On Fair Allocation of Mixed Goods

TDT4501 - Computer Science, Specialization Project

Sivert Utne

December 14, 2022

Preface

This project report is written as part of the course *TDT4501 - Computer Science, Specialization Project* at Norwegian University of Science and Technology (NTNU). The intended purpose of this project and report is to be a pre-study for a master thesis.

Thank you to the following supervisors for their support, guidance and input:

- *Magnus Lie Hetland*
- *Halvard Hummel*

Abstract

This project aims to analyze algorithms for fair allocation of mixed goods, instances with both divisible and indivisible goods. The project focuses on two main algorithms, one being made to only handle instances with indivisible goods, and one algorithm specialized for mixed goods. The algorithms are assessed by performing experiments on randomly generated instances. The algorithms are then compared to each other using their lowest achieved MMS for an agent in each instance, this will be used as a measure for the fairness of the allocation. The experiments showed that a simple approach of dividing the divisible resource such that there is an equal piece for each agent, and then using the indivisible algorithm proved to perform just as good, as the specialized mixed algorithm. Further experiments were done to verify these results, in addition to experiments of using approximation of MMS values for the mixed algorithm. These experiments were done in combination with a small runtime analysis that indicated that the indivisible algorithm finds allocations faster than the mixed algorithm, but by using approximate MMS values the mixed algorithm is fastest.

Contribution

This project contributes to the existing literature on fair allocation of mixed goods by providing a quantitative analysis of MMS algorithms for fair allocation of mixed goods. The project examines the correlation and comparison of an indivisible goods algorithm and a mixed goods algorithm. Additionally, a quantitative analysis of using approximate MMS values when finding an allocation compared to the exact MMS values is performed. Through the experiments, the results indicate that algorithms for indivisible items work well if the divisible goods are divided into as many equal pieces as there are agents.

The main contribution of this project in then to give insight into how using already established MMS algorithms for indivisible goods can be sufficient for a fair MMS allocation of mixed goods. These results are valuable information with practical applications as if the results described in this project can be utilized for more general settings, it is a pointer that further research on MMS algorithms for mixed goods might not be necessary.

Contents

Chapter 1

Introduction

While the problem of fairly allocating indivisible items and divisible items have been thoroughly analyzed in different settings for each of them, the work done for instances with both divisible and indivisible goods, termed mixed goods, is not yet as thoroughly researched. Instances for only divisible and indivisible resources are easy to imagine. For instance, fairly allocating possessions, gifts and resources, or fairly dividing a cake, money or land. Mixed goods instances aren't as obvious, but examples of instances where mixed goods are relevant would be dividing possessions and money, dividing a plot of land with buildings, or different activities over time.

The basis for this project is to focus on algorithms that use the MaxiMinShare fairness goal in order to achieve fair allocations, and then use these algorithms and compare them to "simpler" algorithms for indivisible goods, where the research is more thorough. The goal is to find out if the algorithms for indivisible goods can be used for mixed goods, and if so, how well they perform.

This report will look at various approaches and algorithms proposed in the relevant literature for how to handle a mixed goods setting. In Section 1.1 we will define the problem and the terminology used in the report. In Section 1.3 we will look at the relevant literature and the different approaches that have been proposed. In Chapter 2 we will present the approach used to compare and analyze these different approaches. In Chapter 3 we will present the experiments we have conducted and the results we have obtained. Finally, in Chapter 4 we will conclude the report and discuss future work.

1.1 Preliminaries

Here I present all relevant terminology, notation and definitions that will be used throughout the report. Be aware that my notation does not use the most common notations from the literature, but instead uses a notation that I find more intuitive, which mostly means that use different letters than other literature, in order to better distinguish between the different types of goods.

Mixed Goods

When discussion mixed goods we are talking about a mix of both indivisible (i.e. paintings, cars etc.) and divisible goods (i.e. money, land, fuel, time etc.) . A mixed goods instance can contain any combination of number of divisible and indivisible goods.

Item
An indivisible good will be referred to as an *item*, i, where the number of items i_n defines the set of all items I. Formally:

$$i \in \{1, 2, ..., i_n\} \equiv i \in I$$

Cake
A divisible good will be referred to as *cake*, c, where the number of cakes c_n defines the set of all cakes C. Formally:

$$c \in \{c_1, c_2, ..., c_n\} \equiv c \in C$$

Furthermore, since a cake is divisible, a piece of the cake will be defined as en interval of the entire cake, where the entire cake is defined over the interval $[0, 1]$, such that $c = c_{[0,1]}$. A slice of the cake is then defined as $c_{[x,y]}$ where $x, y \in [0, 1]$ and $x \leq y$ where the size of the slices of cake is $c_y - c_x$. As this project focuses on homogenous cake (see Section 1.1), the exact interval of the cake is not important, a slice of cake will for this project then only needs to be specified to by its size z, where $c_z \equiv c_y - c_x \equiv c_{[x,y]}$ where we can assume $x = 0$.

Good
Since we are in a mixed setting, a good can be either divisible or indivisible. It is therefore convenient to able to refer to goods g that can be either an item or cake. Formally:

$$g \in \{1, 2, ..., i_n\} \cup \{c_1, c_2, ..., c_n\} \equiv g \in I \cup C \equiv g \in G$$

Homogenous Cake

A *Homogenous cake* is a cake where all agents valuations for a piece of cake is only proportional to the size of the piece of cake. Formally, for a piece of cake from x to y ($c_{[x,y]}$) and an agent $a \in A$ the value of the pieces is proportional to that agents value of the entire cake:

$$v_a(c_{[x,y]}) = (y - x)v_a(c)$$

Heterogenous Cake

A *Heterogenous Cake* is a cake in which each agent has their own density function $d_a : [0, 1] \to \mathbb{R}^+ \cup \{0\}$ which captures how the agent values different parts of the cake. The value of agent a over a finite union of intervals $S \subseteq [0, 1]$ is defined as $v_a(S) = \int_S d_a dx$.

Agent

An agent a is a person or entity that is to be allocated goods. The number of agents n defines the set of all agents A. Formally:

$$a \in \{1, 2, ..., n\} \equiv a \in A$$

Valuations

A valuation v_a is a function for each agent a that takes a set of goods $\{g_1, g_2, ...\}$ and finds this agents value for this set of goods. For simplicity we will write $v_a(\{g\})$ as $v_a(g)$. Since we only are dealing with goods, we have that the value for all items for all agents is positive, formally:

$$\forall a \in A, \forall g \in G, v_a(g) \geq 0$$

Additive valuations

A well-studied subclass of monotone valuations is that of additive valuations, where an agent's value of any subset of items is equal to the sum of the values of

individual items in the set. Formally:

$$\forall S \subseteq G, \forall a \in A, v_a(S) := \sum_{g \in S} v_a(g)$$

By extension it is also assumed that the value of an empty set of goods is zero, formally:

$$\forall a \in A, v_a(\emptyset) = 0$$

Bundle

The collection of goods, or parts of cake that an agent receives in an allocation is called a bundle B. Formally the bundle agent $a \in A$ gets is B_a where:

$$B_a \subseteq G$$

Allocation

An allocation $\mathcal{A}$ is an n-partition of the set of goods G. The resulting allocation is complete if the set of bundles $\mathcal{A} := \{B_1, B_2..., B_n\}$ allocates all goods in the instance such that no two agents receives the same item, and that all cake is fully allocated amongst the agents. Formally:

$$\forall B_x B_y \in \mathcal{A}, B_x \cap B_y = \emptyset$$

Proportionality (PROP)

Proportionality is a fairness-notion where each agent receives a bundle valued proportionally compared their value of all goods. Formally, an allocation $\mathcal{A}$ satisfies proportionality PROP if:

$$\forall a \in A, v_a(B_a) \geq \frac{v_a(G)}{n}$$

MaxiMinShare (MMS)

MaxiMinShare is a relaxation of the proportionality fairness notion often used for instances with indivisible goods. MMS looks at what each agent would expect to

receive if they were to divide the instance themselves, and then receive the smallest valued bundle. This means each agent attempt to maximize the minimum bundle.

An example of how each agents MMS allocation can look like is shown in Figure 1.1. The MMS value of each agent is then the value of the worst/smallest bundle in their allocation. In this report this value will be referred to as $\text{MMS}_a(\text{I})$. If it is clear from context what instance is being used, the shorthand MMS_a will be used. When talking about MMS in general in this report, simply MMS or MMS_a is used. This is simply a shorthand to talk about the "expected" or 1-MMS value of an agent, and the $\alpha - MMS$ value an algorithm guarantees, or achieves for an agent.

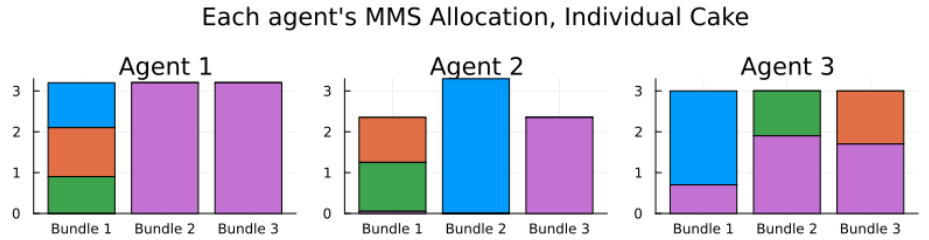

Figure 1.1: Each Agents MMS Allocation for the individual cake instance from Section 1.2.1.

Envy-Freeness

Envy-freeness is another common fairness notion that is prominent int the literature for indivisible goods. This project does not directly apply any of these fairness notions, but as they are still relevant I will explain them briefly here.

EF
An allocation $\mathcal{A}$ is said to be envy-free (EF) if for every pair of agents $a_1, a_2 \in A$, we have $v_{a_1}(B_1) \geq v_{a_1}(B_2)$. In other words, no agent prefers the bundle of another agent over their own bundle.

EF1
An allocation $\mathcal{A}$ is said to be envy-free up to one item (EF1) if for every pair of agents $a_1, a_2 \in A$ there exists a good $g \in B_{a_1} \cup B_{a_2}$ such that $v_{a_1}(Ba_1 \backslash \{g\}) \geq v_{a_1}(B_{a_2} \backslash \{g\})$. In other words, no agent prefers the bundle of another agent over their own bundle if the other agent looses one good.

EFM
An allocation $\mathcal{A}$ is said to satisfy *Envy-Freeness for Mixed goods* (EFM) if for any agents $a_1, a_2 \in A$,

- if agent a_2's bundle consists of only indivisible goods, there exists $g \in B_{a_2}$ such that $v_{a_1}(B_{a_1}) \geq v_{a_1}(B_{a_2} \backslash \{g\})$;
- otherwise, $v_{a_1}(B_{a_1}) \geq v_{a_1}(B_{a_2})$.

Put simply, EFM is achieved if any agent that receives cake is not envied by any other agent. From this definition we see that when the goods are all divisible, EFM reduces to EF; when goods are all indivisible, EFM reduces to EF1. Therefore EFM is a natural generalization of both EF and EF1 to the mixed goods setting.

1.2 Problem Instance

This project and report will focus and analyze instances with a single divisible good, which will be denoted as C, this simplification does not reduce the generality of having multiple cakes on its own, as a set of cakes can be combined into a single divisible heterogenous cake. this project however focuses on instances with homogenous cake(s), so this generalization is not applicable as agents can value cakes differently, meaning combining multiple homogenous cakes would also result in a heterogenous cake.

In short, instances, I, considered in this project consists of the following:

- set of agents: $a \in A = \{1, 2, ..., n\}$.
- set of indivisible items: $i \in I = \{1, 2, ..., i_n\}$
- single divisible good (cake): C, where a piece of cake of size z is $C_z = C_{[0,z]}$.
- set of goods: $g \in G = \{1, 2, ..., m\} := I \cup C$.
- set of valuation functions $v \in V$ where v_a is the valuation function for agent a.

1.2.1 Cake Sizes

In order to compare the algorithms for different types of cakes later the instances are split into four main categories.

Small Cake

A Cake is considered *small* if the value of the entire cake for each agent is equal to or less than the value of any indivisible item See Figure 1.2 for example. Formally:

$$\forall a \in A, \forall i \in I, v_a(C) \leq v_a(i)$$

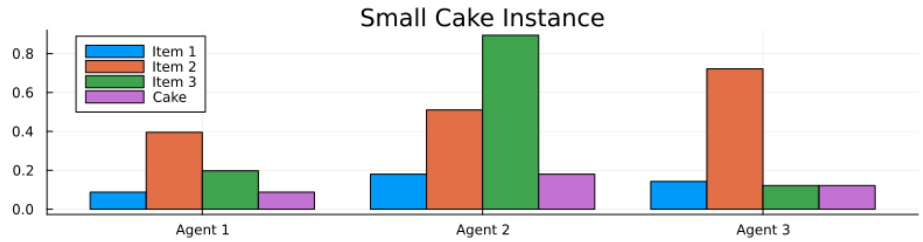

Figure 1.2: Visualization of agent's valuations for an instance with a *small cake*.

Medium Cake

A Cake is considered *medium* if the value of the entire cake for each agent is larger than, or equal to the smallest item, and smaller than or equal to the agent's largest item. See Figure 1.3 for an example. Formally:

$$\forall a \in A, \exists i_1, i_2 \in I, v_a(i_1) \leq v_a(C) \leq v_a(i_2)$$

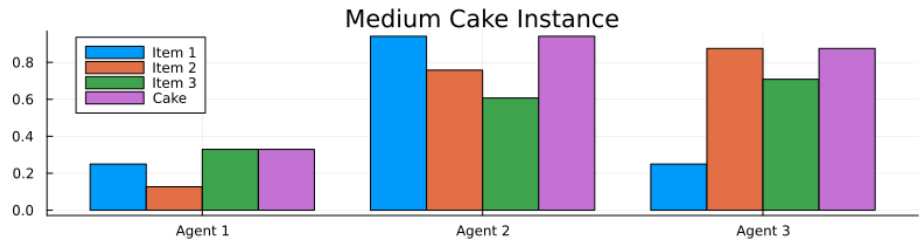

Figure 1.3: Visualization of agent's valuations for an instance with a *medium cake*.

Large Cake

A Cake is considered *large* if the value of the entire cake for each agent is equal to or larger than the sum of the value of all indivisible items. Formally:

$$\forall a \in A, v_a(C) \geq \sum_{i \in I} v_a(i)$$

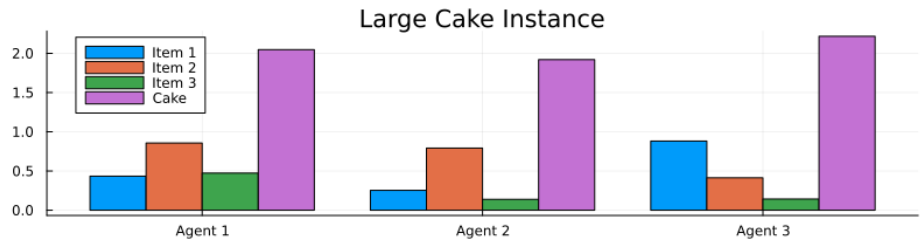

Figure 1.4: Visualization of agent's valuations for an instance with a *large cake*.

Individual Cake

Finally, we have instances with individual cake, meaning that each agent are free to value the cake however they want compared to their indivisible items. In other words, in these instances one agent could value the cake as a *large* cake, while another agent sees it as a *small* cake, hence the name "individual".

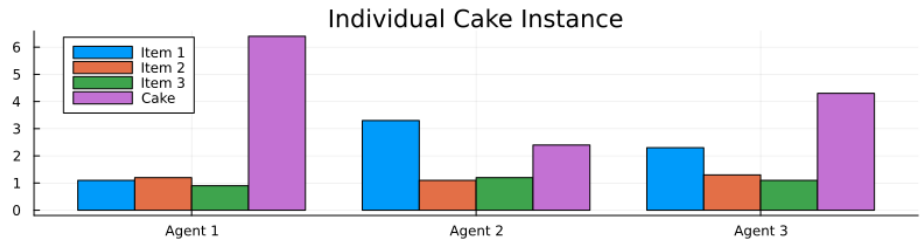

Figure 1.5: Visualization of agent's valuations for an instance with a *individual cake*.

Together these four categories of cake cover all possible mixed instances with a single divisible good, and lets us easier examine and compare results for different types of cakes in Chapter 3 and Section 3.2.

1.3 Previous Work

Let's briefly look at what has been done in the literature for the three main different types of instances that are relevant, I won't go into any detail except list their most relevant findings. Most of this information, and more is covered in the survey [1] and the respective articles.

1.3.1 Indivisible Goods

There has been thorough research on a wide variety of fairness-notions and goals for indivisible goods. The literature describes many algorithm solutions for each fairness-goal such as EF1, PO, MMS, MNW and more. In addition to changes and restrictions of the instance such as binary valuations, positive and negative valuations (chores) and allocation with constraints.

In terms of MMS, as is the most relevant for this report, it is found that an MMS allocation may not always exist for indivisible goods [8]. It is also found that for Additive valuations and a constant number of agents, a 2/3 allocation can always be found [8] in polynomial time. Further several algorithms with varying approximate MMS-guarantees have been proposed. The highest guarantee yet is $\frac{3}{4} + \frac{1}{12n}$ [5].

1.3.2 Divisible Goods

Commonly considered a "simpler" allocation problem, the problem of divisible goods has also been thoroughly researched. For these instances notions such as envy-freeness isn't as applicable. Most of this research is focusing on achieving proportional division of cakes.

1.3.3 Mixed Goods

For mixed goods the research has not yet reached the stage where all areas are thoroughly covered. There have however been minor previews of different aspects of mixed instances in various article. Most notable is [2], which proposes the envy-freeness notion of EFM (Envy-Freeness Mixed). In the context of mixed resource [4], instances where the cake can be negatively valued was examined. A similar instance is also examined in [7]. The article I will base most of my experimentation and that I take my algorithm from is the article of MaxiMin

Fairness for Mixed Goods [3], this article also explains how and when an MMS-allocation is guaranteed to exist for a mixed instance.

In general, both the research and the results for indivisible and divisible goods are well established. Finding a way to connect mixed goods to these instances will then greatly increase the understanding of the problem.

Chapter 2

Method

2.1 Choosing Algorithms and Comparison Metric

My initial idea was to compare a lot of different algorithms in order to find when which algorithm is best for certain instances. This changed early as comparing algorithm specialized for completely different fairness metrics would likely not yield any substantially significant results. AS an example, an algorithm aimed at finding an optimal MMS allocation wouldn't really be fair to compare to an algorithm specifically designed to find the best EF1 allocation. A way around this could be to compare the algorithm using a bunch of different fairness metrics, but this would require much more time and the same issue would remain, each algorithm would most likely outperform other algorithms.

The choice of algorithms was then highly dependent upon what metric i chose to use in order to compare the algorithms. From the literature i found that the research on MMS for mixed allocation wasn't as thoroughly researched as it has been for indivisible instances. This made me choose to compare algorithms using their achieved MMS. Since the allocations are supposed be as fair as possible as well, it doesn't make sense to try to maximize the total MMS of the allocation, but rather the MMS of the agent that "gets the worst deal".

Having chosen a fairness notion, the choice of algorithm was initially to compare a simple algorithm such as round-robin, with more specialized algorithms for mixed goods. The idea was here to see if the simpler, and often faster algorithms

could compete with the complex ones. However, after some discussion it was determined that, as mentioned, comparing algorithms not aimed at MMS on their achieved MMS, would not really give any valuable insight. That is where the idea of comparing two algorithms with exact same fairness goal came from. The final choice was to see how an indivisible algorithm with a $\frac{1}{2}$MMS goal compares to a specialized algorithm that also has a $\frac{1}{2}$MMS goal.

When comparing the algorithm, the *Mixed MMS* value of the instance will of course be used for both algorithms to calculate the achieved MMS. Example of Mixed MMS allocations are shown in Figure 1.1.

Another analysis that will be performed is that since the mixed algorithm is specified to compute the MMS value for all agents, $\text{MMS}_a \forall a \in A$, which even for smaller instances can be time consuming (a Mixed Integer Programming solution is used to find these exact MMS allocations.). I was therefore curious how the exact same algorithm would handle using only approximated MMS values instead. These approximated MMS values can either be found by normalizing each agents valuations such that $v_a(G) = n$, in which case the exact MMS value will be ≤ 1. I however chose to approximate the MMS value by using each agent's PROP_a value instead, which in practice should scale equally as the other method. Formally i approximate the MMS values with $\text{MMS}_a \leq \text{PROP}_a = \frac{v_a(G)}{n}$.

2.2 The Algorithms

2.2.1 $\frac{1}{2}$MMS Indivisible Algorithm

This algorithm I will not implement myself, as I will utilize an already implemented algorithm from [6] for $\frac{1}{2}$MMS under works by *Hummel* and *Hetland*. This specific implementation has the additional specification of handling cardinality constraints. When I am using the algorithm I will not be utilizing these constraints by simply setting the constraint for all items to be the same as the total number of items. As the main benefit of using a already established algorithm is that there isn't such a need to understand all the details of the algorithm, as such I won't go into exact details of this algorithm here, and instead refer to the source directly for more details[6].

2.2.2 $\frac{1}{2}$MMS Indivisible Algorithm, Cut Cake in n Pieces

This isn't exactly a new algorithm, but for simplicity I will name it as such. This "algorithm" simply uses the algorithm from Section 2.2.1, with the only difference being that there is now a small pre-processing step where the cake is cut into n equal pieces, where n is the number of agents.

Take for instance the following matrix of valuations for three agents and two goods (one item and one cake). Each row represents an agent, and each column represents a good, where the final column is the cake. The conversion then looks like this:

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} \overset{\text{cut cake}}{\rightarrow} \begin{bmatrix} 1 & 1/3 & 1/3 & 1/3 \\ 2 & 2/3 & 2/3 & 2/3 \\ 3 & 3/3 & 3/3 & 3/3 \end{bmatrix}$$

The algorithm then uses this new instance, and after the algorithm is completed a post processing step of gathering up these pieces again is performed by simply adding together $1/n$ per piece of cake an agent gets.

A conversion for a fair allocation of the instance above could look like this, the matrix show the allocation by having values from 0 to 1, which for items means they are either allocated to that agent or not. When converting back, since we know which rows correspond to pieces of cake, we simply add these pieces together to find how much of that cake is allocated to that agent in total.

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \overset{\text{gather pieces}}{\rightarrow} \begin{bmatrix} 0 & 2/3 \\ 0 & 1/3 \\ 1 & 0 \end{bmatrix}$$

To find the value each agent has for their bundle these values can they simply perform element-wise multiplication of the allocation matrix and the valuation matrix, and sum the rows. This gives the following result:

$$\begin{bmatrix} 0 & 2/3 \\ 0 & 1/3 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 2/3 \\ 0 & 2/3 \\ 3 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 2/3 \\ 2/3 \\ 3 \end{bmatrix}$$

A consequence of the cake cutting is that the size of the instance increases by n items per cake. This will have a negative impact on the running time of the algorithm. This is also why, in an attempt to "mimic" a divisible good, cutting a cake into as tiny pieces as possible isn't a viable strategy.

2.2.3 $\frac{1}{2}$MMS Mixed Algorithm

This is an implementation of the Half-MMS Algorithm for Homogenous Cake described by *Bei et al.*[2]. For completeness the pseudocode for the algorithm is shown in Algorithm 1. The algorithm is split into two main phases.

Phase 1: Large Items

Initially any item i worth more than $\frac{1}{2}\text{MMS}_a$ for any agent a are allocated to that agent. This is done until no more such items exists in the instance. Each agent (and of course the item) is then removed from any further consideration. See while loop from line 3 in Algorithm 1.

Phase 2: Small Items and Cake

Next an empty bundle B is initialized. A small item (width less than $\frac{1}{2}\text{MMS}_a$) is then added to the bundle and each agent looks at this item, and figures out how much cake they would need in addition to this item in order to achieve a total valuation of $\frac{1}{2}\text{MMS}_a$ for the bundle, formally $v_a(B) + xv_a(C) == \frac{1}{2}\text{MMS}_a$. This bundle, and that amount of cake is then allocated to the agent that required the *least* amount of cake. This repeats until either all items are gone, or all agents have received their $\frac{1}{2}\text{MMS}_a$. See while loop from line 8 in Algorithm 1.

Finally any remaining items or cake is allocated by using bag-filling to the last agent.

Algorithm 1 Mixed-MMS-Homogenous

Require: Agents A, indivisible goods I and a homogenous cake C and Valuation Functions V.

Compute MMS_a, for each $a \in A$.
$B_1, B_2, ..., B_n \leftarrow \emptyset$
while $\exists a \in A, i \in I$ s.t. $v_a(i) \geq \frac{1}{2}\text{MMS}_a$ **do**
    $B_a \leftarrow \{i\}$
    $A \leftarrow A \backslash \{a\}$
    $I \leftarrow I \backslash \{i\}$
end while
while $n \geq 2$ **do**
    $B \leftarrow \emptyset$
    Add one item to B until $\exists a \in A, v_a(B) \geq \frac{1}{2}\text{MMS}_a$ or $B = I$.
    For each $a \in A$ set x_a s.t. $v_a(B \cup C_{x_a}) \geq \frac{1}{2}\text{MMS}_a$.
    $a^* \leftarrow \arg\min_{a \in A} x_a$
    $B_{a^*} \leftarrow B \cup C_x$
    $A \leftarrow A \backslash \{a^*\}$
    $I \leftarrow I \backslash B$
    $C \leftarrow C \backslash C_x$
end while
Give all remaining items to last agent.
return $\{B_1, B_2, ..., B_n\}$

2.2.4 Visualized Allocations

In order to visually distinguish how these three variations of the algorithms work, more specifically how they handle the cake, I included Figure 2.1. The stacked bars show how each agent values each bundle in the allocation, so both their own bundle, and the others bundle (semi-transparent). The dashed black line shows MMS_a for that agent $a \in \{1, 2, 3\}$, and the red dashed line is $\frac{1}{2}\text{MMS}_a$. See Figure 1.1 to see exactly how each agent achieves this MMS. The goal for the algorithms is then to allocate the goods such that the value each agent gives their own bundle crosses the red dashed line.

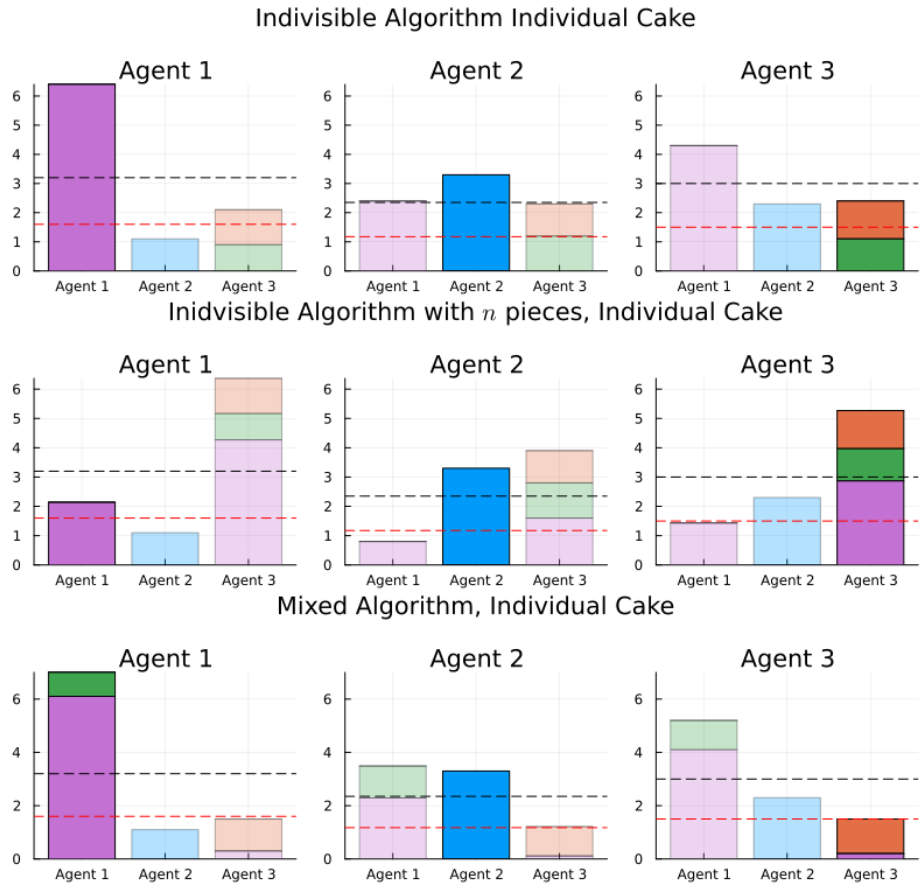

Figure 2.1: Result of the three main algorithms on the same instance with Individual Cake from Section 1.1.

For instance if we look at the top left chart, which is how Agent 1 sees the allocation after using the Indivisible algorithm. Agent 1 receives well over their expected MMS value, and Agent 1 is not envious of any other agent as the other agents bundles (semi-transparent) are all lower than their bundle. We also see (unsurprisingly) that the indivisible algorithm doesn't divide the cake, while the after cutting the cake into 3 pieces, the same algorithm gives 1 piece of cake to agent 1, and 2 pieces to agent 3. Finally the mixed algorithm gives just enough cake to agent 3 for agent three to receive $\frac{1}{2}\text{MMS}_3$, and then gives the rest of the cake to agent 1.

2.3 The Experiments

As described in Section 2.1 we will compare the algorithms using their lowest achieved MMS value. This will allow us to verify if and when the algorithms are able to maintain their MMS guarantee.

The following method will be used to compare and analyse the algorithms. A large collection of randomly created instances is generated according to Section 2.3.1. Each of the algorithms will then be tasked with allocating these goods. This way we are sure that the algorithm have the exact same instance to work with, so a significant difference/distribution for one algorithm gives an indication that one algorithm might be better than the others. Of course since the algorithms are targeting $\frac{1}{2}$MMS, a result where one algorithm is better than the other does not necessarily mean that the algorithm is better. The mixed algorithm for instance essentially stops allocating goods fairly once the guarantee is reached, and might therefore actually be able to achieve a higher minimum MMS value by replacing the bag filling with a more complex algorithm. In addition to the MMS, the algorithms will also be compared in regards to the time they use to find their allocation.

2.3.1 Instances

When testing the algorithms they will be compared using the exact same instances. Each instance is generated randomly, that is each agents valuations are created randomly in the range $[0, 1]$ while following the requirements for the different cake sizes described in Section 1.2.1. For the results desscribed in this project, each plot has 10 000 individual instances. The instances are generated using the following parameters:

- Number of agents: $n = 4$
- Number of Goods: $m = 8$
  - Number of cakes: $c_n = 1$
  - Number of items: $i_n = m - c_n = 7$

This number of agents and goods were chosen as they gave a good balance between their complexity and the time it took to run the algorithms. During experimentation other combinations of agents and goods were also tested, these tests are explained in Section 3.1.2.

Chapter 3

Results

Note: The code used for this project has not been properly peer-reviewed and as such might contain oversights and/or bugs. For completeness, all source code for this project can be found at: github.com/sutne-NTNU/TDT4501-Computer-Science-Specialization-Project.

3.1 Experimentation Results

3.1.1 Regular Indivisible Algorithm vs. Mixed Algorithm

This analysis simply looks at the results of the indivisible algorithm from Section 2.2.1 and compares it to the mixed algorithm from Section 2.2.3. This will give us an idea of how the indivisible algorithm handles the mixed instance. As explained in Chapter 2, both algorithms will be measured in terms of their Mixed MMS values, explained in Section 1.1.

The results of this analysis are shown in Figure 3.1. Since the plots contain a fair bit of information ill explain that here. The scatter plot shows each individual instance as a single dot, where one algorithm represents the dots x-value, and the other the y-value. This means that if $x = y$ both algorithm achieved the exact same MMS, the line $x = y$ divides then the plot in half. Dots that fall closer to the x-axis means that algorithm X performed better than algorithm Y and vise versa. To the right of the scatter plot is a pie chart that summarizes the scatter diagram by showing the percentages of instances where each instance was

better, or equal. Due the a lot of the instances/dots overlapping in the actter diagram there is also a histogram on the bottom that shows the distribution of the achieved MMS values for each algorithm on top of each other.

Note that in Figure 3.1 and Figure 3.4, you can see a tiny amount of instances that fall below $\frac{1}{2}$MMS in the histogram for the mixed algorithm. After inspecting these instances it seems this is only a floating point rounding error, as the lowest achieved MMS for these instances was 0.49999999999999944.

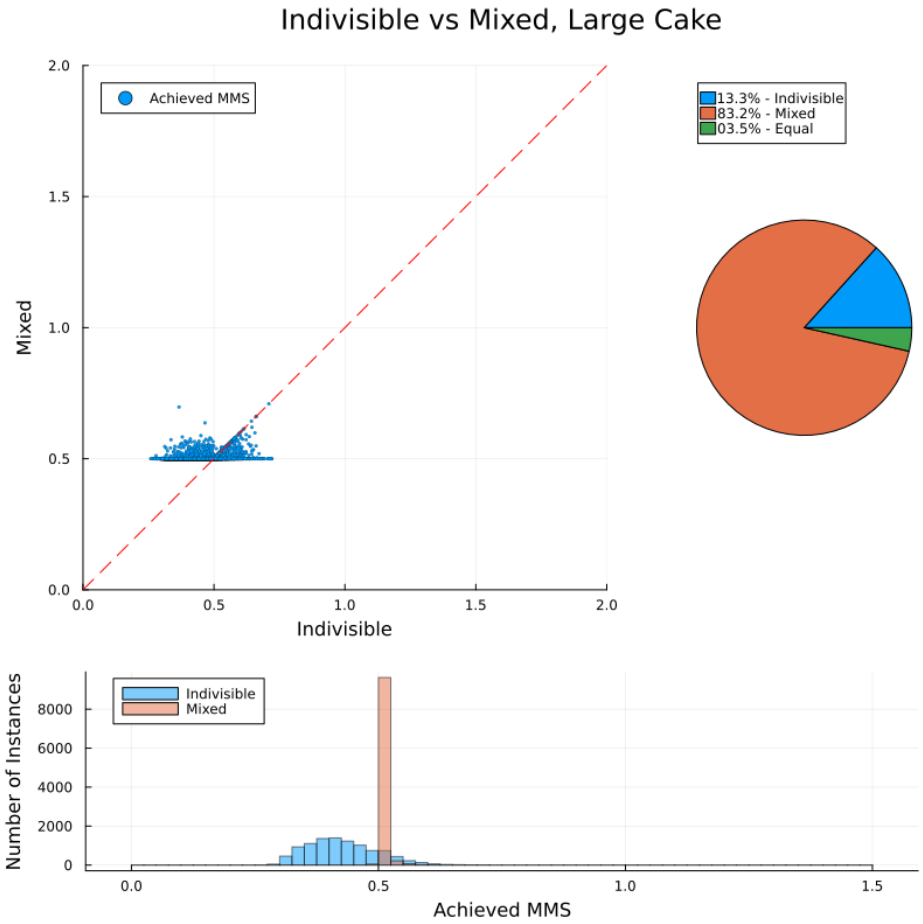

Figure 3.1: Regular indivisible Algorithm vs Mixed Algorithm.

3.1.2 Cutting the cake

Now it is time to perform the analysis when cutting the divisible resource into n pieces. We will perform this analysis on all cake variants seperately to more accurately analyze their results.

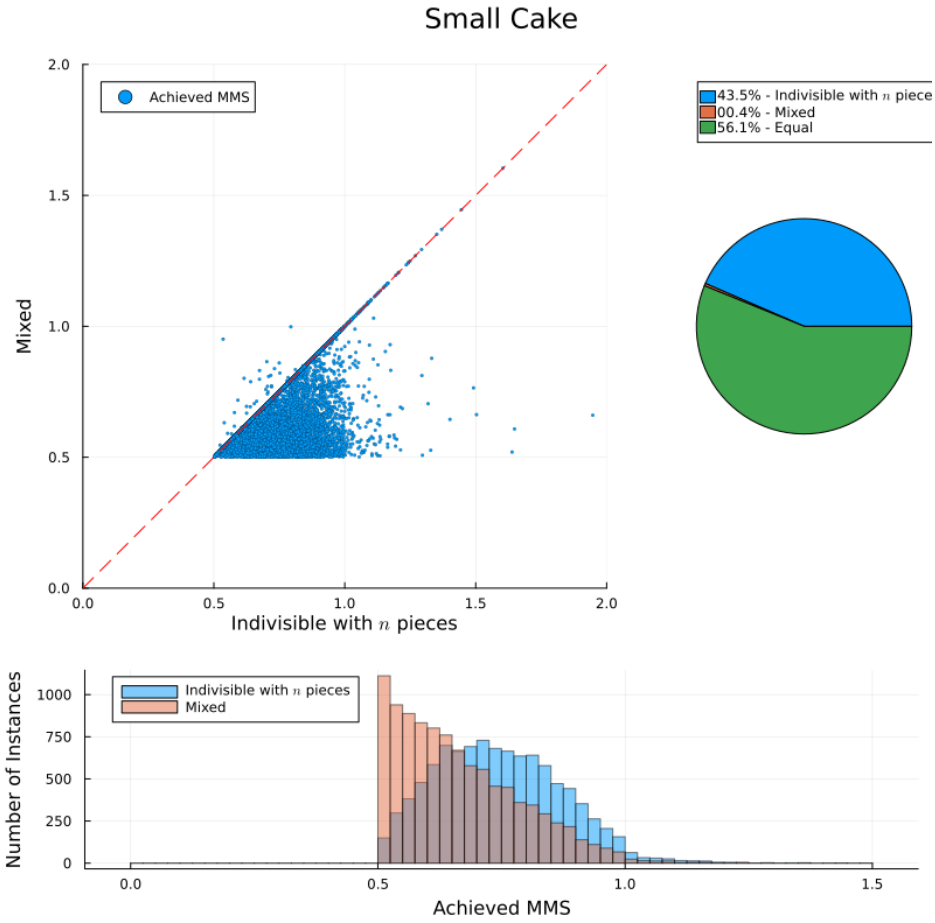

Figure 3.2: Indivisible Algorithm Cutting cake into n pieces vs. Mixed Algorithm for Small Cakes.

We observe in Figure 3.2 that as expected, the indivisible algorithm manages to

maintain its $\frac{1}{2}$MMS guarantee as the size of the cake small enough that there really isn't ever any need to cut it anyway. This was shown in a smaller experiment, not included in this report, the the indivisible algorithm managed to maintain its $\frac{1}{2}$MMS guarantee even without cutting the cake.

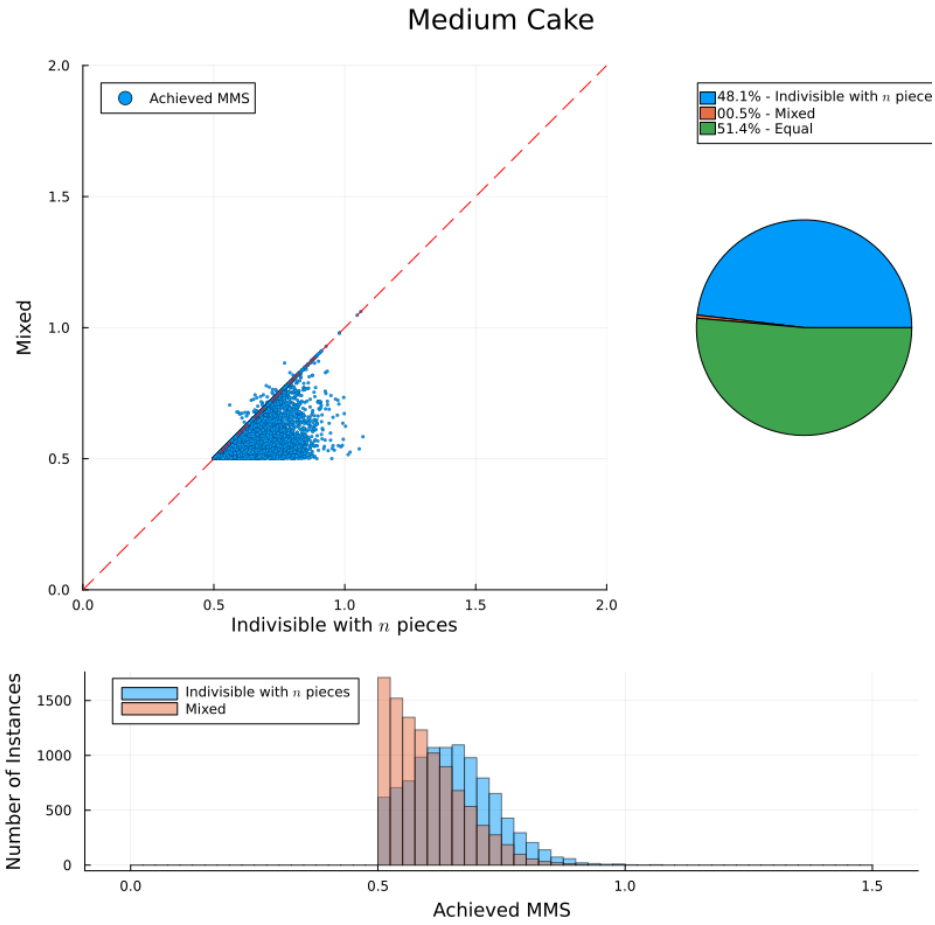

Figure 3.3: Indivisible Algorithm Cutting cake into n pieces vs. Mixed Algorithm for Medium Cakes.

In Figure 3.3 not much has changed, but the mixed algorithm has more cake to work with so it is more often able to, and forced to split the cake to achieve

$\frac{1}{2}$MMS.

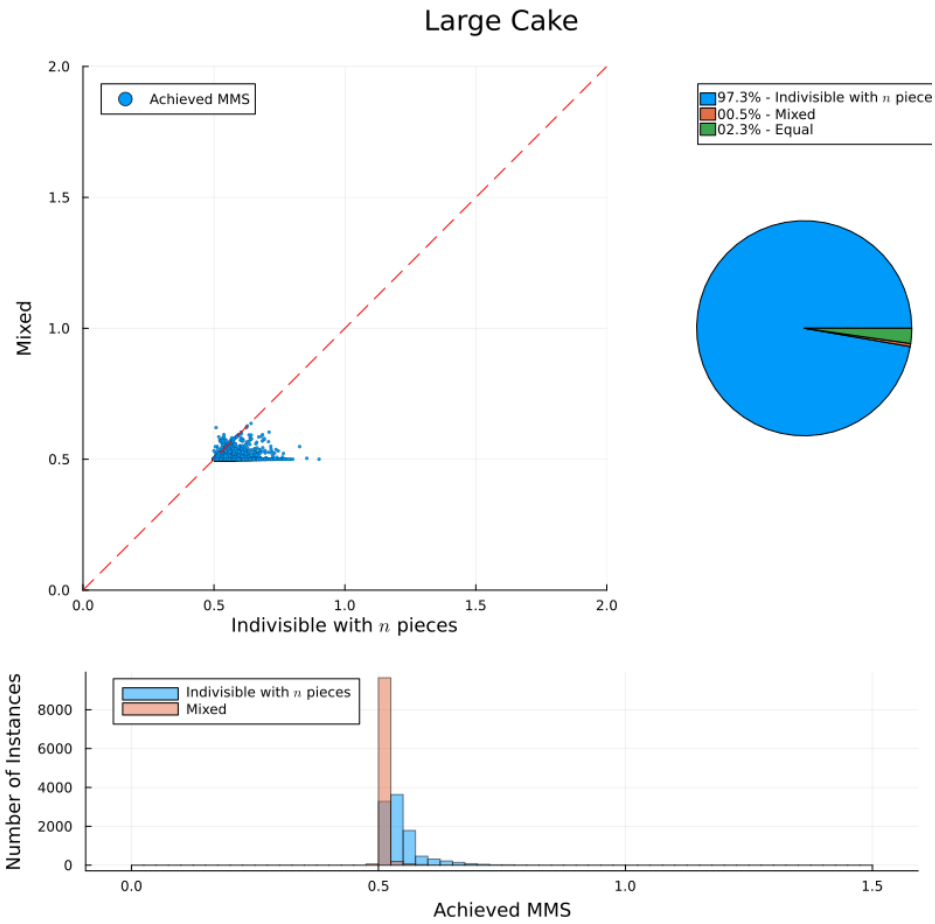

Figure 3.4: Indivisible Algorithm Cutting cake into n pieces vs. Mixed Algorithm for Large Cakes.

AS the cake has grown larger, we see in Figure 3.4 that the mixed algorithm is now almost always forced to cut the cake, achieving exactly $\frac{1}{2}$MMS for the worst bundle.

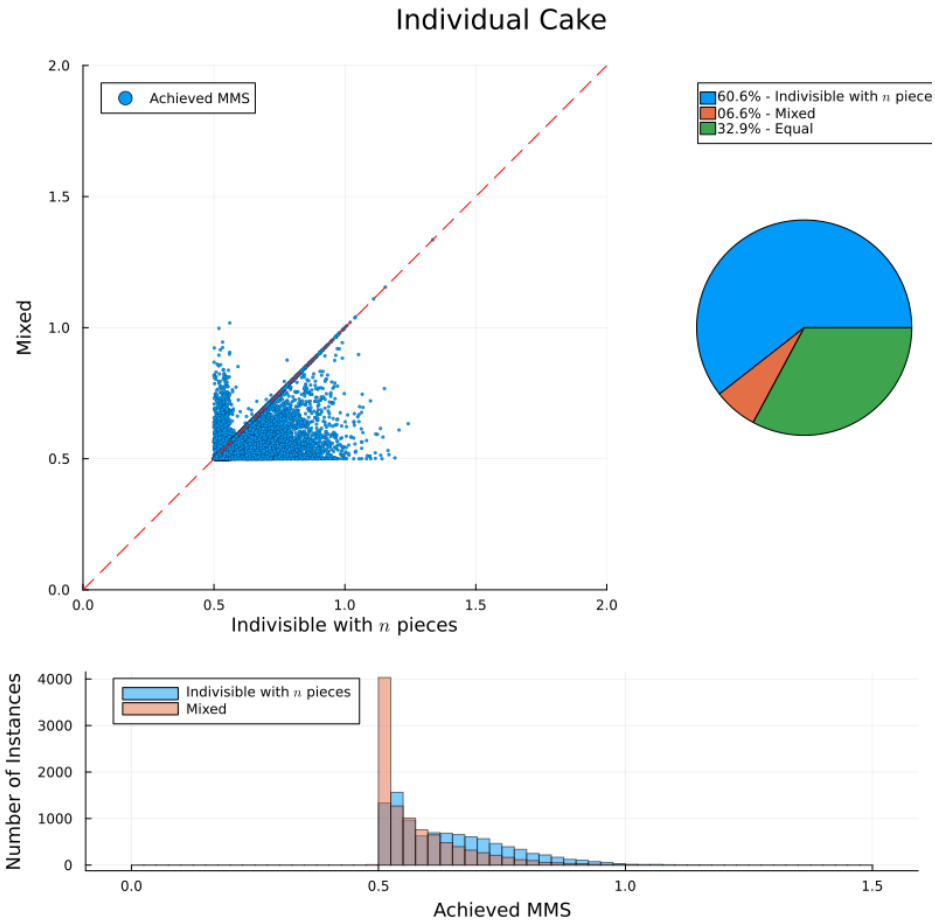

Figure 3.5: Indivisible Algorithm Cutting cake into n pieces vs. Mixed Algorithm for Individual Cakes.

Other Experiments

In order to verify that the results of the indivisible algorithm managing to maintain its $\frac{1}{2}$MMS guarantee aren't exclusive to this exact type of instance, there was also performed a more general analysis on 10 000 instances of varying sizes.

These instances had a random number of agents n where $2 \leq n \leq 7$, and where

the number of goods m were $n \leq m \leq 2n$. These limits were set based on the time it takes to allocate these instances for both algorithms. The values were also chosen to ensure these instances cover a more nuanced range of instances, as the main analysis is done on instances were the number of goods could be evenly divided amongst the agents. I limit the instances to have at least n goods, as the indivisible algorithm (without cutting the cake), each agent's MMS allocation would give them an empty bundle as there wouldn't be enough items to go around. This limit could have been removed however when only using the algorithm that cats the cake into n pieces, as this way it would always be at least one slice of cake to each agent.The results of these verifying experiment are shown in Figure 3.6.

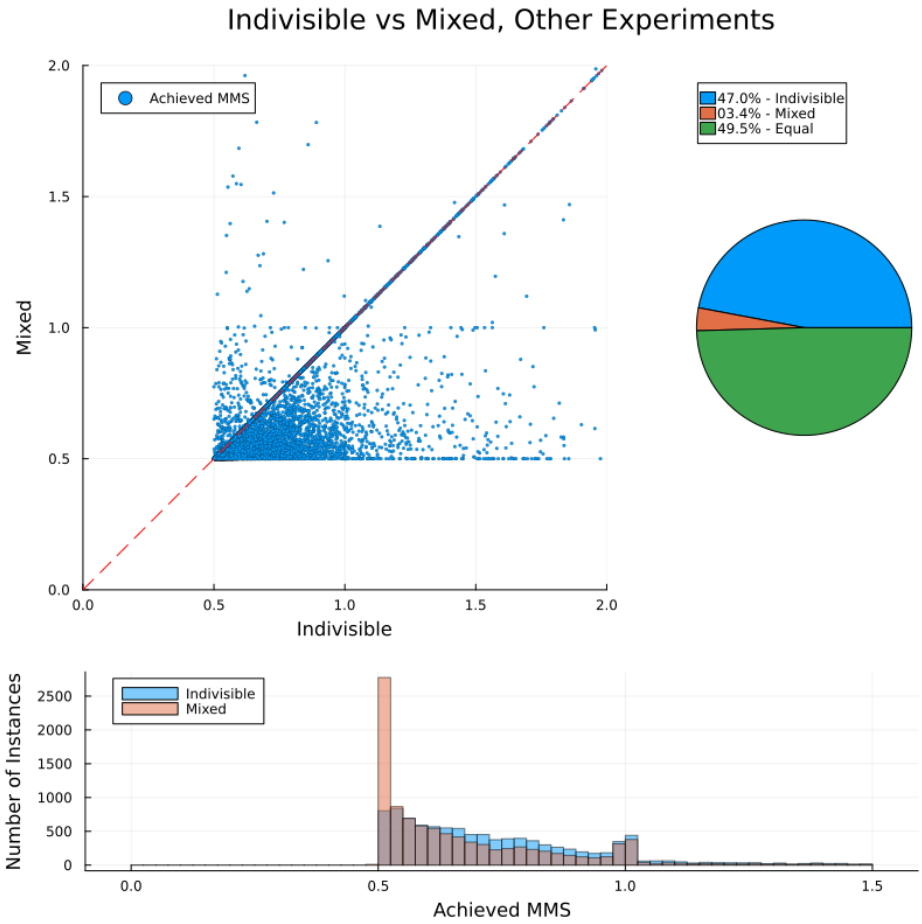

Figure 3.6: Indivisible Algorithm Cutting cake into n pieces vs. Mixed Algorithm for a wide range of agents and number of goods.

3.1.3 Using Approximated MMS for the Mixed Algorithm

Next I will perform the same analysis as when comparing the algorithms to compare the results of using a approximated MMS value for the mixed algorithm. The results are shown in Figure 3.7.

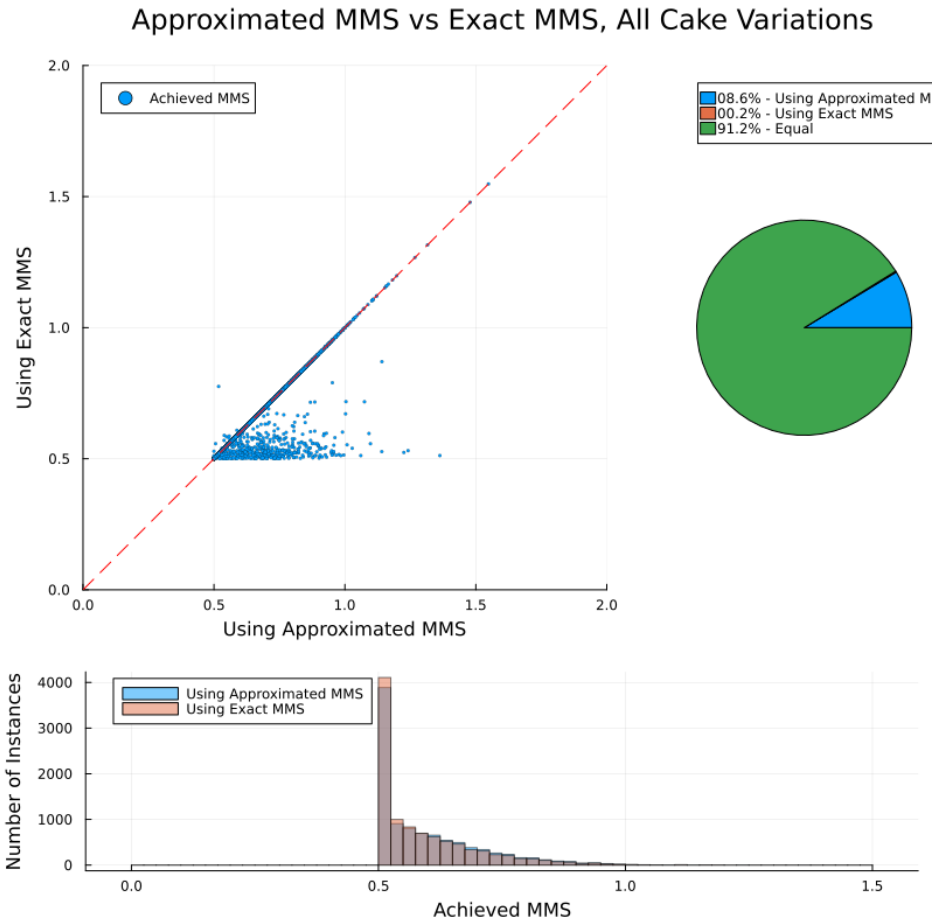

Figure 3.7: Approximate vs. Exact MMS calculation for the Mixed Algorithm.

Finally a very basic runtime analysis was done for the three variations of algorithms. These results are the average over 10 000 allocations of each type of cake. The results are shown in Table 3.1. Again these experiments are run on instances with $n = 4$ and $m = 8$. A mor comprehensive analysis should look at how the time is affected by changing the number of agents and goods as well to see how well each algorithm scales along the size of the instance.

Table 3.1: Average time each algorithm spent allocating an instance.

	Indivisible n Pieces	Mixed Exact	Mixed Approxmation
Small Cake	0.138ms	187ms	0.00538ms
Medium Cake	0.170ms	129ms	0.00215ms
Large Cake	0.148ms	25.0ms	0.00589ms
individual Cake	0.200ms	121ms	0.00252ms

3.2 Discussion

3.2.1 Indivisible vs. Mixed Algorithm

As mentioned in Chapter 1, I initially expected the indivisible algorithm to not be able to maintain its 1/2MMS guarantee, even when cutting the cake into pieces. My expectation was that as the size of the cake pieces get smaller and smaller, the indivisible algorithm will approach the results of the mixed algorithm. My intuition told me that as the number of pieces approached ∞, the indivisible algorithm would essentially be turned into an algorithm for divisible goods, albeit an incredibly slow one as a massive instance such as that would require immense computing power and time using the current known algorithms.

The initial results with the purely indivisible cake (Figure 3.1) seemed to confirm my intuition, in that the algorithm mostly achieves MMS values around 0.4 MMS, and that is with the cake being only the exact sum of the indivisible items, this achieved MMS value would strictly decrease for an instance if the cake would increase in size.

The results from the next experiments were therefore somewhat surprising. Immediately after cutting the cake into n pieces the indivisible algorithm achieves its guarantee, regardless of the size of the cake. We also see as for the small and medium cake both algorithm achieve the exact same MMS over 50% of the time which likely mean they find hte exact same allocation. AS the cake gets large though, the indivisible algorithm achieves higher MMS for almost all instances, which is quite surprising as one would expect an indivisible algorithm to perform worse the larger the cake is. The same can be said for the individual cake, where the cake can have any value, which seems to be a good mix of all of the above cases, as it seems it doesn't matter as much if the perceived value of the cake is shared amongst the agents.

Even though these results initially was quite suprising. It is yet to be proven for

all instances that an indivisible algorithm will maintain its 1/2MMS guarantee. We can however convince ourselves that they make sense.

Let us first assume an instance I, has no items, only a cake C, and the number of agents $n \geq 2$, all the agents MMS allocations, and MMS values for I will simply be their proportional piece of the cake C, which the indivisible algorithm with n pieces of cake can achieve 1/2MMS easily. Now lets add a single indivisible item i to this instance. If $v_a(i) < v_a(C)/n$ then this good does not affect the MMS value of the agent at all, as this good can simply be given to any other agent. And if $v_a(i) \geq v_a(C)/n$ this will increase the MMS value of the agent, but in return this item is now worth more than a piece of cake and such any agent that receives this item does no longer receive any cake and the MMS value of the agent increases to $v_a(C)/()n-1)$. This still isn't a problem for the 1/2MMS guarantee though as $v_a(C)/(n-1) > \frac{1}{2}v_a(C)/n$ as long as $n > 2$. and if the instance only has 2 agents, then the agent that doesn't receive the item can receive more pieces of the cake.

Intuitively it also makes sense as a small and medium cake doesn't necessarily need ot be cut into any pieces as the items will likely be of equal or higher value so the cake doesn't affect the MMS value much. For large cakes we have the opposite that this means each agents gets at least one piece of the cake, and the remaining items are them split such that the 1/2 MMS is still achieved. A more through theoretical analysis and proof of these theorems are needed, but as the results weren't expected there simply wasn't enough time for such an analysis in this project, but this will be a natural part of future work, in combination with generalizing the conepts for heterogenous cake (and by extenesion multiple cakes).

In order to generalize these findings for heterogenous cake, one could possibly utilize what is *Weighted Proportional Cake Cutting* as explained in [3]. This concept generalizes proportionality to the weighted case in cake cutting using a weight profile. This would however require some more pre-processing in order to convert to and from homogenous and heterogenous cakes, which reduces one of the main benefits of simply using a indivisible algorithm directly

Another surpising result is that there seems to be some small subset of variable instances that the mixed algorithm is able to solve better than the indivisible algorithm, despite the indivisible algorithm outperforming for all cake sizes. No further analysis was performer to extract these instances for further analysis as they represented a small percentage of the instances, they are however frequent enough that they should be investigated further.

In Figure 3.6 we also see how the achieved MMS values for both algorithms has

a little spike at $1 - MMS$, this is likely due to the randomly created instances creating instances that are easy to give each agent exactly what they expect. This would be instances where all agents have very similar valuations for the items, which is more likely to happen as the number of goods and agents decrease.

3.2.2 Approximation vs Exact

While the analysis of using $\text{MMS}_a \approx \text{PROP}_a$ to approximate the MMS value suffers from some of the same limitations as the indivisible vs mixed analysis. My results indicate, as expected that for most instances this approximation has no real effect on the resulting allocation. This is especially tru for instances with medium, and large cake, as these instances often have such an abundance of cake that it allows each agent to create an MMS allocation where all bundles have the exact same value, which will be equal to PROP, clearly shown in Figure 1.1.

The use of an exact MMS value is done as per the described algorithm[3], with the uncertainty that my implementation may not be the most techincally efficient, the basic runtime analysis show a massive speedup in computation time, with comparable or mostly equal results to using the exact values.

3.2.3 Runtime Analysis

Since the runtime analysis is fairly limited, I would hesitate to draw any major conclusions from their result. However since the mixed algorithm needs hundreds of times longer per instance depending on the instance than the indivisible (while cutting the cake into n pieces), I would argue that the indivisible algorithm outperforms the mixed algorithm in terms of runtime. However when using the approximation of each agents MMS value instead of the exact one, this runtime decreases drastically and is now faster than the indivisible algorithm. This is again not a completely fair comparison as the indivisible algorithm used in this project also handles constraints, while the mixed algorithm does not.

Chapter 4

Conclusion

The implication of the results in this report is that in order to achieve a higher MMS guarantee in the Mixed Goods setting for homogenous cake, one could possible apply a indivisible algorithm with a higher MMS-guarantee, where the highest MMS approximation guarantee so far is $\frac{3}{4} + \frac{1}{12n}$ [5]. The benefit here being that a very simple algorithm that performs just as well as, or close to, a complex algorithm can be considered a better, or more appliable algorithm.

In the MMS article [3] Section 4.4, improving the MMS-guarantee of the mixed algorithm is considered by using a similar logic. However, their theorem implies a that a MMS-guarantee algorithm for indivisible goods can be used to achieve a approximate similar allocation for mixed instances. My results indicate however that if such an algorithm exists, then this algorithm can be used directly so solve a mixed instance with the same MMS-guarantee. However, any firm conclusions should not be drawn from the experiments presented until proper and general proof is presented.

Not many conclusions can be drawn from the runtime analysis either as this analysis is too limited. The analysis also isn't fair as the indivisible algorithm [6] includes handling of cardinality constraints. For a runtime analysis to be completely fair both/all algorithms should solve the same problem.

4.1 Future Work

For a proper analysis of these results, and for the results to be significant it should be explored wether these results can be applied to heterogenous cake instances as well, in which case these results could indicate that the best approach to finding better MMS-allocation for mixed instances would be to find better algorithms for indivisible instances.

For this to be achieved a few things should be done:

- The algorithms should be tested on a larger number of instances, and with a more varied number of agents and goods.
- Proof of the resulting indications must be provided.
- The algorithms should be tested and applied for instances with heterogenous cake.
- The experiments run in this report should be repeated with indivisible algorithms with higher MMS-guarantees to confirm wether these algorithms are as stable.
- The code used for the experimentation and algorithms must be properly peer-reviewed for quality assurance.

Bibliography

[1] Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A. Voudouris, and Xiaowei Wu. *Fair Division of Indivisible Goods: A Survey.* arXiv, 2022.

[2] Xiaohui Bei, Shengxin Liu, Xinhang Lu, Jinyan Liu, and Zihao Li. *Fair Division of Mixed Divisible and Indivisible Goods.* arXiv, January 2021.

[3] Xiaohui Bei, Shengxin Liu, Xinhang Lu, and Hongao Wang. *Maximin fairness with mixed divisible and indivisible goods.* Springer Science, June 2021.

[4] Umang Bhaskar, A. R. Sricharan, and Rohit Vaish. *On Approximate Envy-Freeness for Indivisible Chores and Mixed Resources.* Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[5] Jugal Garg and Setareh Taki. *An improved approximation algorithm for maximin shares.* arXiv, 2021.

[6] Magnus Lie Hetland and Halvard Hummel. Allocations.jl, 2022. Repository: https://github.com/mlhetland/Allocations.jl.

[7] Rucha Kulkarni, Ruta Mehta, and Setareh Taki. *Indivisible Mixed Manna: On the Computability of MMS+PO Allocations.* arXiv, April 2021.

[8] David Kurokawa, Ariel D. Procaccia, and Junxing Wang. Fair enough: Guaranteeing approximate maximin shares. *J. ACM*, feb 2018.