popular assignments that were introduced by Kavitha et al. [KMN11]. They are called popular because these assignments would win or at least tie if the agents had to take a vote between them and any other assignment. Kavitha et al. showed that these assignments would always exist and that they could be computed efficiently using linear programming. They also provided tight bounds of the “price of anarchy” and “price of stability” and demonstrated several other properties of this family of assignments.

Aziz et al. [ABS13] showed that there always exists a popular assignment that treats all agents with equal preferences equally. They also found examples with three agents and objects where all popular assignments fail to be free of envy or immune to manipulation by individual agents.

Brandt et al. [BHS17] analysed both popular random assignments as well as so-called pure popular assignments using majority graphs. They gave a sufficient criterion for the uniqueness of a popular assignment and proofed that there are situations in which no popular assignment is immune to even less extreme attempts of manipulation than the ones considered by Aziz et al. if the number of agents is at least seven. They also found a situation with five agents where an even weaker version of envy-freeness than the one considered by Aziz et al. was violated by all popular assignments.

Brandl et al. [BBH17] also analysed if the outcome of some of the algorithms could be manipulated by a person who chooses to abstain from the assignment process i.e. a person who does not give any preferences. They showed that both Random Serial Dictatorship and Extended Probabilistic Serial can not be manipulated by individuals or groups choosing not to participate. The Boston Mechanism is only secure from individuals and can guarantee that no group can get a better outcome by not participating, whilst popular random assignment rules can only guarantee the latter.

3. Formalisation

Before implementing the different assignment methods, we start by formalising the problem. An assignment problem (often also referred to as “allocation problem” or “one-sided matching problem”) consists mainly of three parts:

1. A set of n agents $A = \{a_1, \dots, a_n\}$.
2. A set of m objects $O = \{o_1, \dots, o_m\}$.
3. A set of preferences $\succ = \{\succ_1, \dots, \succ_n\}$ over the objects.

The task is to assign an object to each agent, whilst respecting the agent’s preferences. An example could be a set of houses in a social housing program, that are to be distributed among the poor, or offices in a company, that have to be assigned to the employees.

Generally it is assumed that the number of agents and the number of objects are equal (i.e. $n = m$). This assumption can be made without loss of generality, since we can always add “dummy objects” or “indifferent agents” to pad out the sets [KS06]. If an agent receives a dummy object by one of the algorithms, that means he will not get an actual object and is left unmatched. This is necessary if the number of agents is greater than the number of objects, like in the case of applicants to a popular school program.

3.0.1. Preferences

As we have said, every agent a_i has preferences \succ_i over the objects. These can be plain expressions of what an agent “likes more” as in the case of housing or school choice, but they can also be more complex as in the case of matching donated organs to the people that need them. In this case, the preferences indicate the chance of a successful surgery given the medical records of donor and receiver. There are a couple of properties all preferences need to fulfil in our model:

1. **Totality:** All objects can be compared with each other. More formally: for all objects $o_i, o_j \in O$ it is $o_i \succ o_j$ or $o_j \succ o_i$.

2. **Transitivity:** If an agent prefers o_i over o_j and o_j over o_k , he should also prefer o_i over o_k , so for all objects $o_i, o_j, o_k \in O$ if we have $o_i \succ o_j$ and $o_j \succ o_k$, then we also have $o_i \succ o_k$.

We distinguish two different kinds of preferences: Strict preferences and weak preferences. A preference ranking is called strict, if all objects can be put into a clear order. So if we compare any two distinct objects, the agent can pick a clear favourite. This property is called **antisymmetry** and its formal definition states that for all objects $o_i, o_j \in O$ if we have $o_i \succ o_j$ and $o_j \succ o_i$, then it must be that $o_i = o_j$.

Weak preferences, however, allow the agents to be indifferent between two objects. We usually denote weak preferences with \succeq similar to $>$ and \geq with natural numbers. If an agent weakly prefers o_i to o_j and o_j to o_i , we say he is **indifferent** between o_i and o_j (written as $o_i \sim o_j$). Weak preferences are always **reflexive**, meaning an agent is indifferent if he is given the “choice” between two copies of the same object ($o_i \sim o_i$).

To write down explicit preferences, we can either write them down with the presented notation or by using curly braces and commas:

If an agent likes o_1 the most, then o_2 and then o_3 , we write this as $o_1 \succ o_2 \succ o_3$ or just o_1, o_2, o_3 .

If he were indifferent between o_2 and o_3 we would write this as $o_1 \succeq o_2 \sim o_3$ or $o_1, \{o_2, o_3\}$.

As mentioned, every agent has preferences over the objects. So for every set of agents there is a set of preferences. We call this set a **preference profile**. We use \mathcal{P} to denote the set of all possible preference profiles.

3.0.2. Assignments

Since our goal is to assign objects to agents, we need to define how these assignments will look like. Any particular assignment can be represented by a matrix $P = (p_{ij})_{(i,j) \in A \times O}$, where p_{ij} is 1 if agent a_i is assigned the object o_j and 0 otherwise. For example if agent a_1 gets object o_3 , agent a_2 gets object o_1 and agent a_3 gets object o_2 , we would write this the following way:

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

So each column represents an object and each row an agent. We call these matrices **deterministic**. These kind of binary matrices are however only a special case if we only either assign an object to an agent or do not assign it, but for many applications we might want to turn assigning an object into a lottery. The reason for this is that in many

cases this is the only possibility to guarantee fairness. For example if three agents all have the same preferences:

$$\begin{aligned} a_1 &: o_1, o_2, o_3 \\ a_2 &: o_1, o_2, o_3 \\ a_3 &: o_1, o_2, o_3 \end{aligned}$$

No matter how we assign the objects, one agent will get a better outcome than the other two. So instead of just assigning the objects, we instead return a lottery over the objects. A lottery $[q_1 : o_1, \dots, q_n : o_n]$ is defined as a n -tuple of objects o_i and probabilities q_i , such that the probabilities are non-negative and add to one. In a lottery q_i represents the chance of winning the object o_i . For our assignment matrix $P = (p_{ij})_{(i,j) \in A \times O}$ the i th row P_i represents a lottery for agent a_i . This means that the matrix entry p_{ij} is the chance of agent a_i to win object o_j .

It is easy to see that giving everyone an equal chance of getting any object is the only solution to our example problem that treats all agents equally. The resulting assignment would look like this:

$$P = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}$$

Since the matrix contains probabilities, it has to be **bistochastic**, i.e., the sum of each row and each column has to add up to one. This means that every object, including possible dummy objects, will be distributed and every agent will be matched with an object, even if this object might be a dummy. We write \mathcal{D} for the set of deterministic assignments and \mathcal{R} for the set of all (possibly random) assignments.

Knowing all this, we can formally define an **assignment rule** as a function that maps a preference profile to an assignment, i.e. $f : \mathcal{P} \rightarrow \mathcal{D}$ or $f : \mathcal{P} \rightarrow \mathcal{R}$ depending on the algorithm.

3.0.3. Stochastic Dominance

In the previous example we have already used an idea of what fairness might be without actually defining it. In order to do this, we need to extend our simple preferences \succ and \succeq to also include random lotteries of objects, so we can say things like “agent a_4 would prefer a 50 : 50 chance of getting either o_1 or o_2 over getting o_3 .”

We call the extension of the preference relation over lotteries **stochastic dominance** (shortened to SD):

We say agent a_i SD-prefers the lottery $X = [x_1 : o_1, \dots, x_n : o_n]$ over $Y = [y_1 : o_1, \dots, y_n : o_n]$ (written $X \succeq_i^{SD} Y$) if and only if

$$\sum_{\substack{o_k \in O \\ o_k \succeq_i o_j}} x_k \geq \sum_{\substack{o_k \in O \\ o_k \succeq_i o_j}} y_k \quad \text{for all } o_j \in O \quad (3.1)$$

Since an assignment is basically a list of lotteries, we can also define stochastic dominance for them: Agent a_i SD-prefers the assignment $X = (x_{ij})$ over assignment $Y = (y_{ij})$ (written $X \succeq_i^{SD} Y$) if and only if

$$\sum_{\substack{o_k \in O \\ o_k \succeq_i o_j}} x_{ik} \geq \sum_{\substack{o_k \in O \\ o_k \succeq_i o_j}} y_{ik} \quad \text{for all } o_j \in O \quad (3.2)$$

Intuitively, this just means that for any object o_j the chance of agent a_i getting an object that is at least as good as o_j is at least as high in X as it is in Y . Note that stochastic dominance only depends on one row of each assignment, so we only need to consider X_i and Y_i when thinking about the stochastic dominance of agent a_i . As an example we look at the following preference relations and assignments:

$$\begin{array}{l} a_1 : o_1, o_2, o_3 \\ a_2 : o_1, o_3, o_2 \\ a_3 : o_2, o_3, o_1 \end{array} \quad X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 0 & 3/4 & 1/4 \end{pmatrix}$$

We now look at agent a_2 . In X he would get object o_3 . In Y he would get object o_3 with a chance of 0.5 and object o_1 with a chance of 0.5. Since he likes object o_1 more than object o_3 , he would prefer Y over X ($Y \succeq_2^{SD} X$ or $Y_2 \succeq_2^{SD} X_2$). Agents a_1 and a_3 would prefer X for similar reasons.

We will also use the idea of strict stochastic dominance ($X \succ_i^{SD} Y$). X strictly dominates Y if we have $X \succeq_i^{SD} Y$ but not $Y \succeq_i^{SD} X$. So from the perspective of agent a_i , X has to be better than Y .

3.1. Axiomatic Properties

3.1.1. Optimality

We say X **SD-dominates** Y if we have $X \succeq_i^{SD} Y$ for all agents a_i and there is at least one agent a_i , such that we have $X \succ_i^{SD} Y$. An assignment is called **SD-optimal** or **SD-efficient** if there is no assignment that SD-dominates it. More intuitively, we can say an assignment is SD-optimal if it is not possible for anyone to get a better outcome without someone else being worse off.

An easy example for optimality can be found if we consider just two agents who like two different objects the most:

$$a_1 : o_1, o_2$$

$$a_2 : o_2, o_1$$

Let us look at the two possible deterministic allocations:

$$X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

It is easy to see that X is optimal while Y is not, since all agents would prefer X over Y .

3.1.2. Equal Treatment of Equals

The most basic idea of fairness is the idea of **equal treatment of equals**. An allocation $X = (x_{ij})$ fulfils this notion if for all agents $a_i, a_j \in A$ with $\succeq_i = \succeq_j$ we also have $X_i = X_j$. Put more simple, an allocation fulfils equal treatment of equals if all agents with equal preferences receive the same chances of getting any object.

We already encountered equal treatment of equals when we considered the preference profile with three agents who like the same objects:

$$a_1 : o_1, o_2, o_3$$

$$a_2 : o_1, o_2, o_3$$

$$a_3 : o_1, o_2, o_3$$

As we have seen, the only allocation that satisfies equal treatment of equals is the allocation that gives everyone equal chances of winning any object:

$$\begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}$$

3.1.3. Envy-Freeness

More strict is the idea of envy-freeness: An allocation is envy-free if every agent likes his allocation as much as any of the allocations of the other agents, i.e. $X_i \succeq_i^{SD} X_j$

for any agent $a_i \neq a_j$. There is also the idea of **weak envy-freeness** that states that no agent should strictly prefer another allocation to his. It can be shown that envy-freeness implies equal treatment of equals, while weak envy-freeness does not [ABS13]. We can see the different types of envy-freeness when considering the following preference profile:

$$a_1 : o_1, o_2, o_3$$

$$a_2 : o_1, o_3, o_2$$

$$a_3 : o_3, o_1, o_2$$

Lets look at three different allocations:

$$X = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad Y = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad Z = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 1/4 & 1/4 \\ 0 & 1/4 & 3/4 \end{pmatrix}$$

X gives two agents their first preference. However, if we consider a_1 , we can see that he would strictly prefer the allocation of a_2 over his own. X therefore violates both notions of envy-freeness.

Y solves this problem, however there can still be found some envy. Consider a_2 and a_3 and remember the definition of stochastic dominance: We say $Y_2 \succeq_2^{SD} Y_3$ if for every object o_i the chance of winning an object at least as good as o_i is at least as high in Y_2 as it is in Y_3 . But now look at object o_3 . The objects at least as good as o_3 are o_1 and o_3 . Agent a_2 has a 0.5 chance of winning o_1 but no chance of winning o_3 . Agent a_3 is guaranteed to win o_3 so his chance is 1. So despite the fact that Y is weak envy-free it still violates envy-freeness.

All violations are solved in Z . We can compare any two agents and will find no violation of envy-freeness. If we rerun the calculation from before, now both a_2 and a_3 have a chance of $3/4$ of winning an object at least as good as o_3 , so there is no longer a violation.

3.1.4. Popularity

Since we will consider popular assignments, we have to define what “popular” actually means. In order to do this, we introduce popularity functions: First we define a function $\phi_i : O \times O \rightarrow \{-1, 0, 1\}$ for every agent a_i . This function is defined as follows:

$$\phi_i(o_j, o_k) = \begin{cases} 1 & \text{if } o_j \succeq_i o_k \\ 0 & \text{if } o_j \sim_i o_k \\ -1 & \text{if } o_k \succeq_i o_j \end{cases}$$

$\phi_i(o_j, o_k)$ can be thought of as a voting function in an election between the objects o_j and o_k . If the agent prefers o_j he will vote for it, if he prefers o_k he will vote against o_j and if he is indifferent he will abstain. Next we extend this idea to entire allocations: $\Phi : \mathcal{R}^2 \rightarrow \mathbb{R}$ compares two allocations X and Y by going through every possible situation in which an agent a_i is matched with the object o_j in allocation X and object o_k in allocation Y . He votes for his favourite outcome. His vote is then weighted with the likelihood of this situation happening and summed for all possible situations and agents.

$$\Phi(X, Y) = \sum_{a_i \in A} \sum_{o_j, o_k \in O} x_{ij} y_{ik} \phi_i(o_j, o_k)$$

If $\Phi(X, Y)$ is positive, this means that in the majority of situations more agents prefer their allocation in X over Y than vice-versa, so X is more popular than Y and the other way round if $\Phi(X, Y)$ is negative. An allocation X is then called **popular** if there is no allocation Y more popular than X .

As an example consider the profile from before:

$$a_1 : o_1, o_2, o_3$$

$$a_2 : o_1, o_3, o_2$$

$$a_3 : o_3, o_1, o_2$$

We also had the two allocations X and Z :

$$X = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad Z = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 1/4 & 1/4 \\ 0 & 1/4 & 3/4 \end{pmatrix}$$

Z was satisfying (weak) envy-freeness and X was not. But Z is not a popular assignment. Both a_2 and a_3 would be better off in X , so if there was an election a_1 would be in the minority and would lose.

4. Algorithms

Now that we know the problem domain, we can take a look at different approaches of solving allocation problems and their axiomatic properties.

4.1. Random Serial Dictatorship

Random Serial Dictatorship (RSD) is an extension of a regular serial dictatorship method:

In a serial dictatorship we have a fixed order of agents. Then we let each agent pick their preferred object from the set of objects that are still available. So for example if we have the preferences

$$a_1 : o_1, o_2, o_3$$

$$a_2 : o_1, o_3, o_2$$

$$a_3 : o_2, o_3, o_1$$

and the order (a_1, a_2, a_3) , agent a_1 would pick o_1 , agent a_2 would pick o_3 since o_1 is not available anymore and agent a_3 would pick o_2 . Leading to the following allocation:

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

In RSD we randomise over all possible serial dictatorships, so that the resulting (random) allocation is just the weighted (component wise) sum of all possible serial dictatorships:

$$P = \begin{pmatrix} 1/2 & 1/6 & 1/3 \\ 1/2 & 0 & 1/2 \\ 0 & 5/6 & 1/6 \end{pmatrix}$$

More formally: Let $\mathcal{S}(\succeq)$ be the set of all possible serial dictatorship allocations for a given preference profile \succeq . The RSD allocation is then simply defined as

$$\mathcal{RSD}(\succeq) = \frac{1}{|\mathcal{S}(\succeq)|} \sum_{s \in \mathcal{S}(\succeq)} s$$

Properties

RSD is quite intuitive. It fulfils the notions of equal treatment of equals and weak envy-freeness [BM01]. However, there are a couple of problems with using RSD. Firstly, the calculation of an RSD allocation is a #P Complete problem [ABB13], meaning it is not feasible for big numbers of agents and objects. Additionally RSD violates efficiency, as can be seen with this example [BM01]:

$$a_1 : o_1, o_2, o_3, o_4$$

$$a_2 : o_1, o_2, o_3, o_4$$

$$a_3 : o_2, o_1, o_4, o_3$$

$$a_4 : o_2, o_1, o_4, o_3$$

RSD results in this allocation:

$$\begin{pmatrix} 5/12 & 1/12 & 5/12 & 1/12 \\ 5/12 & 1/12 & 5/12 & 1/12 \\ 1/12 & 5/12 & 1/12 & 5/12 \\ 1/12 & 5/12 & 1/12 & 5/12 \end{pmatrix}$$

However, it is dominated by the following allocation:

$$\begin{pmatrix} 1/2 & 0 & 1/2 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \end{pmatrix}$$

It was also shown by Mihai Manea [Man09] that the probability of a preference profile being SD-efficient converges to zero as the problem size increases.

4.2. (Extended) Probabilistic Serial

Extended Probabilistic Serial (EPS) is an extension of Probabilistic Serial (PS). PS models the allocation problem as an “eating competition” where every agent has the same eating speed. Every agent starts consuming their currently most preferred object that has not yet been fully consumed. If an object is fully eaten up, the agents move on to their next preferences until all objects are consumed. Each agent is then awarded with a chance of getting the object equal to the amount of the object he managed to consume. EPS extends this idea to also work with non-strict preferences by reducing the problem

to sequence maximum flow problems on weighted graphs.

Every agent usually has a constant eating speed. Each eating speed is described by a measurable eating speed function $\omega_i : [0, 1] \rightarrow \mathbb{R}^+$. This function gives the eating speed at any given time t . An agent consumes an entire object worth in one time step such that one step is sufficient to finish the entire process for all agents. This means for the eating speed function that $\int_0^1 \omega_i(t)dt = 1$. As an example consider again the profile

$$a_1 : o_1, o_2, o_3$$

$$a_2 : o_1, o_3, o_2$$

$$a_3 : o_2, o_3, o_1$$

Both agent a_1 and a_2 start consuming object o_1 , while agent a_3 consumes o_3 . If we assume a constant eating speed, object o_1 will be consumed at time $t = 0.5$. The incomplete allocation up to this point looks like this:

$$P_{0.5} = \begin{pmatrix} 1/2 & 0 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{pmatrix}$$

Now agent a_1 and a_3 both consume the remains of o_2 while a_2 starts eating o_3 . At $t = 0.75$, o_2 will also be consumed and the allocation looks like this

$$P_{0.75} = \begin{pmatrix} 1/2 & 1/4 & 0 \\ 1/2 & 0 & 1/4 \\ 0 & 3/4 & 0 \end{pmatrix}$$

Finally all agents will turn to the remains of o_3 and consume it, so the final allocation at $t = 1$ is

$$P = \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 0 & 3/4 & 1/4 \end{pmatrix}$$

Properties

EPS satisfies all properties we are considering except popularity and can also be computed in polynomial time by solving a series of parametric-maxflow problems [KS06].

For an example for how EPS violates popularity we can consider the following profile:

$$a_1 : o_1, o_2, o_3$$

$$a_2 : o_1, o_2, o_3$$

$$a_3 : o_2, o_3, o_1$$

EPS returns the following matrix:

$$\begin{pmatrix} 1/2 & 1/6 & 1/3 \\ 1/2 & 1/6 & 1/3 \\ 0 & 2/3 & 1/3 \end{pmatrix}$$

It can be easily seen, that both a_1 and a_2 would prefer the following:

$$\begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4.3. Popular Assignment Rules

Popular Assignment Rules are, as the name suggests, a whole family of rules that all satisfy the notion of popularity as we have already defined it in section 3. Formally, Popular Assignment Rules are all rules \mathcal{POP} for which all allocations $\mathcal{POP}(\succeq)$ are popular, i.e. have $\Phi(\mathcal{POP}(\succeq), Y) \geq 0$ for all allocations Y . We call allocations with this property **Popular Random Allocations** (PRA).

In this thesis we will mainly consider an assignment rule that always satisfies both popularity and equal treatment of equals. In this case the “rule” is just “return the allocation given by the solution of the LP found in the appendix”. We will refer to the result of this rule as PRA but note that this might not be the unique popular assignment but just *one* popular assignment. This is important to keep in mind when considering the data sheets in the appendix.

Properties

Individual popular assignments can be computed using linear programs, making them quite performant [KMN11]. It was also shown that popular assignments always satisfy efficiency and that there always exists at least one popular assignment that also satisfies equal treatment of equals and which can also be computed efficiently (namely the one we will be using) [ABS13]. However, it can be shown that all popular assignments violate envy-freeness in certain situations with three or more agents and weak envy-freeness for five or more agents [BHS17].

4.4. Boston Mechanism

The Boston Mechanism (BM) uses a turn based approach to allocation. There are two versions: The Naive Boston Mechanism (NBM) and the Adaptive Boston Mechanism (ABM).

4.4.1. Naive Boston Mechanism

The Naive Boston Mechanism works as follows: Every turn n every agent checks if his n th preference is still available. If it is, the agent tries to allocate it. If multiple agents try to go for the same object, the tie is broken randomly. If an object is already allocated, the agent will skip his turn.

4.4.2. Adaptive Boston Mechanism

The Adaptive Boston mechanism works in a similar way. The only difference is that instead of skipping his turn, every agent just picks the most preferred object that is still available. ABM looks at first glance very similar to RSD, however where in RSD every possible order of agents is considered, ABM considers just one and uses tie-breaking instead of summing over all possible sequences of agents.

The difference can be seen in the following example:

$$a_1 : o_1, o_2, o_3, o_4$$

$$a_2 : o_1, o_2, o_3, o_4$$

$$a_3 : o_1, o_3, o_2, o_4$$

$$a_4 : o_2, o_1, o_3, o_4$$

Since a_4 is the only agent to pick o_2 as his first preference, he is guaranteed to win o_2 in the first round, so we only need to consider the remaining agents. In the first turn agents a_1 , a_2 and a_3 all compete for o_1 . Lets consider the situation in which a_1 wins. Since o_2 is already taken, a_2 will skip his turn with NBM instead of competing for his third preference like in ABM. A possible order of events in both algorithms could look like this:

Now we consider the case where a_3 wins the tiebreaker. In NBM both a_1 and a_2 skip their turn despite the fact that they are the only agents left:

Note that a_3 is guaranteed to win either o_1 or o_3 in NBM instead of having to compete

	Turn 1	Turn 2	Turn 3		Turn 1	Turn 2	Turn 3
a_1	o_1			a_1	o_1		
a_2	o_1	Skip	o_4	a_2	o_1	o_3	
a_3	o_1	o_3		a_3	o_1	o_3	o_4
a_4	o_2			a_4	o_2		

Table 4.1.: NBM vs. ABM if a_1 wins the first tie break.

	Turn 1	Turn 2	Turn 3	Turn 4		Turn 1	Turn 2	Turn 3
a_1	o_1	Skip	o_3		a_1	o_1	o_3	
a_2	o_1	Skip	o_3	o_4	a_2	o_1	o_3	o_4
a_3	o_1				a_3	o_1		
a_4	o_2				a_4	o_2		

Table 4.2.: NBM vs. ABM if a_3 wins the first tie break.

with a_1 and a_2 , so the allocation in NBM looks like this:

$$\begin{pmatrix} 1/3 & 0 & 1/6 & 1/2 \\ 1/3 & 0 & 1/6 & 1/2 \\ 1/3 & 0 & 2/3 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

In ABM the three agents compete with each other resulting in this allocation:

$$\begin{pmatrix} 1/3 & 0 & 1/3 & 1/3 \\ 1/3 & 0 & 1/3 & 1/3 \\ 1/3 & 0 & 1/3 & 1/3 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

In this case NBM violates weak envy-freeness while ABM does not.

Properties

The Boston Mechanisms do not differ a lot when just considering weak envy-freeness and efficiency. Both mechanisms can violate these properties, as can be easily seen in

the following examples:

$$a_1 : o_1, o_2, o_3$$

$$a_2 : o_1, o_2, o_3$$

$$a_3 : o_2, o_1, o_3$$

The resulting allocation in both cases is:

$$\begin{pmatrix} 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \end{pmatrix}$$

We can see that all agents would envy a_2 , since he gets o_2 , while they only get $1/2$ of any object at least as good as o_2 . We have also seen that NBM can violate weak envy-freeness.

Now let us consider efficiency and the following profile:

$$a_1 : o_1, o_2, o_3, o_4$$

$$a_2 : o_1, o_2, o_3, o_4$$

$$a_3 : o_1, o_2, o_4, o_3$$

$$a_4 : o_1, o_2, o_4, o_3$$

The resulting allocation in both cases is:

$$\begin{pmatrix} 1/4 & 1/4 & 5/12 & 1/12 \\ 1/4 & 1/4 & 5/12 & 1/12 \\ 1/4 & 1/4 & 1/12 & 5/12 \\ 1/4 & 1/4 & 1/12 & 5/12 \end{pmatrix}$$

Which is dominated by

$$\begin{pmatrix} 1/4 & 1/4 & 1/2 & 0 \\ 1/4 & 1/4 & 1/2 & 0 \\ 1/4 & 1/4 & 0 & 1/2 \\ 1/4 & 1/4 & 0 & 1/2 \end{pmatrix}$$

Lastly, it can be shown that the Boston Mechanisms fulfil equal treatment of equals [KÜ15].

	Efficiency	Equal Treatment of Equals	Envy- Freeness	Weak Envy- Freeness	Performance
RSD	×	✓	×	✓	×
EPS	✓	✓	✓	✓	✓
PRA	✓	✓	×	×	✓
NBM	×	✓	×	×	
ABM	×	✓	×		

Table 4.3.: The axiomatic properties of the discussed algorithms

4.5. Axiomatic Properties

Before we move on; another overview over the properties discussed in this thesis:

5. Software Design

In this chapter we will discuss parts of the implementation of the “Preference Allocation Wizard” (PAW):

5.1. Preferences and Profiles

Every preference relation is an ordered collection of `PreferenceGroups`. That is, the agent modelled with this relation is indifferent between all elements within one of those groups. The methods of the class `PreferenceRelation` enforce consistency within the relation, so adding, moving or removing objects will not leave an invalid relation.

The preference relations are then collected into a `Scenario`. A scenario consists of the names of the objects and agents and their preferences. Again we provide methods for modifying preferences and names that enforce consistency within the scenario. Such, that all agents have a preference relation of equal size. In order to communicate with the user interface, we use a listener pattern. `ScenarioListeners` can be added to the scenario and will be informed of changes, by sending them `ScenarioUpdateEvents`.

5.1.1. Helper Classes

In order to generate and edit scenarios and preference profiles, several helper methods are provided within the `PreferenceRelationCreator` and `PreferenceRelationTransformer` classes. These methods include the sampling methods discussed in chapter 7 and methods for cloning and rotating relations. The method `createStrictPreference(PreferenceRelation rel)` can be used to create a copy of a given relation, where all indifferences are replaced by strict preferences.

The class `PreferenceTypeIdentifier` can be used to identify **strict**, **weak**, **indifferent** (only one preference group) and **dichotomous** (only two preference groups) preference relations, which is used to check whether a certain algorithm can be used on a specific scenario.

5.1.2. Scenario Files

The class `ScenarioIO` is used to save, load, and parse scenarios. A file created using the methods provided in `ScenarioIO` has the following structure:

1. A comma-separated array of the names of agents (within angle brackets). If a name is not renamed, it will store an empty string.
2. A comma-separated array of object names, formatted the same way.
3. The preference relations of every agent, written in “comma and curly braces” notation.

5.2. Allocations

The class `Allocation` is in essence a wrapper for a two-dimensional double array with a method for pretty printing but without any constraint checks.

5.2.1. Allocation Files

Similar to scenarios, allocations have dedicated export methods, this time provided by `AllocationIO`. There are two `AllocationExportOptions`:

- `CSV_ROUND` exports the matrix as a “.csv” file. Values are rounded to six significant figures.
- `CSV_FRACTION` exports the matrix as a “.csv” file. Values are approximated as fractions.

All relevant methods for exporting, converting and approximating can be found in the classes `AllocationExportOption`, `AllocationConverter`, `AllocationIO`, and `Helper-Functions`.

5.3. Assignment Strategies

In order to implement the different assignment strategies, we make use of the “strategy” pattern:

- `AllocationStrategyBase` is the base interface for all allocation strategies. It contains methods for checking if a given scenario can be used as input for the strategy.

- `AllocationStrategy` is the first extension of the base interface. It has a simple `allocate(Scenario scenario)` method.
- `AllocationStrategyFamily` is the second extension of the base interface. It adds the method `getAllAllocations(Scenario scenario)` and can be used for allocation strategies that return multiple allocations, as it is the case with the convex set of popular assignments.

5.3.1. Implementations

We have already discussed the theoretical part of different allocation algorithms. In this section we will have a brief look at the available implementations of these algorithms:

RSD

The RSD method implemented in the `RandomSerialDictatorship` class is a simple recursive method that sums all possible serial dictatorships and then normalises the allocation. It only works on strict preferences. There is also a check to only allow allocations on scenarios with less than 11 agents, since the algorithm is very slow.

PS

The PS algorithm is distributed over several classes: `ExtendedProbabilisticSerial` is where the algorithm gets called, but the actual calculations are done in the `ExtendedProbabilisticSerialAlgorithm` class and the `allocation.algorithms.parametric-MaximumFlow` package. We use the algorithm of Ketta et al. [KS06] and the basic version of the parametric max-flow algorithm of Gallo et al. [GGT89]. There are more optimised ways to solve parametric max-flow problems, but they are unnecessary given the problem sizes. There is also an unused implementation of the Dichotomous Probabilistic Serial algorithm described in the same paper by Ketta et al. [KS06]. However, this will just return the same allocation as regular EPS.

PRA

Since popular random assignments are not necessarily unique, we just use one basic example of an assignment rule with equal treatment of equals. The allocation is done by solving a linear program. The used LP can be found in the appendix. It is a modified version of the original LP provided by Kavitha et al. [KMN11]

Popular Convex Set

The `PopularConvexSet` class and `allocation.algorithms.convexLPSet` package are used to calculate all corners of the convex polytope defined by the linear program, which defines the set of popular random assignments. It uses the method described in the thesis of Martin Suderland [Sud15] that we have translated from the original Matlab code to Java. This method only works for small scenarios with an equal number of agents and objects.

BM

The Boston Mechanisms are also split over several classes: `NaiveBostonMechanism` and `AdaptiveBostonMechanism` both call the `allocate(Scenario scenario, boolean adaptive)` method provided by the `BostonMechanism` class but using different parameters. This can be done because the implementation of NBM and ABM only differ in minor details. Both enforce equal numbers of agents and objects and strict preferences.

5.4. Testing Axiomatic Properties

We currently are able to check three axiomatic properties of the allocations: Efficiency, envy-freeness, and weak envy-freeness.

Efficiency is checked using a linear feasibility test that was adapted from Brandt et al. [BBG16] The LP can be found in the appendix.

The envy tests are done in the `StochasticDominance` class that also checks the strict and weak variations of stochastic dominance. Both checks return lists of `ViolationPair` containing the information of the agents violating the envy-freeness property.

5.5. The GUI

The user interface is split over several windows:

5.5.1. The Main Frame

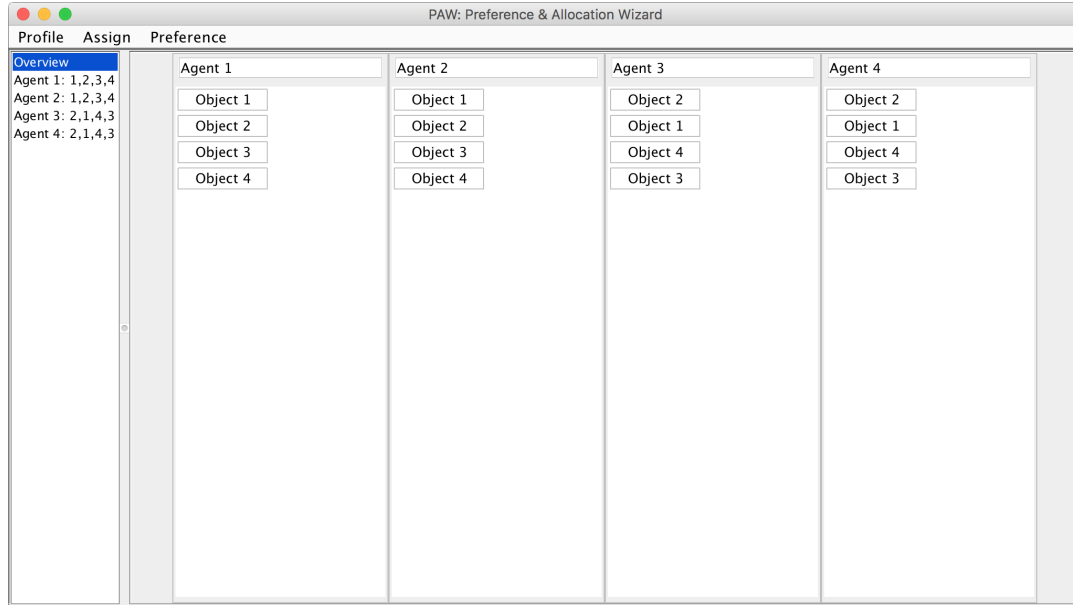
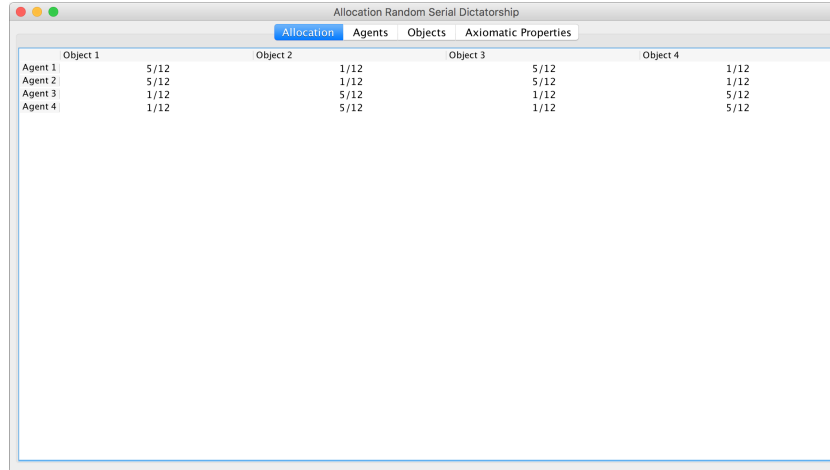


Figure 5.1.: The main screen of the GUI.

The `MainFrame` class defines the main window of the tool. It consists of the main menus for preference creation and modification and several panels for direct modification using drag and drop. These UI elements are mostly defined in the `gui` and `gui.preferenceTree` packages. If an allocation algorithm is called via the *Assign* menu, it opens a new window. This is either a static window or a live window. Both are defined in the `gui.allocation` package. Their main difference is that static windows will not change on edits in the main frame and are more performant with bigger scenarios. Both have the same four possible views:

5.5.2. The Allocation Main Frame



	Object 1	Object 2	Object 3	Object 4
Agent 1	5/12	1/12	5/12	1/12
Agent 2	5/12	1/12	5/12	1/12
Agent 3	1/12	5/12	1/12	5/12
Agent 4	1/12	5/12	1/12	5/12

Figure 5.2.: The allocation screen.

The allocation view just displays the allocation matrix provided by the given allocation rule.

Agents

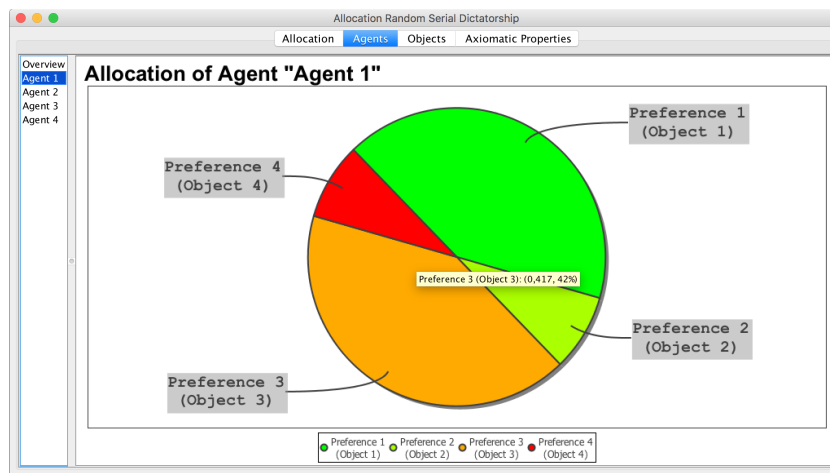


Figure 5.3.: Chart of the allocation of agent a_1 .

The agent view shows how much of their first, second, third etc. preference each agent received.

Objects

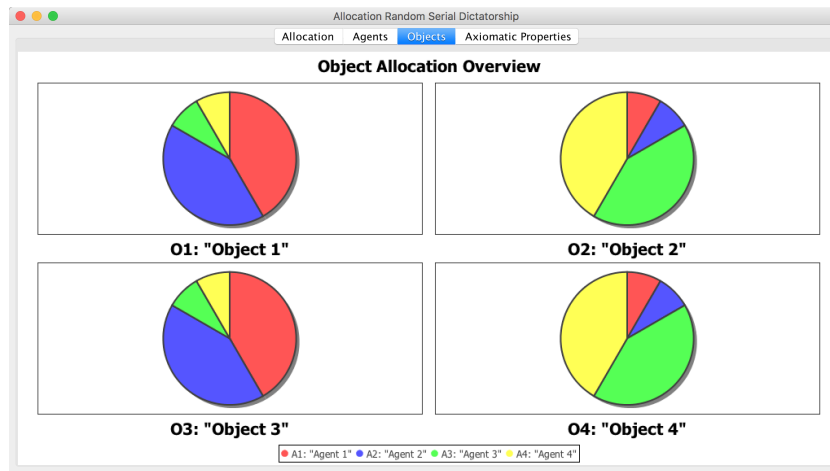


Figure 5.4.: Chart of the allocation of the different objects.

The object view shows how each of the objects was distributed among the agents.

Axiomatic Properties

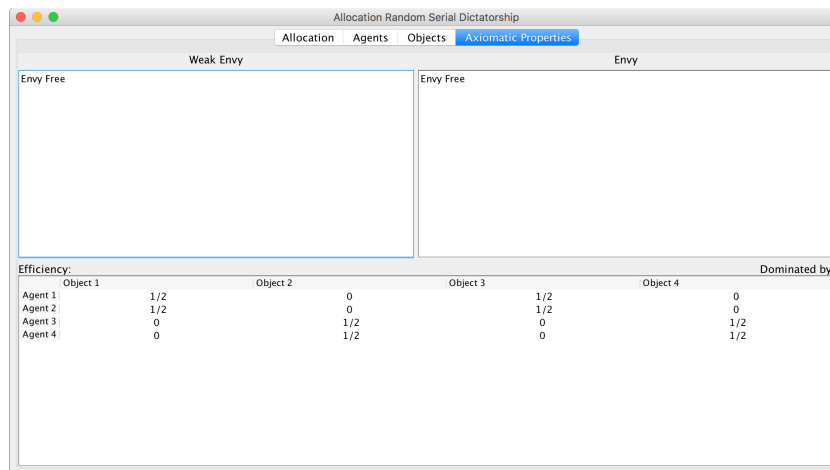


Figure 5.5.: Overview over the axiomatic properties.

The axiomatic properties view lists all violations of (weak) envy-freeness and also displays a dominating allocation if the allocation is not optimal.

5.6. Libraries

In our implementation we use the **SCPSolver** LP Solver library (<http://scpsolver.org/>).

We also use the **EJML** library for our work with matrices in the `allocation.algorithms.convexLPSet` package (http://ejml.org/wiki/index.php?title=Main_Page).

Lastly we use JFreeChart to generate pie charts as a visual aid (<http://www.jfree.org/jfreechart/>).

6. Profile Enumeration

Our next goal was to enumerate all possible strict preference profiles up to a certain amount of agents and objects. The intuitive approach would just be to enumerate all possible preference relations over the objects and then let every agent pick a relation thereby constructing all possible combinations of those relations. We can construct a simple algorithm to accomplish this:

Figure 6.1. A naive algorithm for preference profile enumeration

```
1: function ENUMERATE PROFILES( $n$ )
2:    $rel \leftarrow EnumerateRelations(n)$ 
3:    $out \leftarrow []$ 
4:   for all  $x_1, \dots, x_n \in Relation$  do                                 $\triangleright$  Generate all combinations.
5:      $out.add([x_1, \dots, x_n])$ 
6:   end for
7:   return  $out$ 
8: end function
9:
10: function ENUMERATE RELATIONS( $n$ )
11:   if  $n = 1$  then
12:     return  $[[1]]$ 
13:   end if
14:    $out \leftarrow []$ 
15:    $recRel \leftarrow EnumerateRelations(n - 1)$      $\triangleright$  Generate relations with  $n - 1$  objects.
16:   for all  $rel:recRel$  do
17:     for  $i \leftarrow 0, \dots, n - 1$  do                                 $\triangleright$  For every possible position ...
18:        $nRel \leftarrow insert(rel, n, i)$      $\triangleright$  ... generate a new relation by inserting  $n$ .
19:        $out.append(nRel)$ 
20:     end for
21:   end for
22:   return  $out$ 
23: end function
```

6.0.1. Anonymity and Neutrality

This is an easy algorithm to implement, but it has a couple of problems: The most obvious is its runtime. A quick calculation reveals that enumerating all possible preference relations has a runtime of $O(n!)$. Enumerating all profiles then has a runtime of $O(n^n)$.

The second problem is that many of these produced profiles are not relevant for our uses: Our algorithm produces for $n = 2$ both

$$a_1 : o_1, o_2$$

$$a_2 : o_2, o_1$$

and

$$a_1 : o_2, o_1$$

$$a_2 : o_1, o_2$$

However, we can see that these two profiles will not get very different results when plugging into one of our algorithms, because we can just swap the agents without getting a wildly different result, apart from the rows of the resulting allocation being swapped. We say our algorithms are **anonymous**, so they do not change their results when the order of the agents is changed. An example for an algorithm that violates this property would be serial dictatorship, where the order of agents is predetermined. There is also another property that works in a similar way. Instead of swapping the agents, we also could have renamed the objects. If we renamed o_1 to o_2 and vice versa, we could have produced the same profile. Our algorithms also do not care about the order in which the objects are put. If we swap the name of two objects, we get the same result, but this time with two swapped columns. We call this property **neutrality**. Ideally, we only want to produce new profiles that can not be turned into an already existing profile by swapping agents or renaming objects.

In order to create a new algorithm, we need to define some additional terms:

We call the preference relation o_1, o_2, \dots, o_n (i.e. the ascending ordering of n objects) the **identity relation** (written Id_n or just Id). Since we can always rename our objects, we can enforce the criterion that the first agent should always have this identity relation as his preference.

Additionally, we will need an arbitrary, antisymmetric ordering \geq over the set of preference relations, that has Id_n as its minimum. For the implementation plain lexicographic ordering was used. Since we can swap our agents in any way we want,

we can enforce that for the preference relations p of any two agents a_i and a_{i+1} , we have $p_i < p_{i+1}$. Since $p_0 = Id_n$ is the minimum, this won't interfere with our first assumption.

6.0.2. Preference-transformations and Preference-duality

We then define a **renaming** as a bijective function (i. e. a permutation) that maps the set of objects to itself. For a renaming π and a (strict) preference p we write $\pi(p)$ for the preference relation we would get by renaming the objects in it using π . Now we can associate every preference relation with the renaming that applied to the identity yields our preference. We call this the **preference-dual** (p^D). We have $p^D(Id) := p$. Since all renamings are bijective, we can also invert them and get $p^{-D}(p) = Id$. We call these **inverse duals**.

As an example let our p be the following relation: $p = o_3, o_1, o_2$. We then get

$$p^D = \begin{cases} p^D(o_1) = o_3 \\ p^D(o_2) = o_1 \\ p^D(o_3) = o_2 \end{cases}$$

And

$$p^{-D} = \begin{cases} p^{-D}(o_1) = o_2 \\ p^{-D}(o_2) = o_3 \\ p^{-D}(o_3) = o_1 \end{cases}$$

Notice that the dual just renames object o_i to the object on the i th position in the preference and vice versa for the inverse dual. This also means that $Id^D(p) = Id^{-D}(p) = p$ for all p .

We can also apply these renamings to entire preference profiles. So if we have a renaming π and a profile $P = (p_1, \dots, p_n)$, we can just write $\pi(P) = (\pi(p_1), \dots, \pi(p_n))$. However, we add the caveat that if a renamed profile violates our ordering constraint, it will be brought back into order after the renaming. With all these definitions out of the way, we can show the following:

Theorem 1. *Two ordered preference profiles $P = (p_1, \dots, p_n)$ and $Q = (q_1, \dots, q_n)$ with $p_1 = q_1 = Id$ are equivalent modulo anonymity and neutrality if and only if there is a $p_i \in P$ such that $p_i^D(Q) = P$.*

Proof. The full proof can be found in the appendix. □

We can then see that our theorem is equivalent to saying P and Q are equivalent if and only if there is a $p_i \in P$ such that $p_i^{-D}(P) = Q$.

This fact allows us to construct a new algorithm that only returns one relation from every equivalence class:

6.1. The Extended Profile Enumeration Algorithm

Figure 6.2. The extended algorithm for enumerating all preference profile modulo anonymity and neutrality

```

1: function EXTENDED ENUMERATE( $n$ )
2:    $\text{prof} \leftarrow \text{EnumerateSortedProfiles}(n)$ 
3:    $\text{out} \leftarrow []$ 
4:   for all  $P \in \text{prof}$  do
5:      $\text{class} \leftarrow [P]$ 
6:     for all  $p_i \in P$  do
7:        $\text{class.add}(p_i^{-D}(P))$ 
8:     end for
9:      $\text{out.add}(P)$ 
10:     $\text{prof.removeAll}(\text{class})$ 
11:  end for
12:  return out
13: end function
14:
15: function ENUMERATE SORTED PROFILES( $n$ )
16:   $\text{rel} \leftarrow \text{EnumerateRelations}(n)$ 
17:   $\text{out} \leftarrow []$ 
18:  for all  $x_2 \leq x_3 \leq \dots \leq x_n \in \text{rel}$  do ▷ Generate all combinations.
19:     $\text{out.add}([Id_n, x_2, \dots, x_n])$ 
20:  end for
21:  return out
22: end function

```

Intuitively, we just produce the list of all profiles and then for every profile P produce all equivalent profiles by applying all inverse duals p_i^{-D} of the preference relations $p_i \in P$ to P . Then discard all but one of these profiles from the list until we have either kept or discarded every profile.

This algorithm, while slow, is quite able to produce all profiles for $n \leq 5$ in an acceptable time on a modern computer. The resulting profiles and resulting allocations for

$n = 2$ and $n = 3$ can be found in the appendix. The profiles for $n \leq 5$ can be found on the appended DVD together with the Preference Allocation Wizard and a command line tool that implements this profile enumeration algorithm and several small helper functions used to generate the datasets provided in the appendix.

The profiles are named using the following scheme:

“Number of agents, Number of objects - Profile ID”

The Profile ID is just an incrementing number.

Our assumptions that the profiles are ordered and start with the identity help pruning a sizeable chunk of the search space. For $n > 5$ the computation time gets a lot longer, since we then have to consider over two million equivalence classes [Öme09].

7. Profile Sampling

There are several possible methods to randomly generate preference profiles. **Impartial Culture** and the **Spatial Model** are commonly used sampling methods. We also extend the spatial model to produce weak preference profiles using greedy distance clustering and introduce a new method called **Iterative Joining**.

7.1. Impartial Culture

Impartial culture (IC) is the easiest method of generating preference profiles. It gives equal probability to all preference profiles. There are several variations of IC that only produce profiles of a given set, like only strict preference profiles or only profiles, such that no two possible profiles are equivalent modulo anonymity and neutrality.

7.2. Spatial Model

The spatial model is an attempt to add more “realism” to the sampling process. Instead of just randomising over all profiles, the sampling is done in an abstract n -dimensional space, most commonly the unit square, cube, or tesseract. All agents and objects are randomly placed within this space. The profile is then just reflecting the distance between the objects and the respective agent. This means that objects that get placed closer together will also be “closer together” within the agents’ preferences.

7.2.1. Greedy distance clustering

Since the standard Spatial Model only produces strict preference profiles, we have extended the regular model to allow for weak preferences. In order to accomplish this, we have introduced a **margin of indifference** in which an agent is indifferent between two objects. These margins can be thought of as concentric rings with the agent as its center point. A ring is drawn for every object, such that the distance between the object and the agent is the average radius of the ring.

For every ring we check the number of objects on it and remove the biggest cluster as a new group. We keep doing this until no objects are left. Then we can order them like before, by using the average distance between the objects in the cluster and the agent.

7.3. Iterative Joining

The last algorithm is an attempt of parametrising Impartial Culture to create weak preference profiles. For a given parameter $p \in [0, 1]$, we start out by creating a strict random preference relation using impartial culture. Now we start iterating over the preference, starting with the second object. With a probability p we will add the object to the indifference class above it. With a probability $1 - p$ it will create its own indifference class. We continue doing this until all object have either created new classes or joined an existing one. We repeat this process for every agent.

This process is equivalent to strict Impartial Culture for $p = 0$. For $p > 0$ it might create weak preference profiles that are, however, no longer created impartially.

8. Outlook

There are several possible applications both for the tool and the enumerated profiles: It should be easy to extend the tool for new allocation mechanisms. For example Bogomolnaia's and Moulin's [BM04] mechanism to allocate under dichotomous preferences. Additionally, one might want to add additional axiomatic properties like strategyproofness or participation to test how safe the strategies are from being manipulated by the agents.

If one were to add a method for batch processing profiles, one could use the already implemented axiomatic tests and mechanisms to check current hypotheses. An example here would be that there is no popular mechanism that satisfies SD-strategyproofness for $n \geq 7$. One could easily check if this bound given by Brandt et al. [BHS17] is actually tight or if a smaller violation of strategyproofness can be found.

One could also use the profile sampling to analyse the frequencies of certain events, like how often RSD violates optimality or how often popular assignments violate strategyproofness or envy-freeness and how different sampling methods affect these frequencies.

One could also add new visualisations for allocations, axioms and preferences. An example for this would be graphs to visualise envy between different agents.

Lastly, it might be handy to add more export functions, for example export allocations, so that they can be directly imported into \LaTeX environments and therefore easily added as examples for papers or theses.

A. Appendix

A.1. Proofs

Theorem 1. *Two ordered preference profiles $P = (p_1, \dots, p_n)$ and $Q = (q_1, \dots, q_n)$ with $p_1 = q_1 = Id$ are equivalent modulo anonymity and neutrality if and only if there is a $p_i \in P$ such that $p_i^D(Q) = P$.*

Proof. Since this theorem is trivial if $P = Q$, we assume that $P \neq Q$.

Let P and Q be equivalent.

Since they are both ordered under the same antisymmetric ordering, they must be equivalent under neutrality. This means there is a renaming π , that if it were applied to Q would yield P . Since renamings can be reversed, there is also a renaming τ that yields Q when applied to P with $\pi = \tau^{-1}$. We know that τ must map one relation p_i in P to the identity. So knowing this, we know that $\tau = p_i^{-D}$ and therefore $\pi = p_i^D$.

Now let there be a $p_i \in P$, such that $p_i^D(Q) = P$.

Since we are allowed to rename the objects and p_i^D is a valid renaming of objects, P and Q must be equivalent modulo anonymity and neutrality. \square

A.2. Adapted Linear Program to Check Optimality

Let p be an allocation. p is optimal, if the following LP yields zero. Adapted from [BBG16].

$$\begin{aligned}
 & \text{minimise} && \sum_{a_i \in A} \sum_{o_j \in O} r_{i,j} \\
 & \text{subject to} && \sum_{o_k \succeq_i o_j} (q_{i,k} - r_{i,j}) = \sum_{o_k \succeq_i o_j} p_{i,j} \quad \forall o_j \in O, a_i \in A \\
 & && \sum_{a_i \in A} (q_{i,j}) = 1 \quad \forall o_j \in O \\
 & && \sum_{o_j \in O} (q_{i,j}) = 1 \quad \forall a_i \in A \\
 & && r_{i,j} \geq 0 \quad \forall o_j \in O, a_i \in A
 \end{aligned}$$

A.3. Linear Program for Popular Random Assignments with Equal Treatment of Equals

Extended version of the LP by Kavitha et al. [KMN11]. It returns a popular allocation \vec{x} that satisfies equal treatment of equals.

$$\begin{aligned}
 & \text{minimise} && \sum_{a_i \in A} \vec{\alpha}_i + \sum_{o_j \in O} \vec{\beta}_j \\
 & \text{subject to} && \vec{\alpha}_i + \vec{\beta}_j \geq \sum_{o_k \in O} \vec{x}_{i,k} \phi_i(o_j, o_k) \quad \forall o_j \in O, a_i \in A \\
 & && \sum_{a_i \in A} (\vec{x}_{i,j}) = 1 \quad \forall o_j \in O \\
 & && \sum_{o_j \in O} (\vec{x}_{i,j}) \leq 1 \quad \forall a_i \in A \\
 & && \vec{x}_{i,j} = \vec{x}_{l,j} \quad \forall o_j \in O, a_i, a_l \in A \text{ with } \succeq_i = \succeq_l \\
 & && \vec{x} \in \mathbb{R}_+^{A \times O} \\
 & && \vec{\alpha} \in \mathbb{R}^A \\
 & && \vec{\beta} \in \mathbb{R}_+^O
 \end{aligned}$$

A.4. First Difference Between the Allocation Algorithms with Equal Number of Objects and Agents

Note that PRA might not be unique. We used the solution of the LP provided above.

	RSD	PS	PRA	NBM	ABM
RSD		3,3 - 7	3,3 - 2	3,3 - 3	3,3 - 3
PS	3,3 - 7		3,3 - 2	3,3 - 3	3,3 - 3
PRA	3,3 - 2	3,3 - 2		3,3 - 2	3,3 - 2
NBM	3,3 - 3	3,3 - 3	3,3 - 2		4,4 - 52
ABM	3,3 - 3	3,3 - 3	3,3 - 2	4,4 - 52	

Table A.1.: Overview over the simplest profile that produces different results with the given algorithms.

A.4.1. Profile 3,3 - 2

	NBM, ABM, RSD & PS	PRA
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/3 & 1/2 & 1/6 \\ 1/3 & 1/2 & 1/6 \\ 1/3 & 0 & 2/3 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
$a_2 : o_1, o_2, o_3$		
$a_3 : o_1, o_3, o_2$		

A.4.2. Profile 3,3 - 3

	NBM & ABM	RSD & PS
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \\ 1/2 & 1/6 & 1/3 \\ 0 & 2/3 & 1/3 \end{pmatrix}$
$a_2 : o_1, o_2, o_3$		
$a_3 : o_2, o_1, o_3$		

A.4.3. Profile 3,3 - 7

	RSD	PS
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/4 & 1/4 \end{pmatrix}$
$a_2 : o_1, o_3, o_2$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$
$a_3 : o_2, o_1, o_3$	$\begin{pmatrix} 0 & 5/6 & 1/6 \end{pmatrix}$	$\begin{pmatrix} 0 & 3/4 & 1/4 \end{pmatrix}$

A.4.4. Profile 4,4 - 52

	NBM	ABM
$a_1 : o_1, o_2, o_3, o_4$	$\begin{pmatrix} 1/3 & 0 & 1/6 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/3 & 0 & 1/3 & 1/3 \end{pmatrix}$
$a_2 : o_1, o_2, o_3, o_4$	$\begin{pmatrix} 1/3 & 0 & 1/6 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/3 & 0 & 1/3 & 1/3 \end{pmatrix}$
$a_3 : o_1, o_3, o_2, o_4$	$\begin{pmatrix} 1/3 & 0 & 2/3 & 0 \end{pmatrix}$	$\begin{pmatrix} 1/3 & 0 & 1/3 & 1/3 \end{pmatrix}$
$a_4 : o_2, o_1, o_3, o_4$	$\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$

A.5. List of all Profiles Modulo Anonymity and Neutrality

A.5.1. Profiles of size $n = 2$

Profile 2,2-1

	ABM, NBM, EPS, PRA & RSD
$a_1 : o_1, o_2$ $a_2 : o_1, o_2$	$\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$

Profile 2,2-2

	ABM, NBM, EPS, PRA & RSD
$a_1 : o_1, o_2$ $a_2 : o_2, o_1$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

A.5.2. Profiles of size $n = 3$

Profile 3,3-1

	ABM, NBM, EPS, PRA & RSD
$a_1 : o_1, o_2, o_3$ $a_2 : o_1, o_2, o_3$ $a_3 : o_1, o_2, o_3$	$\begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}$

Profile 3,3-2

	NBM, ABM, RSD & PS	PRA
$a_1 : o_1, o_2, o_3$ $a_2 : o_1, o_2, o_3$ $a_3 : o_1, o_3, o_2$	$\begin{pmatrix} 1/3 & 1/2 & 1/6 \\ 1/3 & 1/2 & 1/6 \\ 1/3 & 0 & 2/3 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Profile 3,3-3

	ABM, NBM & PRA	EPS & RSD
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \end{pmatrix}$
$a_2 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \end{pmatrix}$
$a_3 : o_2, o_1, o_3$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2/3 & 1/3 \end{pmatrix}$

Profile 3,3-4

	ABM, NBM & PRA	EPS & RSD
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \end{pmatrix}$
$a_2 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \end{pmatrix}$
$a_3 : o_2, o_3, o_1$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2/3 & 1/3 \end{pmatrix}$

Profile 3,3-5

	ABM, NBM, EPS, PRA & RSD
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 1/2 & 0 \end{pmatrix}$
$a_2 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 1/2 & 0 \end{pmatrix}$
$a_3 : o_3, o_1, o_2$	$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$

Profile 3,3-6

	ABM, NBM, EPS, PRA & RSD
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 1/2 & 0 \end{pmatrix}$
$a_2 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 1/2 & 0 \end{pmatrix}$
$a_3 : o_3, o_2, o_1$	$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$

Profile 3,3-7

	ABM & NBM	EPS	RSD	PRA
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/4 & 1/4 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$
$a_2 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$
$a_3 : o_2, o_1, o_3$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 3/4 & 1/4 \end{pmatrix}$	$\begin{pmatrix} 0 & 5/6 & 1/6 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$

Profile 3,3-8

	ABM & NBM	EPS	RSD	PRA
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/4 & 1/4 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/6 & 1/3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$
$a_2 : o_1, o_3, o_2$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 0 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$
$a_3 : o_2, o_3, o_1$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 3/4 & 1/4 \end{pmatrix}$	$\begin{pmatrix} 0 & 5/6 & 1/6 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$

Profile 3,3-9

	ABM, NBM, EPS, PRA & RSD
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$
$a_2 : o_2, o_1, o_3$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$
$a_3 : o_3, o_1, o_2$	$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$

Profile 3,3-10

	ABM, NBM, EPS, PRA & RSD
$a_1 : o_1, o_2, o_3$	$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$
$a_2 : o_2, o_3, o_1$	$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$
$a_3 : o_3, o_1, o_2$	$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$

List of Figures

5.1. GUI Main Frame	24
5.2. GUI Allocation Main Frame	25
5.3. GUI Allocation Agents	25
5.4. GUI Allocation Objects	26
5.5. GUI Allocation Properties	26
6.1. A naive algorithm for preference profile enumeration	28
6.2. The extended algorithm for enumerating all preference profile modulo anonymity and neutrality	31

List of Tables

4.1. Boston Mechanism 1	17
4.2. Boston Mechanism 2	17
4.3. Axiomatic properties	19
A.1. First Difference	38

Bibliography

- [ABB13] H. Aziz, F. Brandt, and M. Brill. “The Computational Complexity of Random Serial Dictatorship.” In: *Economics Letters* 121.3 (2013), pp. 341–345.
- [ABS13] H. Aziz, F. Brandt, and P. Stursberg. “On popular random assignments.” In: *Proceedings of the 6th International Symposium on Algorithmic Game Theory (SAGT)* 8146 of Lecture Notes in Computer Science (LNCS) (2013), pp. 183–194.
- [AS03] A. Abdulkadiroğlu and T. Sönmez. “School Choice: A Mechanism Design Approach.” In: *American Economic Review* (2003).
- [BBG16] F. Brandl, F. Brandt, and C. Geist. “Proving the Incompatibility of Efficiency and Strategyproofness via SMT Solving.” In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*. 2016.
- [BBH17] F. Brandl, F. Brandt, and J. Hofbauer. “Random assignment with optional participation.” In: *Proceedings of the 16th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (2017).
- [BCKM13] E. Budish, Y.-K. Che, F. Kojima, and P. Milgrom. “Designing Random Allocation Mechanisms: Theory and Applications.” In: *American Economic Review* 103.2 (2013), pp. 585–623. doi: 10.1257/aer.103.2.585.
- [BHS17] F. Brandt, J. Hofbauer, and M. Suderland. “Majority Graphs of Assignment Problems and Properties of Popular Random Assignments.” In: *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems (AMMAS)* (2017).
- [BM01] A. Bogomolnaia and H. Moulin. “A New Solution to the Random Assignment Problem.” In: *Journal of Economic Theory* 100.2 (2001), pp. 295–328.
- [BM04] A. Bogomolnaia and H. Moulin. “Random Matching Under Dichotomous Preferences.” In: *Econometrica* 72.1 (2004), pp. 257–279.
- [GGT89] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. “A Fast Parametric Maximum Flow Algorithm and Applications.” In: *SIAM J. Comput.* 18.1 (1989), pp. 30–55.

- [Gib77] A. Gibbard. "Manipulation of Schemes That Mix Voting with Chance." In: *Econometrica* 45.3 (1977), pp. 665–681.
- [KMN11] T. Kavitha, J. Mestre, and M. Nasre. "Popular mixed matchings." In: *Theoretical Computer Science* 412.24 (2011), pp. 2679 –2690. ISSN: 0304-3975. DOI: <http://dx.doi.org/10.1016/j.tcs.2010.03.028>.
- [KS06] A.-K. Katta and J. Sethuraman. "A solution to the random assignment problem on the full preference domain." In: *Journal of Economic Theory* 131.1 (2006), pp. 231–250.
- [KÜ10] F. Kojima and M. U. Ünver. *The "Boston" School-Choice Mechanism*. Tech. rep. Boston College Department of Economics, 2010.
- [KÜ15] O. Kesten and M. U. Ünver. "A theory of school-choice lotteries." In: *Theoretical Economics* (2015).
- [Man09] M. Manea. "Asymptotic ordinal inefficiency of random serial dictatorship." In: *Theoretical Economics* 4.2 (2009).
- [Sud15] M. Suderland. "Popular Random Assignments." MA thesis. TUM, 2015.
- [Öme09] E. Ömer. "Uniform generation of anonymous and neutral preference profiles for social choice rules." In: *Monte Carlo Methods and Applications* 15.3 (2009), pp. 241–255.