Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Sivert Utne

Fair Allocation of Mixed Divisible and Indivisible Goods: A Comparative Study

Exploring Naive and Adapted Algorithms

Master's thesis in Computer Science
Supervisor: Magnus Lie Hetland
June 2023

NTNU
Norwegian University of
Science and Technology

Sivert Utne

Fair Allocation of Mixed Divisible and Indivisible Goods: A Comparative Study

Exploring Naive and Adapted Algorithms

Master's thesis in Computer Science
Supervisor: Magnus Lie Hetland
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

This thesis addresses the challenge of fairly allocating mixed divisible and indivisible goods by adapting established algorithms from their respective fields, and creating naive polynomial time algorithms. The objective is to explore the effectiveness of these algorithms in solving the problem of mixed allocation in an attempt to bridge the gap to the more well studied field with only indivisible goods. Empirical experiments were conducted on randomly generated instances, evaluating metrics such as MaxiMinShare and Maximum Nash Welfare. The results demonstrate satisfactory outcomes in a majority of cases, suggesting the potential of adapting indivisible algorithms for mixed allocation problems. However, the scope of these results is limited to the conditions employed in this study, so further theoretical proofs are needed to establish the robustness and applicability of these algorithms across various scenarios. Nevertheless, this research provides valuable insights into the field of fair allocation of mixed goods.

Sammendrag

Denne avhandlingen tar for seg utfordringen med å rettferdig fordele blandete delbare og udelelige varer ved å tilpasse etablerte algoritmer fra deres respektive fagfelt, samt å utvikle enkle polynomiske algoritmer til å finne gode foredlinger. Målet er å utforske effektiviteten og praktikaliteten til disse algoritmene samt koble feltet med blandete varer til det mer velkjente feltet med kun udelelige varer. Det ble utført empiriske eksperimenter på tilfeldig genererte instanser, der rettferdighetsmål som MaxiMinShare og Maximum Nash Welfare ble evaluert. Resultatene viser tilfredsstillende utfall i flertallet av tilfellene, noe som antyder potensialet for å tilpasse udelelige algoritmer til problemer med blandet fordeling. Imidlertid er omfanget av disse resultatene begrenset til betingelsene som ble brukt i denne studien, så ytterligere teoretiske beviser er nødvendig for å etablere robustheten og anvendeligheten til disse algoritmene i ulike scenarier. Likevel gir denne forskningen verdifulle innsikter i feltet for rettferdig fordeling av blandete varer.

Acknowledgements

I would like to express my sincere appreciation to my supervisor Magnus Lie Hetland for his input, guidance, and support throughout the completion of this master's thesis. Further thanks to those who have aided in feedback, structure, language and clarity of the thesis, family and friends.

Preface

This report is a master's thesis written for the course *TDT4900 - Computer Science, Master Thesis* at the Norwegian University of Science and Technology (NTNU). The thesis presents an in-depth study conducted as part of the requirements for completing the master's program in Computer Science at NTNU.

Table of Contents

TABLE OF CONTENTS

Chapter 1

Introduction

Fairness is a concept that is hard to define, and even harder to quantify using values and formulas. The root of this is that fairness is often a complex concept determined by varying subjective opinions from different entities. Because of this, using a computer to find solutions of optimal fairness is difficult.

> According to the Subjective theory of value, there cannot be an objective measure of the value of each item. Therefore, objective fairness is not possible. (Yaari & Bar-Hillel, 1984)

1.1 Fair Allocation and Division

Fair Division and fair allocation is the problem of allocating a set of resources among a set of participants in a way that is considered fair or beneficial by the participants. The term "Allocation" is usually used when the resources to be distributed are indivisible (e.g., paintings, jewelry), and "Division" is used when the resource is divisible (e.g., time, food, land).

For many cases, fairness is a straight forward concept, for instance, when cutting a pizza its understood that a "fair division" is one in which all participants get an equal amount of pizza. The problem gets complex when there is no longer a shared understanding of what the fair division is. And even more so when the resources to be distributed are indivisible, and there is no way for everyone to get an equal share. This is where algorithms for fair allocation come in. They provide various metrics and methods for determining what the "fairest possible" allocation is. Even so, even allocating indivisible goods when the valuations are equal is NP-hard. Because of this algorithms often have to approximate the optimal solution. Fair allocation of cases with both indivisible and divisible resources are a subset of fair allocation problems and is currently a less studied field than divisible and indivisible separately.

1.2 Motivation

Resource allocation is a fundamental challenge across various domains, ranging from economics and finance to healthcare and logistics. The efficient and fair allocation of resources has significant implications for optimizing outcomes and ensuring equity and equality.

The motivation behind this thesis is to analyze algorithms for fairly allocating mixed divisible and indivisible resources. The aim is to contribute to the field of resource allocation by providing novel insights, analysis, practical algorithms, and a deeper understanding of the associated challenges.

Furthermore, the development of efficient algorithms is essential for real-world applications. The analysis and evaluation of different algorithms for allocating mixed resources can help identify the most suitable approaches for specific scenarios. By comparing their performance, scalability, and fairness properties,

practical guidelines for decision-makers and resource allocators can be established.

1.3 Contribution

The objective of this thesis is to offer a comprehensive overview of the current progress in the field of fair allocation scenarios where the goods to be distributed consist of a combination of indivisible items and divisible items.

The main contribituions of the thesis will be to look at proposed ways to adapt algorithms for indivisible goods to allow them to be used for mixed goods. Should the work of adapting the algorithms be beneficial this would allow leveraging established algorithms designed for indivisible goods to directly improve algorithms and work in the field of mixed goods allocation. Moreover, the thesis explores alternative "naive" approaches to find allocations for mixed goods.

1.4 Structure

The theory chapter starts by introducing necessary preliminaries, such as notation and definitions, to establish a common understanding. The existing research and studies related to the problem are summarized, highlighting what has been done before. Furthermore, the limitations of previous work are identified, creating a foundation for the experiments presented in the thesis.

The method chapter describes how the experiments and data are constructed and run for replication and verification. The following two chapters present the specific experiments, their approaches, algorithms and goals.

The results chapter presents the outcomes of these experiments. The discussion chapter interprets the results, shedding light on their meaning and implications, the limitations of the experiments and on how fair allocation can be used in real-life scenarios.

Finally the conclusion summarizes the most important results and findings while acknowledging the limitations in order to provide a foundation for future work as well as contribute to the field of fair allocation.

Chapter 2

Theory

2.1 Preliminaries

Before delving into the details of algorithms and previous work, it is important to establish some common terminology, notation, and definitions. The notation used in this report differs slightly from the more common notations found in the literature. For instance, in the literature, an agent is typically denoted as i, and a good is denoted as j. However, this thesis uses notation that aligns more closely with the terminology rather than mathematical notation. Specifically, agents are denoted as a, and goods are denoted as g. This notation closely resembles the one established in the pre-study project (Utne, 2022).

2.1.1 Definitions

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:

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

$$a \in A$$

Good
A good is any resource that is to be allocated amongst the agents. The term "good" encapsulates both divisible and indivisible resources. Formally:

$$\{1, 2, ..., m\} \equiv G$$
$$g \in G$$

Item - Indivisible Good
An indivisible good will be referred to as an *item*, i, where the number of items $|I|$ defines the set of all items I. Items are goods such as paintings, jewelry, furniture etc. essentially anything that cannot be arbitrarily cut into pieces. Formally:

$$I \subseteq G$$
$$\{1, 2, ..., |I|\} \equiv I$$
$$i \in I$$

Cake - Divisible Good
A divisible good will be referred to as *cake*, C. Cakes can be goods such as money, land, fuel, time etc. In the general literature there is the possibility of instances with several cakes, however for the scope of this report only instances with at most 1 cake is considered.Formally:

$$C \subseteq G$$

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 piece of cake is then defined as the interval of the cake from x to y where $x, y \in [0, 1]$ and $x \leq y$. Formally:

$$c_{x,y}$$

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

$$v_a(c_{x,y}) = (y - x)v_a(C)$$

As this project focuses on homogenous cake, the exact interval of cake pieces is not important, only its size $z = y - x$, where $c_z \equiv c_{x,y} = (y - x)v_a(c)$.

Its important to clarify that each agent can assign different values for the cake. This is important as if all agents have the exact same valuations for the cake, the problem is reduced to a proportional fair division problem that has already been studied. Instances where all agents have the same valuations for the cake are usually instances where the cake is money.

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 $v_a(S) = \int_S d_a dx$. In other words, some parts of cake "taste better" that other parts and are valued higher.

Mixed Goods Instances
An instance with agents and both indivisible *and* divisible goods is referred to as a *mixed instance*, I. Formally:

$$I = G \cup A$$

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 $v_a(\{g\})$ will be written as $v_a(g)$. As all resources are goods, all valuations are positive, formally:

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

Additive valuations
The most studied subclass within fair allocation instances are those with additive valuations. Valuations are additive when an agent's value of any subset of

items is equal to the sum of the values of the individual items in the set. In other terms, the value an agent has for a set of goods cannot decrease if another good is added to it. 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
A collection of goods is often referred to as a bundle, B. The bundle can consist of both items and cake. 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 bundles contains the same item, and that the goods are allocated amongst the agents.

Formally:

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

$$\forall g \in G \sum_{a \in A} B_a(g) = 1$$

Oracle
Some algorithms rely on what they call a *oracle*. An oracle is simply a function that can be used to find any required value needed by the algorithm. This means the algorithm might not have direct access to all valuations for instance, but it can access it when needed.

2.1.2 Fairness Notions

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.

One could imagine two siblings sharing a pizza, where sibling 1 cuts the pizza, and sibling 2 gets to choose the piece they want. Sibling 1 will then naturally attempt to cut the pizza such that no matter which piece sibling 2 chooses, sibling 1 gets as good a piece as possible. Since sibling 2 will choose the best piece for themselves, MMS must be less than or equal to PROP.

Pareto-Optimality (PO)
A Pareto-Optimal allocation is an allocation where no agent can be made better off without making another agent worse off.

Nash Social Welfare (NSW)
The Nash Social Welfare is a measure of the social welfare of an allocation $\mathcal{A}$. It is defined as the product of the value each agent assign their bundle. Formally:

$$\text{NSW}(\mathcal{A}) = \prod_{a \in A} v_a(B_a)$$

A desireable fairness allocation is that which maximizes the nash social welfare, Maximum-Nash-Welfare (MNW). A MNW allocation will always be Pareto-Optimal.

Envy-Freeness
Envy-freeness encompasses several fairness notion that is prominent in the literature. This project does not directly apply any of these fairness notions, but they are included for completeness due to their relevance in the literature.

EF
An allocation $\mathcal{A}$ is said to be envy-free (EF) if for every pair of agents $a_1, a_2 \in A$,

$v_{a_1}(B_1) \geq v_{a_1}(B_2)$. 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 bundle is missing one item.

EFx
An allocation $\mathcal{A}$ is said to be envy-free up to any item (EFx) if for every pair of agents $a_1, a_2 \in A$ and any good $g \in B_{a_1} \cup B_{a_2}$, $v_{a_1}(Ba_1 \backslash \{g\}) \geq v_{a_1}(B_{a_2} \backslash \{g\})$. EF1, but the good g can be the "worst" or "smallest" good in the other agents bundle.

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 the definition it is true 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.

2.1.3 Mixed Integer Programming

Mixed Integer Programming is a mathematical programming technique used to solve optimization problems that involve both discrete (integer) and continuous variables. It is a subfield of mathematical optimization. The objective is to find the optimal solution that maximizes or minimizes an objective function, subject to a set of constraints. The constraints can involve linear equations or inequalities.

MIP is useful in a wide range of applications where decisions need to be made

under constraints. Some common applications include:

- Resource allocation: MIP can be used to optimize the allocation of limited resources, such as assigning tasks to workers or scheduling production.
- Supply chain management: MIP can optimize the flow of goods and materials in a supply chain, considering factors like transportation, inventory management, and production planning. Facility location: MIP can help determine the optimal locations for facilities, such as warehouses, factories, or distribution centers, considering factors like transportation costs and customer demand.
- Network design: MIP can be used to optimize the design and configuration of networks, such as telecommunication networks or transportation networks.
- Project scheduling: MIP can optimize the scheduling of activities in a project, considering factors like task dependencies, resource availability, and time constraints.

Overall, MIP provides a powerful framework for modeling and solving optimization problems that involve both discrete and continuous decision variables, making it valuable in various industries and domains.

Mixed Integer Programming does not always guarantee finding the optimal solution. The goal is to find the best feasible solution that optimizes a given objective function. The complexity of MIP problems increases as the problem size grows, and the time required to find the optimal solution can become prohibitively large for large-scale instances. In practice, MIP solvers employ various techniques, such as branch and bound, cutting planes, and heuristics, to efficiently explore the solution space and search for the optimal solution.

However, due to the complexity of certain problems or time limitations, MIP solvers may terminate before finding the global optimum. In such cases, they provide a feasible solution that may or may not be optimal. It is also possible that a MIP problem is inherently difficult, and finding the optimal solution is computationally infeasible within a reasonable time. Nevertheless, MIP solvers often find near-optimal or good-quality solutions for a broad range of practical optimization problems.

2.2 Previous Work

This section provides a concise overview of the pertinent literature and research conducted in the domain of fair division and allocation, setting the foundation for the subsequent analysis.

Fair division and allocation have garnered considerable attention in both theoretical and practical settings. Scholars and practitioners have extensively explored various approaches, models, and algorithms to address the challenge of distributing resources or goods among multiple participants in a fair and equitable manner. This literature review aims to highlight key contributions and advancements made in this field.

2.2.1 Fair Division

In (Aumann, Dombb, & Hassidim, 2012) it was found that a common task faced by MAS designers is the efficient allocation of resources among multiple agents. The focus of their study was on a scenario where a single divisible resource, referred to as a "cake," needed to be divided among n agents, each having a potentially different valuation function for different cake portions. They addressed the problem of finding divisions that maximize social welfare, with a specific emphasis on divisions where each agent receives a single contiguous piece of the cake.

The researchers presented an approximation algorithm for the problem and demonstrated that it achieves a constant factor approximation. Moreover, they showed that finding the optimal division is NP-hard. These findings held true both when the algorithm had complete knowledge of all agents' valuations and when it had only oracle access to their valuation functions.

In contrast, the results varied depending on whether agents were allowed to receive multiple non-contiguous pieces of the cake. If the algorithm had complete valuation functions of all agents, the problem was found to be easily solvable. However, if the algorithm needed to query agents for their valuations, it was shown that no non-trivial approximation (i.e., approximation factor less than n) could be guaranteed.

2.2.2 Fair Allocation

(Woeginger, 1997) considers the problem of assigning a set of n jobs to a system of m identical parallel machines so as to maximize the earliest machine completion time to reduce machine idle time. The approach can be adapted to finding a maximin allocation for instances with indivisible goods when all valuations are identical between the agents. In other words find a agents maximinshare. This is explained more closely in Listing 4.1.

(Amanatidis, Markakis, Nikzad, & Saberi, 2017) performed a probabilistic analysis of MMS-Allocations. They found that in randomly generated instances, maximin share allocations exist with high probability. This provided a justification of previously reported experimental evidence. Finally, they provided further positive results for two special cases arising from previous works. In the first case of three agents, they provided an improved 7/8-approximation. In the second case where all item values belong to 0, 1, 2, they obtained an exact algorithm.

(Garg & Taki., 2021) proposes a new approach for finding a 3/4-MMS allocation. The paper shows that this approach is also powerful enough to be extended in two directions: first, to find a strongly polynomial-time algorithm for computing a 3/4-MMS allocation without approximating the MMS values, and second, to prove the existence of a (3/4 + 1/12n)-MMS allocation, which improves the previous best factor for small n. This is an important result in the field of fair division and contributes to the understanding of how to achieve fair allocations in various settings.

(Barman, Biswas, Krishnamurthy, & Narahari, 2018) investigate the concept of maximin shares (MMS). They find however, that MMS has its limitations and does not necessarily lead to satisfactory outcomes. In this paper, the authors introduce a stronger notion of fairness called groupwise maximin share guarantee (GMMS), which extends the concept ofMMS to subgroups of agents. Under GMMS, the maximin share guarantee must be achieved not only for the overall allocation but also for each subgroup of agents. The authors prove the existence of GMMS allocations in specific settings and show that GMMS implies approximate envy-freeness. They also develop a polynomial-time algorithm to find approximate GMMS allocations under additive valuations and demonstrate their results empirically on a large set of randomly generated instances. Overall, this work provides a more robust and comprehensive notion of fairness for

resource allocation problems.

(Suksompong, 2018) extend the study of the maximin share fairness notion to the setting where goods are allocated to groups of agents. They assume that the agents within each group share the same set of goods, but may have conflicting preferences over them. The authors consider the case of two groups and investigate the cardinality of the groups for which a positive approximation of the maximin share is possible regardless of the number of goods. They show that when each group has at least three agents, a positive approximation of the maximin share is always possible, regardless of the number of goods. When one group has two agents and the other has at least four, a positive approximation of the maximin share is also possible. However, when both groups have only two agents, it is not always possible to achieve a positive approximation of the maximin share. The authors also consider settings with more than two groups and show that in some cases, a positive approximation of the maximin share is possible, while in other cases it is not. They provide specific examples of such settings. Overall, the paper provides insights into the maximin share fairness notion in the group allocation setting and highlights some of its limitations and challenges.

(Caragiannis et al., 2019) found that their paper's assumptions about goods being indivisible can be extended to cases where goods are both divisible and indivisible. In such cases, the MNW solution can be seen as the limit of the MNW solution on an instance where each divisible good is partitioned into k indivisible goods, as k goes to infinity. According to Theorem 3.2, this implies that the MNW solution is envy-free up to one indivisible good, meaning that a player would not envy another player who has both divisible and indivisible goods if one indivisible good is removed from the bundle of the other player. This provides an alternative proof for envy-freeness of the MNW/CEEI solution when all goods are divisible. The results of Section 4 also hold in this case, and the proof of the MMS approximation result (Theorem 4.1) already uses the liquidation of some goods as a technical tool. In the full version, they outline a modified and scalable version of the implementation described in Section 5 that can allocate a mix of divisible and indivisible goods, which they deployed on *spliddit.org* (see Section 2.2.4).

They also observed that when all goods are divisible, the MNW solution, the CEEI solution, and proportional fairness (PF) coincide, whereas this is not the case for indivisible goods. However, their investigation showed that the PF

solution and the MNW solution are closely related via a spectrum of solutions, which offers two advantages. Firstly, it allows them to view the MNW solution as the optimal solution among those that lie on this spectrum and are guaranteed to exist. Secondly, it gives a way to break ties among all MNW allocations, and possibly even choose a unique allocation. The full version provides a detailed analysis of this connection between MNW and PF, and raises an interesting question about whether it is possible to relate the MNW solution to the CEEI solution when the goods are indivisible.

In previous studies, it was demonstrated that the existence of anMMS(Minimax Share) allocation might be impossible, with a counterexample requiring an exponential number of goods relative to the number of players. However, (Kurokawa, Procaccia, & Wang, 2016) presents a novel approach that achieves anMMSallocation using only a linear number of goods. They also provide a formalization of the notion that these counterexamples are highly intricate. They accomplish this by developing an algorithm that can reliably identify anMMSallocation with a high probability, particularly when valuations are randomly generated.

2.2.3 Mixed Goods

(Bei, Li, Liu, Liu, & Lu, 2021) established that traditional notions of fairness, such as envy-freeness (EF) and envy-freeness up to one good (EF1), are not directly applicable to the mixed goods setting. To address this, they proposed a new fairness concept called envy-freeness for mixed goods (EFM), which is a direct extension of both EF and EF1 to the mixed goods scenario. They proved that an EFM allocation always exists for any number of agents with additive valuations. Additionally, they proposed efficient algorithms to compute an EFM allocation for two agents with general additive valuations and for n agents with piecewise linear valuations over divisible goods. Finally, they relaxed the envy-freeness requirement and presented an efficient algorithm to find a δ-EFM allocation.

They also given a polynomial-time algorithm for finding a EFM allocation for instances with 2 agents, along with an approach to achieve a more strict version of EFM where each agents has EFx to any agent with only indivisible goods, and EF to any agent with cake. In order to accomplish this a EFx algorithm for the indivisible goods is required which may not always exist. They also find

that for any number of agents with piecewise linear density functions over the cake, an EFM allocation can be computed in polynomial time.(Amanatidis et al., 2022)

(Bei, Liu, Lu, & Wang, 2021) Describe an algorithm that guarantees to find a 1/2-MMS Allocation for instances with mixed goods, my specialization project (Utne, 2022) showed that for such low guarantees, cutting the divisible resource into n indivisible pieces was enough to guarantee the same 1/2-MMS Allocations for all instances. They do however describe how their approach allows for an approximation ratio of maxα, β - ϵ, which can be arbitrarily close to the currently best-known ratio of 3/4 + 1/12n for indivisible goods (Garg & Taki., 2021). Suppose there exists a polynomial-time algorithm that guarantees to output a β-MMS allocation with indivisible goods for some β. Then given a mixed good problem instance, first compute α' via Theorem 3 and compare it with β : if $\alpha' \geq \beta$, directly apply Theorem 3; otherwise, cut the cake C into small intervals, each valued at most $\epsilon \cdot u_i(C)/2n$ for each agent i, and use the β-MMS algorithm to obtain the allocation of this instance with only indivisible goods.

(Bhaskar, Sricharan, & Vaish, 2022) found that their results are applicable to a mixed resources model, which includes indivisible items and a divisible, undesirable heterogeneous resource, commonly referred to as a "bad cake." They demonstrated that there is always an allocation that meets the envy-freeness for mixed resources (EFM) requirement in this setting.

(Nishimura & Sumita, 2023) found that an MNW allocation for mixed goods is always envy-free up to one indivisible good (EF1M) and Pareto-optimal (PO). They also noted the existence of an EF1M and PO allocation and provided insights into the computation of an EF1M allocation. They demonstrate that although an EF1M allocation can be found in finite steps, an EF1M allocation is less fair than an MNW allocation. With the example of 2 agents with 1 cake and 1 item, the EF1M algorithm they propose gives half the cake to each and the item to one agent, which achieves EF1M, however the MNW, and arguably most fair allocation is that one agent gets the item, and the other agent gets the cake. Work still needs to be done in constructing an algorithm to find an MNW allocation for mixed goods.

In the prestudy conducted for this thesis (Utne, 2022), the approach of utilizing established algorithms for indivisible goods was explored and found to be applicable to instances involving mixed goods with only minor adjustments.

Specifically, an algorithm for determining the 1/2-maximin-share (1/2-MMS-share) for mixed goods, as introduced in (Bei, Liu, et al., 2021), was compared against a naive approach of dividing the cake into n equal pieces.

Surprisingly, the naive approach successfully maintained the minimum envy-free allocation (MMS-guarantee) for all instances, implying that adopting this approach and employing an algorithm with a higher MMS-guarantee could directly translate into a higher MMS-guarantee for mixed goods. This observation suggests the potential for leveraging existing algorithms for indivisible goods in the domain of mixed goods, as demonstrated in the prestudy (Utne, 2022).

2.2.4 Spliddit.org

Spliddit.org was a notable website that provided users with the ability to create real-life instances and obtain fair division solutions. Regrettably, as of the time of writing, the website has been unavailable for several months without any updates communicated through their public channels. Spliddit.org was developed by a group of researchers from Carnegie Mellon University (Spice, 2014).

According to available sources, Spliddit offered provably fair solutions for various everyday fair division problems, ranging from rent splitting and goods division to credit sharing (Ariel Procaccia and Nisarg Shah, 2023). The website gained significant popularity, indicating the practical applicability and demand for these algorithms and approaches in real-world scenarios.

> Spliddit received nearly 40,000 unique visitors, who combined to use the three launch application (Sharing Rent, Dividing Goods, and Assigning Credit) over 9,000 times. Feedback has been overwhelmingly positive, with many users commending the interfaces, ease-of-use, and overall mission of Spliddit. (Goldman & Procaccia, 2015)

The absence of the website is unfortunate, especially in the context of this thesis, as one of the intentions was to compare the results of the algorithms implemented here with those generated by the website's algorithms. Furthermore, the unavailability of a platform that allows users to easily create and solve their own fair division problems is a notable loss.

2.3 Unsolved Problems

The existing literature indicates that the research on mixed goods is not as extensive as that on indivisible goods. Within this context, there are two main unsolved problems and interesting areas that this thesis aims to address.

2.3.1 Finding MaxiMinShare Allocations

It is frustrating to observe that even for instances with equal valuations, finding a Maximin allocation is NP-Hard, and in some cases, it may not even exist. However, for mixed goods, one would intuitively assume that a Maximin allocation always exists, given a certain size of the cake. This assumption arises from the fact that a proportional (PROP) allocation always exists for any cake. This will be explored in Chapter 4.

2.3.2 Finding Maximum Nash Welfare

An allocation that achieves the maximum Nash Welfare (MNW) possesses desirable properties in terms of fairness. Therefore, it is crucial to advance the understanding of finding MNW allocations for mixed goods. This topic was explored in-depth in (Nishimura & Sumita, 2023) where making progress towards finding such an algorithm was out of their scope but will be explored in Chapter 5.

Chapter 3

Method

This chapter presents the overall methodology and decisions made for conducting the experiments. The specific methods and approaches, tailored to each fairness notion, have been outlined in separate chapters to address the unique requirements of each notion. By organizing the content in this manner the methodology aligns with the specific objectives and considerations associated with each fairness notion.

3.1 Choosing Technologies

3.1.1 Language

Julia was selected as the language in large part due to the Allocations.jl package Section 3.1.2. Furthermore Julia provides a powerful and expressive programming language specifically designed for scientific computing and numerical analysis.

Furthermore, Julia offers exceptional performance. It leverages a just-in-time compilation approach that dynamically optimizes code execution, often achiev-

ing performance comparable to statically compiled languages like C or Fortran. This makes Julia well-suited for computationally intensive tasks, such as simulations, data analysis, and mathematical modeling. It has a built-in read-eval-print loop (REPL) that allows for quick prototyping and interactive experimentation. This interactive workflow is particularly valuable for scientific research, as it facilitates iterative development, rapid experimentation, and data exploration.

3.1.2 Allocations.jl

The Julia package, *Allocations* (Hetland & Hummel, 2022), provided ready-made implementation of various allocation algorithms for fair allocation problems, both with and without constraints. Leveraging this package greatly facilitated the experimentation process, saving significant time that would have otherwise been spent on implementing algorithms from scratch. It is important to note that Allocations.jl is specifically designed for scenarios involving indivisible goods, aligning with the scope of the experiments where the cake was divided into indivisible pieces. To adapt the package for mixed instances, a straightforward translation was performed to transform the mixed instances into indivisible ones, and vice versa for the allocation results. This seamless integration allows exploration of different algorithms and comparison of their performance efficiently.

3.1.3 HiGHS

HiGHS (high performance software for linear optimization) is an open-source solver that specializes in solving linear programming (LP) and mixed integer programming (MIP) problems. It features high-performance algorithms, including an interior point method for LP and a branch-and-bound algorithm for MIP. It is written in C++ and is available as a library for C, C#, FORTRAN, Julia and Python(Hall, Galabova, Feldmeier, & Zanetti, 2023).

3.1.4 Hardware

Efficiency and performance are crucial factors in conducting experiments. All analysis and experiments in this master's thesis were performed on a single machine equipped with the following hardware configuration:

- **CPU**: 2,2 GHz 6-Core Intel Core i7
- **RAM**: 16 GB 2400 MHz DDR4
- **GPU**: Radeon Pro 555X 4 GB, Intel UHD Graphics 630 1536 MB
- **OS**: MacOS Ventura

3.2 Creating Instances

When creating instances that aim to represent real-world scenarios, an approach was adopted to generate them using random valuations while imposing certain constraints. Firstly, the valuations assigned by each agent to an item are confined to the interval $[0, 1]$. This decision is based on the understanding that, for most algorithms, the relative values of items for each agent are more significant than the specific numerical values. Thus, in all generated instances, the following condition holds:

$$\forall a \in A, \forall i \in I, 0 \leq v_a(i) \leq 1$$

For the experiments creating the instances in this way ensures that although the instances are randomly generated, they are guaranteed to be different types of likely real world instances.

3.2.1 Number of Agents

When choosing how many agents to include in the instances, it is important to consider the computational complexity of the algorithms. The number of agents was then kept relatively small with a maximum number of agents of 11.

Instances with 2 agents will not be examined as there is already established algorithms for finding this already. To find a 1-MMS allocation with 2 agents

simply let 1 agent create 2 bundles of the goods such that the smallest bundle is as large as possible. Then simply let the other agent choose whichever bundle they consider most valuable, since they chose the best bundle, they receive at least theirMMSas the other bundle can contain at most PROP. Instances with less than 2 agents are of course also excluded as they are trivial cases.

3.2.2 Number of Goods

The number of goods also impact the relevancy and complexity of the algorithms. For this reason a number of goods wsa chosen to always be more than the number of agents, to prevent trivial cases of assigning a single good to each agent, or agents not being able to receive anything at all. Furthermore than number of goods was chosen to be both odd and even to ensure a wider range of different possibilities.

The number of goods for each instance was the decided to start at $n + 1$ and increment upwards to ensure that both odd and even numbers of goods were used for each number of agents. Additionally some experiment increment by 3 each time to cover a wider range in a shorter amount of time.

3.2.3 The Cake

The cake is a vital part of the mixed goods instance. In that its perceived size has a significant impact on what a fair allocation looks like. For instance if the cake has value ≈ 0, then the instance can be seen as a indivisible goods as the cake is so small its not worth splitting. On the other hand if the cake has value far greater than the other goods, fair division of the cake becomes a lot more important than fairly allocating the items.

To account for this the generated instances are categorized into three distinct types.

- Small cake: All agents valuations for the cake is in the same order of magnitude as their valuations for the other items. Formally:

$$v_a(C) \leq \max v_a(I)$$

- Large cake: All agents assign a value for the cake that is significantly larger

than their valuations for the other items. Formally:

$$\max v_a(I) \leq v_a(C) \leq \max v_a(I) \cdot \left(1 + \sum v_a(I)\right)$$

- Random cake: For each agent, their value for the cake is randomly chosen to be Large or small.

Combined these three cases should ensure that the generated instances cover a wide range of different scenarios.

3.2.4 Normalizing an Instance

Normalization of agents' valuations is a valuable technique that enhances the efficiency and effectiveness of fairness algorithms. This process can be likened to an analogy from Season 4, Episode 10 of the television series "Succession." In this episode, the characters are given the same amount of sticker that they can place on items they desire, with the number of stickers indicating their level of preference in a action.

By normalizing agents' valuations, each agent has an equal sum of valuations (equivalent to the total number of stickers). An additional advantage of normalization is its flexibility, as the specific normalization scheme can be adjusted depending on the algorithm's requirements. For example, the platform *Spliddit.org* (refer to Section 2.2.4) enabled users to distribute a total of 1000 points among the items. This distribution can be achieved simply by normalizing the agents' valuations, ensuring that the sum of their normalized valuations equals 1000.

In addition to its algorithmic benefits, it is important to note that normalization of valuations can have implications for certain fairness notions. For example, scaling the valuations can alter the Nash Social Welfare (NSW) value. On the other hand, fairness notions such as MaxiMinShare (MMS) are scale invariant, meaning that the normalization process does not affect the MMS value.

To address this, one approach is to maintain the original instance data while using the normalized instance solely for the algorithmic calculations. By separating the normalized instance from the original one, any modifications or transformations made to the normalized instance do not impact the original

instance. This allows for a clear distinction between the normalization process used for algorithmic efficiency and the underlying fairness properties of the original valuations and the resulting allocation.

To normalize an instance, the following formula is used to achieve a valuations sum of δ:

$$\forall g \in G, a \in A, v_a(g) = \delta \cdot \frac{v_{a(g)}}{\sum_{g^* \in G} v_a(g^*)}$$

3.3 Experiment Pipeline

In order to streamline the extensive analysis of algorithms and reduce the time required for experimentation, a well-defined experimentation pipeline was established. This pipeline is divided into three distinct parts, with each part generating specific outputs that are saved to files. To ensure proper management and tracking of these files, a version control system using Git was implemented. This system enables safe and efficient tracking of any changes, overrides, or accidental deletions that may occur during the experimentation process.

3.3.1 Generating Instances and Finding Allocations

The first and crucial part of the pipeline involves generating instances and finding suitable allocations for these instances. This step forms the foundation for the subsequent analysis and evaluation. Special attention was given to ensure that the algorithms being compared utilized the exact same instances during the experimentation process. The allocations, along with their instances, are serialized and saved to files before they are analyzed.

3.3.2 Analyzing the Allocations

Depending on the specific experiment being conducted, various analyses are performed on the generated allocations. These analyses involve extracting and processing the data obtained from the previous step, calculating relevant metrics and fairness measures such as the achieved $\alpha - \text{MMS}$ value and/or the NSW

of the allocation. The results of these analyses are then saved to a CSV file for plotting, again ensuring that relevant data organized such that it enables comparison parity.

3.3.3 Plotting the Results

Once the allocations have been analyzed and the relevant data has been saved to a CSV file, the final step involves adjusting and plotting the results. This is accomplished by reading the saved file and extracting the necessary data points. This approach allows for quick and straightforward adjustments to the plots without the risk of accidentally modifying the underlying data.

Visualization of Instances and Allocations

To facilitate a better understanding of the instances and allocations, visualization of the instances and allocations is necessary. This visualization needed to be as intuitive and human-readable as possible without loosing to much information.

For instances, the tool displays each agent's valuations for all goods, along with the agent's mixed MaxiMinShare (MMS) value, which is calculated using a HiGHS solver. On the other hand, for allocations, the tool provides information about the amount of each good allocated to each agent, the value assigned by each agent to their bundle, and the achieved $\alpha - \text{MMS}$ value, as well as the smallest achieved MMS and Nash Social Welfare (NSW) values of the allocation.

An example of the visualization output is shown below:

```
Instance with RANDOM cake:
  Goods:    Item  Item  Item  Cake
  Agent 1: [0.46, 0.94, 0.75, 0.85] - Mixed MMS: 1.00
  Agent 2: [0.55, 0.03, 0.98, 1.34] - Mixed MMS: 0.96
  Agent 3: [0.77, 0.75, 0.09, 1.04] - Mixed MMS: 0.88

Allocation with Maximum Nash Welfare Algorithm:
  - cake is cut into 5 indivisible pieces.

2600.0 ms: MMS=1.11, NSW=1.26
  Agent 1: [    , 1.00,     , 0.20] - Bundle Value: 1.11
  Agent 2: [    ,     , 1.00, 0.40] - Bundle Value: 1.51
  Agent 3: [1.00,     ,     , 0.40] - Bundle Value: 1.19
```

Chapter 4

Adapting Indivisible Algorithms for Mixed Goods

4.1 Equal Valuations

From the definition of MMS, it is easy to see why an MMS-allocation always exists for instances with identical valuations. It is hence desirable to establish a new algorithm that finds such an allocation for mixed instances. The idea for this is simple; find a 1-MMS algorithm for the indivisible items, and then give cake incrementally to the agent(s) with the lowest valued bundles. However, finding a maximin allocation for indivisible items is NP-Hard. The idea is then to use the cake to establish instances where a 1-MMS allocation can be found in polynomial time.

To establish an algorithm that accomplishes this, the algorithms bases itself on LPT. An algorithm designed to maximize the minimum completion time of a set of jobs on a set of identical machines. In other words, maximize the smallest bundle in an instance with identical valuations. The algorithm is described as follows:

> Longest Processing Time Heuristic, **LPT**. The LPT heuristic sorts all jobs into a nonincreasing sequence and then sequentially assigns each job to the next machine available. LPT will always remain within a factor of $(3m-1)/(4m-2)$ of the optimum solution where m is the number of jobs, and this bound is tight (Woeginger, 1997).

Algorithm 1 is an attempt to find an efficient 1-MMS allocation algorithm for mixed instances. Distinguishing each agents valuations is not necessary as they are equal and $v_{\forall a \in A}(g)$ is then simplified to $V(g)$.

Algorithm 1 LPT Algorithm for Mixed Instances with Identical Valuations

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

1: $B_1, B_2, ..., B_n \leftarrow \emptyset$
2: **while** $I \neq \emptyset$ **do** ▷ Phase 1: Allocate all items with LPT
3: $a^* \leftarrow \arg\min_{a \in A} V(B_a)$
4: $i^* \leftarrow \arg\max_{i \in I} V(i)$
5: $B_{a^*} \leftarrow B_{a^*} \cup i^*$
6: $I \leftarrow I \backslash \{i^*\}$
7: **end while**
8: **while** There is any unallocated cake: C **do** ▷ Phase 2: Allocate cake
9: $B_1 \leftarrow \min_{a \in A} V(B_a)$
10: $A^* \leftarrow \forall a \in A, V(B_a) = V(B_1)$
11: $B_2 \leftarrow \min_{a \in A \backslash A^*} V(B_a)$
12: $c \leftarrow (V(B_2) - V(B_1)) / V(C)$ ▷ Cake needed for $B_1 = B2$
13: **if** $c < (|A^*| \cdot c)$ **then**
14: $c \leftarrow C/|A^*|$
15: **end if**
16: **for** $a \in A^*$ **do**
17: $B_a \leftarrow B_a \cup \{c\}$
18: $C \leftarrow C \backslash \{c\}$
19: **end for**
20: **end while**
21: **return** $\{B_1, B_2, ..., B_n\}$

The algorithms follows a naive approach in 2 phases. Phase 1 (Line 2, use the LPT algorithm for the items (see Section 4.1)) where the agent with the

currently lowest valued bundle is given the currently highest valued item, the item is then removed from the pool of available items. In Phase 2 (Line 8) the agent(s) with the lowest valued bundles are found, then how much cake they need for their bundles to equal the next lowest bundle are calculated. This is then repeated until there is not enough cake left to satisfy the next lowest bundle, at which point the remaining cake is divided equally amongst the agents with the lowest valued bundles. If all agents have the exact same bundle value, then the second best bundle has infinite value, and the remaining cake is allocated to all the agents.

The algorithm then strictly increase the guarantee from LPT, Since we need a miniumum of i.e. it has a MMS-guarantee of:

$$\frac{3|I|-1}{4|I|-2} + V(C)/n$$

From this, it follows that the algorithm is able to find a 1-MMS allocation for all instances with identical valuations as long as:

$$V(C) \geq \left(1 - \frac{3|I|-1}{4|I|-2}\right) \cdot \sum_{i \in I} V(i)$$

This stands in contrast to finding a $1-$MMS allocation for only indivisible items, which is NP-Hard.

The equation above can be generalized to the following, where an algorithm with a better bound than LPT can be used to find a 1-MMS allocation when the cake is even smaller.

> For an algorithm that guarantees to find a $\alpha-$MMS allocation for indivisible items with equal valuations. That algorithm can be used to find a $1-$MMS allocation for mixed goods, as long as:
>
> $$V(C) \geq (1-\alpha) \cdot \sum_{i \in I} V(i)$$

Simply use the algorithm on the indivisible items, achieving at least $\alpha-$MMS

and then continually give cake to the agent(s) with the lowest valued bundles until the cake is gone.

In order to verify this experimentilly, instances will be generated as follows:

1. Instances are generated as described in Chapter 3.
2. All agents valuations are made equal to the first agents valuations.
3. The valuation of the cake is adjusted to the exact lower bound the algorithm requires based on the valuation of the items.

These modifications make sure that the experiments will maintain the desired randomness and coverage. Adjusting the cake size to the minimum required by the algorithm ensures that, if for any instance the algorithm does not actually find a 1-MMS allocation, then the algorithm has failed. On the contrary if all instances are solved, this indicates that the guarantee is correct. Any instance that has a cake of a larger size than the minimum requirement will simply split the "additional" cake amongst all the agents in the instance.

For completness the constraints of a mixed instance for a MIP model is shown in Listing 4.1, and the additional constraints and objective for the MIP model to achieve maximin is shown in Listing 4.2.

Listing 4.1 MIP model for instances with mixed goods

```
function init_mip_mixed(instance)
    model = JuMP.Model(HiGHS.Optimizer)
    # variable is n * m matrix
    JuMP.@variable(
        model,
        A[Agents(instance), Goods(instance)],
        lower_bound = 0.0,
        upper_bound = 1.0
    )
    # all goods must be assigned amongst the agents
    for good in Goods(instance)
        JuMP.@constraint(
            model,
            sum(A[agent, good] for agent in Agents(instance)) == 1
        )
    end
    # items can only be assigned to a single agent
    for item in Items(instance), agent in Agents(instance)
        JuMP.set_binary(A[agent, item])
    end
    return context
end
```

Listing 4.2 MIP constraints and objective for finding maximin allocation. This is done after creating the model and adding mixed constraints as shown in Listing 4.1.

```
JuMP.@variable(model, min_bundle_value)
for agent in Agents(instance)
    JuMP.@constraint(
        model,
        min_bundle_value <= sum(
            A[agent, good] * instance.valuations[agent, good]
            for good in Goods(instance)
        )
    )
end
JuMP.@objective(model, Max, v_min)
```

4.2 Individual Valuations

4.2.1 Cutting Cake into n pieces

The results from the pre-study revealed that simply cutting cake into n pieces always allowed a 1/2-MMS indivisible algorithm to achieve its guarantee for the mixed instance. AS the algorithm used to test this had such a low MMS-guarantee this result should be verified by using an algorithm with a higher MMS-guarantee.

For this a 2/3-MMS algorithm will be utilized in exactly the same way. The cake will be cut into n pieces, and the algorithm will be run on this "indivisible" instance.

4.2.2 Allocating Items First

Algorithm 1 cannot be directly applied for the more general instance where each agent has their own individual valuations for each good.

To see how viable the thought process is for the more general case, a slightly altered version of Algorithm 1 is proposed.

Algorithm 2 LPT Algorithm for Mixed Instance with Identical Valuations

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

$B_1, B_2, ..., B_n \leftarrow 2/3 - MMS(\mathcal{A}_I)$ ▷ Phase 1: Allocate Items
while There is any unallocated cake: C **do** ▷ Phase 2: Allocate cake
  $B_1 \leftarrow \min_{a \in A} v_a(B_a)$
  $A^* \leftarrow \forall a \in A, v_a(B_a) = v_a(B_1)$
  $B_2 \leftarrow \min_{a \in A \setminus A^*} v_a(B_a)$
  $c \leftarrow (v_a(B_2) - v_a(B_1)) / v_a(C)$ ▷ Cake needed for $B_1 = B2$
  if $c < (|A^*| \cdot c)$ **then**
    $c \leftarrow C/|A^*|$
  end if
  for $a \in A^*$ **do**
    $B_a \leftarrow B_a \cup \{c\}$
    $C \leftarrow C \setminus \{c\}$
  end for
end while
return $\{B_1, B_2, ..., B_n\}$

The functionality is very similar. However Phase 1 (Algorithm 1) now consist of using a 2/3-MMS algorithm for indivisible goods to allocate the items first. Then in Phase 2 (Algorithm 2) the cake is given out to the agents in exactly the same way as in Algorithm 1, however, now the agents individual valuations are of course used as they are not guaranteed to be equal.

Furthermore the instance is normalized before the allocation starts in an attempt to make the valuations between the agents more comparable with each other. Otherwise it might be the case that one agent assigned all goods a value of 1, and another all goods a value of 0, in which case the agent that gives everything value zero would get the entire cake no matter what, as their bundle is always the smallest, and the cake doesn't increase it. After normalizing both these agents will have assigned 0.5 to all the goods.

Another potential issue is that a 1-MMS allocation is not guaranteed to exist. In rare, and highly contrived cases, finding a 1-MMS allocation is not possible with indivisible goods. The same goes for indivisible goods. If the value of the cake is very small ≈ 0, and it impossible to allocate the items on their own to 1-MMS, then finding 1-MMS for the mixed instance is also impossible. However

based on how rare, and how delicate instances like these are, it is not expected to be a problem in practice, in fact it was found to not occur randomly(Kurokawa et al., 2016).

4.3 Improving Efficiency

An alternative to using the cake to improve the fairness of an allocation, an idea is to leverage the cake's size to relax the requirements for solvers when searching for a maximin allocation for the items. The allocation process can still be performed as before, ensuring the algorithm's guarantee while potentially reducing the time needed to find such an allocation.

Chapter 5

Maximum Nash-Welfare for Mixed Goods

As described in (Nishimura & Sumita, 2023), Maximum Nash welfare is a very desirable allocation for mixed goods as it achieves several other fairness notions such as EF1 and Pareto-optimality "passively".

5.1 The Problem

No Maximum Nash Welfare algorithms has been established for mixed goods. One of the reasons for this is that the current solution for finding a approximate-MNW allocation for indivisible goods is using a linear solver using logarithmic expressions for integer valuations. In order to find a solver that finds an exact MNW for mixed goods this would require a solver that can handle polynomial expressions for non-integer values, because the cake needs to be able to be cut into arbitrary pieces in order to be truly 'divisible'.

A proposed solution for adapting the indivisible MNW algorithm to the mixed goods instance is to see the MNW of the mixed goods instance as the limit

as the the cake is cut into infinitely many indivisible pieces(Caragiannis et al., 2019). However, even for indivisible instances with only 5 agents they themselves experienced that finding an allocation took over their set time limit of 60 seconds. Increasing the size and number of variables in the instances to infinity does then not seem like a viable option.

5.2 The Solution

In order to shed some light on this and prove this experimentally, the described experiment will be performed. A randomly created instance will be taken, and the cake will be incrementally cut into more and more pieces. This way, the NSW is likely to change as the number of pieces increases, and measurements can be taken to determine how the increasing number of pieces impacts the time taken to find the allocations.

There is, however, one hiccup. The MNW algorithm with the MIP only works for integer valuations. The reason for this is that calculating the NSW of an allocation requires multiplication, which is not possible in linear programs (hence the term "linear"). To be able to cut the cake into an arbitrary number of pieces, it is necessary to establish how the integer valuations can be maintained.

The goal is for the values to remain integers after cutting the cake into x pieces. One simple solution to this is to multiply all valuations by the number of pieces the cake will be cut into. As long as the original instance is tracked, as described in Section 3.2.4, the NSW will not be affected once the allocation is found. Furthermore, since instances are generated with float valuations between 0 and 1, these valuations need to be adjusted to integers. To maintain consistency, the instance can be normalized to 1000, as described in Section 2.2.4.

For completeness, the constraints on the MIP solver to find MNW are shown in Listing 5.1. It should be noted that these constraints differ from those used in the mixed instance, and instead use constraints where all goods are indivisible (similar to how items are made indivisible in Listing 4.1).

Listing 5.1 MIP constraints and objective for finding maximum nash welfare allocation. The requirement for this solver to work is that all valuations in the instance are integers. Furthermore, all goods are treated as indivisible

```
@assert all(typeof.(instance.valuations) .== Int))
# add variable with utility for each agent
JuMP.@variable(model, W[Agents(instance)])
# find largest total sum of valuations for an agent
agent_valuation_sums = [
    sum(instance.valuations[agent, :])
    for agent in Agents(instance)
]
largest_valuation_sum = Float64(maximum(agent_valuation_sums))
for agent in Agents(instance), k = 1:2:largest_valuation_sum
    JuMP.@constraint(model,
        W[agent] <=
              log(k)
            + (log(k + 1) - log(k))
            * (sum(A[agent, :] .* instance.valuations[agent, :]) - k)
    )
end
JuMP.@objective(model, Max, sum(W))
```

Chapter 6

Results

In this chapter, the results from the empirical experiments are presented. Utelizing the methodology in Chapter 3 and the algorithms and approaches in Chapter 4 and Chapter 5.

6.1 Using Indivisible Algorithms

6.1.1 Equal Valuations

The following are the results of the experiments on Algorithm 1.

m \ n	**3**	**4**	**5**	**6**	**7**	**10**	**11**
4	250	-	-	-	-	-	-
5	100	250	-	-	-	-	-
6	100	100	100	-	-	-	-
7	250	100	100	250	-	-	-
8	100	250	100	100	150	-	-
9	-	100	100	100	-	-	-
10	150	-	100	250	-	-	-
11	-	150	-	100	150	100	-
12	-	-	-	-	-	-	150
13	100	-	-	150	-	-	-
14	-	100	-	-	150	100	-
15	-	-	-	-	-	-	150
16	100	-	-	100	-	-	-
17	-	100	-	-	100	100	-
18	-	-	-	-	-	-	150
19	-	-	-	100	-	-	-
20	-	-	-	-	100	100	-
21	-	-	-	-	-	-	100
23	-	-	-	-	-	100	-
24	-	-	-	-	-	-	100
Total	1150	1150	500	1150	650	500	650

Table 6.1: Number of instances analyzed for each size (number of agents and goods). Number of goods includes the cake. The same instances are used for both Algorithm 1 and HiGHS.

n	**Algorithm**	**MIP**
3	0.9999999999996926	0.9999999999988765
4	0.9999999422369689	0.9999999422369682
5	0.999999799910786	0.9999997999107858
6	0.9999997901858854	0.9999997901858838
7	0.9999999999967798	0.9999999999965957
10	0.999999999999841	0.9999951314717833
11	0.999999999999747	0.9999904061530148

Table 6.2: Lowest achieved MMS value for any instance of the given number of agents.

Because of the large difference in time for the algorithm and the MIP to find their allocations the time each algorithms uses had to be placed in two separate figures. The figures show the time the algorithms used per allocation as the problem size increased.

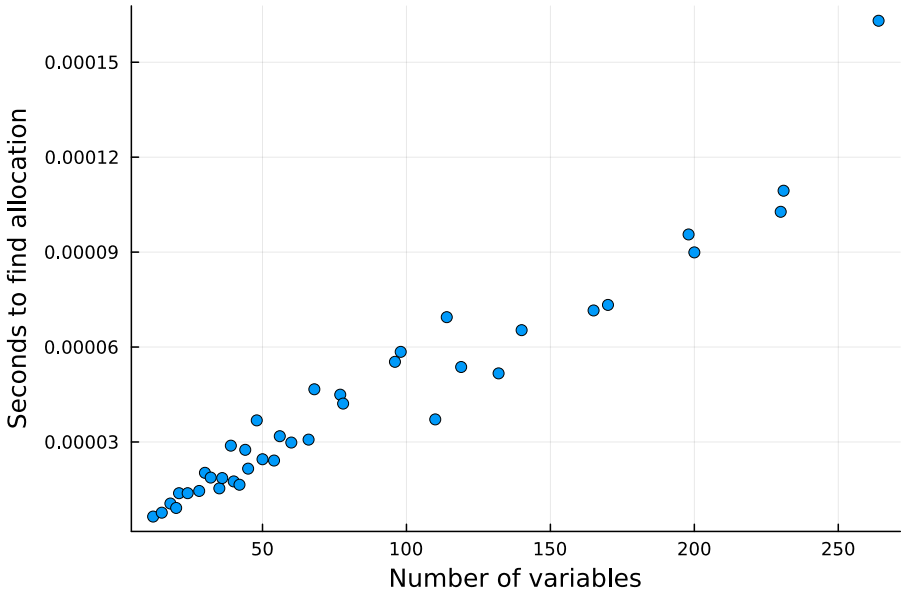

Figure 6.1: Median time to find allocation (s) for Algorithm 1 for each number of variables.

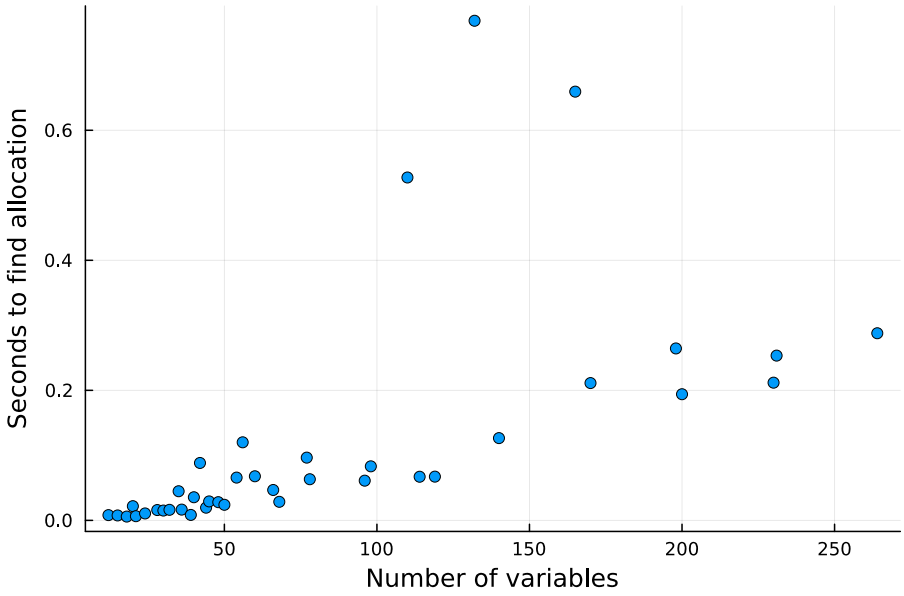

Figure 6.2: Median time to find allocation (s) for HiGHS for each number of variables.

6.1.2 Individual Valuations

Cutting cake into n pieces

Results of the experiment outlined in Section 4.2.1.

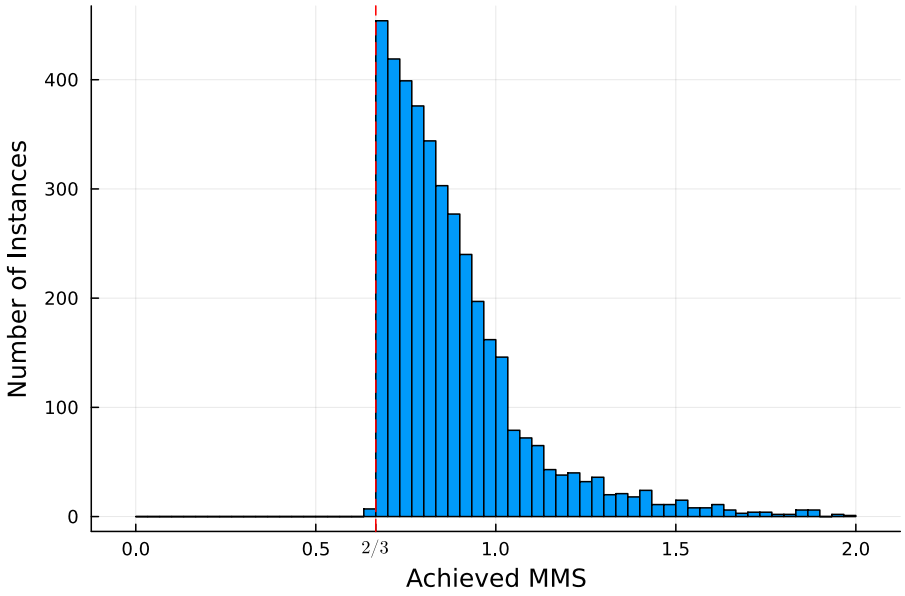

Figure 6.3: Histogram of achieved MMS values when cutting the cake into n pieces and then using a $2/3 - \text{MMS}$ algorithms to find an allocation.

n	**Lowest Achieved MMS**
3	0.6382675744858515
4	0.6423135282885898
5	0.6669351864641233
6	0.6672756126937586
7	0.6466091974982744

Table 6.3: Lowest achieved MMS value for any instance of the given number of agents.

Allocating Items First with Algorithm 2

Results of the experiment outlined in Section 4.2.2.

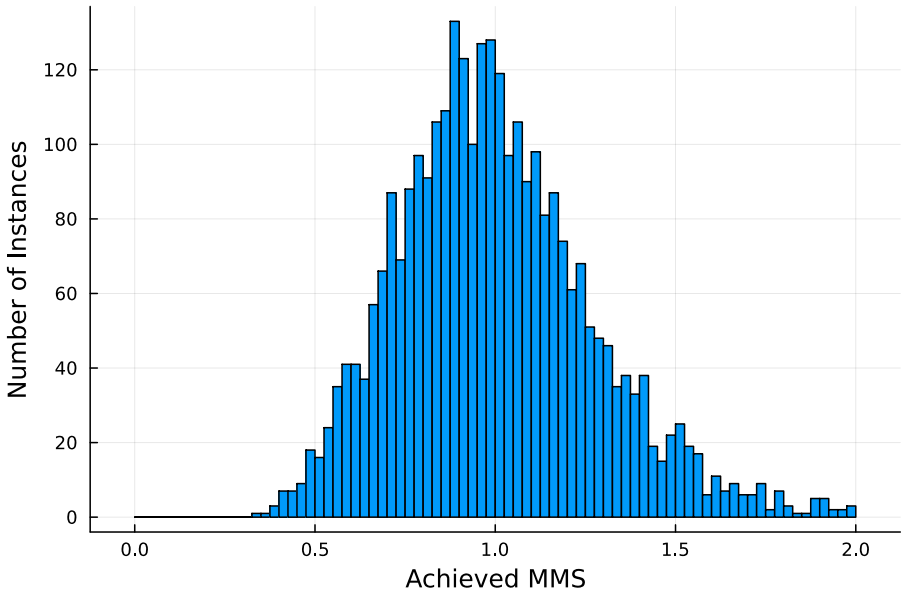

Figure 6.4: Histogram of achieved MMS values for Algorithm 2.

Improving Efficiency

As suggested in Section 4.3, a preliminary time analysis was conducted to evaluate the approach of using a MIP solver with a relaxed constraint for the items based on the size of the cake, followed by the cake assignment. However, the results of this analysis were not promising.

It was observed that the MIP solver was able to find an allocation faster with the cake than when separating the allocation process into two steps. Additionally, the achieved MMS values of the approach were often lower compared to those

obtained using the MIP solver alone. Due to these observations, the approach was not investigated further.

6.2 Maximum Nash Welfare

Results of experiment outlined in Section 5.2.

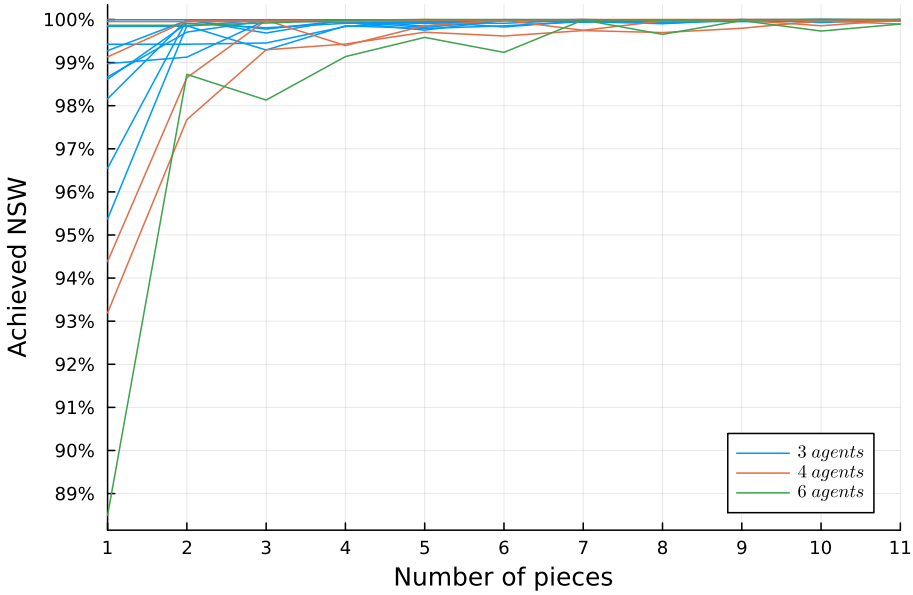

Figure 6.5: Progression of NSW as cake is cut into increasing amount of pieces. Each line is a unique instance. Plot is cropped to increase visibility, see Appendix A for full plot up to number of pieces = 100.

In addition to seeing how the NSW changed, its also interesting to see if the underlying allocation changes. That is does the items in the bundle of each agents change as the cake is cut into more pieces, and is it the same agents that receive any amount of cake. This is show in Figure 6.6

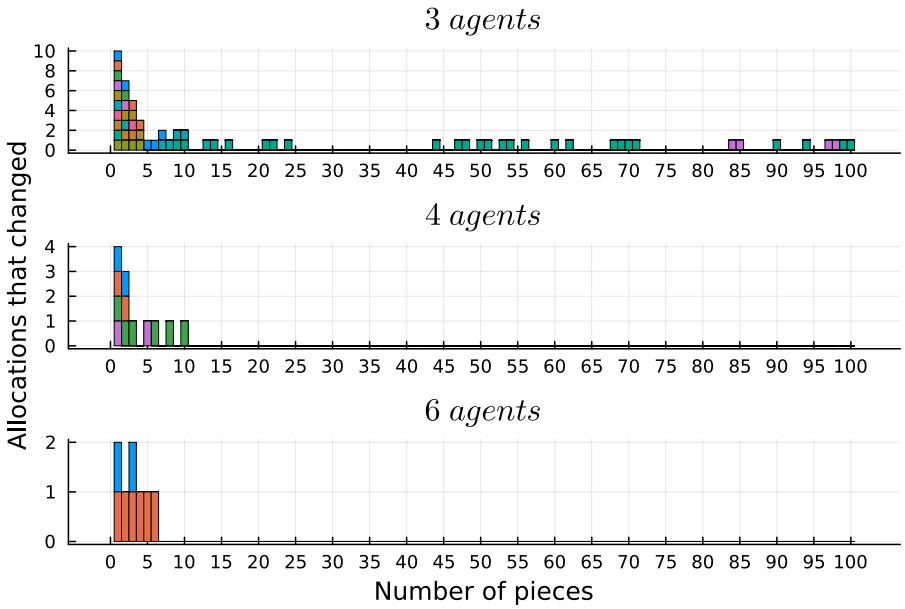

Figure 6.6: Which instances changed compared to when cake was cut into 1 less piece. Each color represents a unique instance.

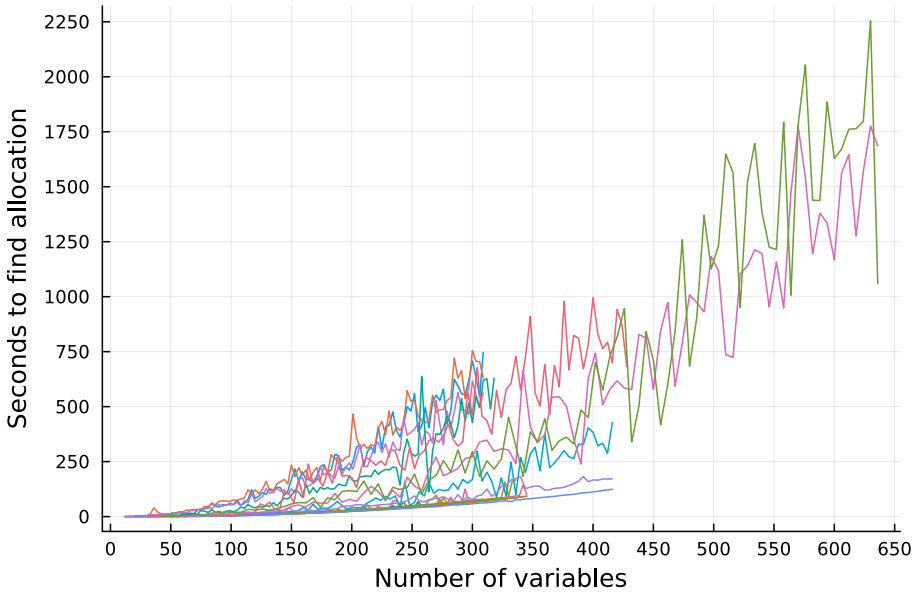

Figure 6.7: Time to find MNW as number of variables increase. Number of variables is determined by number of agents and number of goods (and pieces). An instance with 3 agents, 2 items and a cake cut into 10 pieces will have $3 * (2 + 10) = 36$ variables. Each color represents a unique instance.

The data in Figure 6.5, Figure 6.6 and Figure 6.7 all only include instances where the cake needed to be cut, remaining instances (where the cake is never cut) were excluded as their results are not interesting. The number of instances where the cake needed to be cut is shown in Figure 6.2.

n	**Generated Instances**	**Number with cut cake**	**%**
3	50	10	20.0%
4	7	4	57.1%
6	2	2	100%

Table 6.4: Number of generated instances and number of instances where the cake needed to be cut to achieve MNW.

Chapter 7

Discussion

7.1 Method

In this chapter, the specific results obtained from the empirical experimentation conducted in this thesis are explored. However, before proceeding with the details, it is essential to thoroughly examine the reliability of these results. Since the study relies primarily on empirical analysis, it becomes imperative to address various factors that may impact the reliability and validity of the findings. By critically assessing these factors, a more accurate interpretation of the results can be ensured, allowing for meaningful conclusions to be drawn from the experimentation. The following are the main factors considered to have the most significant influence on the reliability of the results in this thesis.

7.1.1 Instance Size

Perhaps the largest limitation of the experiments in this thesis comes down to the selection of number of agents and goods. The main focus of the instances in this thesis are where the number of goods is proportional to the number of agents. Say for instance you want help allocation all possesions of a deceased

person to their children. There is likely to be a large number of goods compared to number of agents.

The experiments are also meant to find how goods the algorithms and approaches work in practice, which should mean that the experiments should cover as wide range of cases as possible to catch edge cases and rare occurrences. This had to be limited however due to the immense time it takes to run some of the experiments and analysis, which reduced the number of instances that could be run.

7.1.2 Valuations

The problem of fair allocation in itself encompasses a large variety of different instances and settings. As such, it is impossible to cover all possible scenarios. The results presented in this thesis are based on a selection of instances that are deemed to be representative of the problem. However, it is important to note that the results presented in this thesis are not necessarily representative of all possible instances. Some of these instances were deliberately chosen to be excluded, such as finding allocations with only 2 agents, or where the cake is massive compared to the items.

By only using valuations between 0 and 1 for the items, this might seem very restricting, but as mentioned in Chapter 2, for most algorithms the values themselves are not as important as the relative difference between them.

7.1.3 The Cake

As mentioned in Chapter 3, the cake size is limited in terms of its value when generating random instances. This limitation is intentional and serves the purpose of exploring the behavior of algorithms under challenging conditions. The random instances are generated using three different cake size models, which are designed to cover the most probable cases.

By imposing a limit on the cake size, certain scenarios where the cake size is significantly larger than the items are eliminated. In such cases, the items become somewhat redundant as there will always be enough cake to allocate to all agents while ensuring their fair share. These scenarios are not particularly

informative in terms of evaluating the performance of algorithms under more difficult conditions.

The choice to set a limit on the cake size allows for a focus on instances where the cake size is more comparable to the items, posing a greater challenge for the allocation algorithms. This provides valuable insights into the effectiveness and efficiency of the algorithms in scenarios where resources are relatively scarce.

7.1.4 Choice of MIP Solver

It is worth noting that the MIP solver HiGHS, which was used in the experiments, is not considered the best or fastest solver available. There are other solvers, such as Gurobi, that are generally much faster than HiGHS. However, it is important to mention that Gurobi is a commercial solver and requires a license to use. This can make solutions that utilize Gurobi less accessible to those who want to implement or use them.

Additionally, it is important to recognize that Gurobi, like HiGHS, is also subject to the same limitations in terms of scalability. While it may perform well for a larger number of agents and goods, it will eventually encounter the same challenge of requiring an excessive amount of time as the problem size increases.

The choice to use HiGHS in the experiments was driven by its open-source nature, which allows for easier access and implementation. Despite its limitations, it provided a reasonable starting point for exploring and analyzing different algorithms for fair allocation.

7.2 Adapting Indivisible Algorithms for Mixed Goods

7.2.1 Equal Valuations

From Table 6.2, it is observed that Algorithm 1 finds a 1-MMS allocation for all instances. The values are slightly lower than 1, but this is highly likely due to precision errors from floating-point values. However, there are two cases that appear to be a little too low to be a precision error. These instances were then directly compared to the MIP solver. The instances and the allocations can be found in Appendix B. Here, it is visualized that the MIP solver finds either the exact same allocation (remember that all agents are equal, so only the order of the agents can be different) or an allocation with the exact same MMS value. This indicates that there is a different reason for these two instances having a slightly lower MMS value than there being something wrong with the algorithm.

In Figure 6.1 and Figure 6.2, it can be observed that the time taken by the algorithm to find these allocations is thousands of times faster than that of the MIP. The task of checking whether an instance meets this requirement is a simple one that can be performed as a preprocessing step. This check can then be used instead of other potentially slower algorithms, such as the MIP solver.

The limitations of this algorithm is of course the requirement it sets for the size of the cake. This reduces the generality of the algorithm significantly. Finding an algorithm that finds 1-MMS with equal valuations is still significant however because MMS algorithms for mixed goods, often rely on knowing the MMS-value for all agents beforehand (for instance the algorithms described in (Bei, Liu, et al., 2021)), which this algorithm now can find in polynomial time. A counter argument can be made that for the instances where the algorithm is guaranteed to find the 1-MMS allocation, the MMS values could simply be replaced by PROP. ie. the cake is large enough that all the agents can receive $\sum v(G)/n$. which is an approximation that is often used for approximate MMS-algorithms [1].

The exact usecases of this algorithm is then not obvious, however one argue that all instances can be converted to an instance with equal valuations by looking

[1] This is commonly implemented by normalizing each agents valuations to be n, such that PROP for each agent is $n/n = 1$

at all items monetary value, instead of sentimental value, where the cake is now considered a pool of money. In other words this algorithm can find allocations that guarantee all agents get bundles of the same monetary value as long as the pool of money is large enough.

Additionally, as the required size of cake is dependant on the guarantee of the indivisible algorithm, swapping the LPT algorithm with an algorithm that achieves a higher guarantee, will directly reduce the required size of the cake.

7.2.2 Cutting Cake into n Pieces

Expanding on the results from the pre-study, it was found that the relaxed constraint approach achieved the 1/2-MMS guarantee for all instances (Utne, 2022). And it was hypothesized that simply using an algorithm with a higher guarantee for indivisible goods would directly translate to a higher guarantee for mixed goods.

However, the analysis in this thesis revealed that the approach did not consistently achieve this using a $2/3 -$ MMS algorithm. In Figure 6.3, it can be observed that there are instances where the approach failed to meet the desired guarantee. Initially, this discrepancy was attributed to floating-point errors. But as shown in Table 6.3, the discrepancies were too substantial to be solely attributed to floating-point errors.

The underlying reason for this discrepancy lies in the fact that the indivisible algorithm uses the expected MaxiMinShare of the indivisible instance (instance with items and indivisible pieces of the cake) to determine whether an agent has achieved the guarantee or not. In other words, there is a possibility that an agent receives a bundle that is valued more than their indivisible MaxiMinShare, but less than their mixed MaxiMinShare. This is problematic because the indivisible algorithm assumes that the agent has already achieved their 2/3-MMS and will not assign them any additional goods, even if there are more goods available to distribute[2].

Due to the very small probability of this occuring ($7/3912 = 0,18\%$ of the analyzed instances), using this approach is still a very viable option. The oc-

[2]It is a common practice for MMS algorithms to assign any "overflowing" goods that remain after all agents have achieved the guarantee to a single agent to demonstrate the robustness of the guarantee.

currence of instances that don't quite match the guarantee is likely to increase along with the guarantee of the indivisible algorithm. There is also the rare possibility that one agent might require a very small pieces of cake to reach their MaxiMinShare. This probability will likely also increase along with the number of agents, similar to how the cake is more likely to be split the more agents there are, as seen in Figure 6.2.

7.2.3 Allocate Items First

Unfortunately simply using the same approach as in Listing 4.1 does not yield very promisising results, shown in Figure 6.4. The reason for this actually fairly straight forward. When the valuations are individual, that means one agent might value the cake ≈ 0, which means if that agent ever has one of the worst bundles in phase 2 of the algorithm, this agent will receive the entire cake. This obviously ruins the other agents chances to achieve their expected MMS as they valued the cake as large enough to be split between all of them.

Allocating items first also has another major drawback. For any real instance there is a chance that the optimal solution has one agent receive all the items, and another agent receives all the cake, however by forcing the items to be split first any such optimal allocation cannot be found by this algorithm. Although the majority of the allocation are above the expected $2/3-$MMS the distribution is clearly not impacted much by the guarantee of the indivisible algorithm.

If all agents agreed on a value for the cake, but has individual valuations for the items, then this approach would be viable, as the cake would then be able to fill up the worst bundles in a "predictable" way by all agents. Wether it maintains the guarantee for the indivisible algorithm is not certain however.

7.2.4 Improving Efficiency

Due to the limited experiments conducted with the relaxed constraint approach, there are not many results to discuss in detail. The decision to discontinue further analysis of this approach was based on several factors.

Firstly, it was realized that the approach would face similar limitations as discussed in Section 7.2.3. This raised concerns about the effectiveness and fairness

of the resulting allocations.

Secondly, the approach itself introduced additional complexity compared to simply using the MIP solver for mixed instances. The separation of items and the subsequent allocation of the cake added overhead and required more time than the solver needed to find the optimal division of the cake alone.

Considering these factors, it was determined that pursuing further analysis of the relaxed constraint approach would not yield substantial benefits and would not address the underlying limitations identified.

7.2.5 Generalizing for Heterogenous Cake

In order to generalize for heterogenous cake, and by extension multiple cakes, the following is required:

In order to generalize these findings for heterogenous cake, one could possibly utilize what is *Weighted Proportional Cake Cutting* as explained in (Bei, Liu, et al., 2021). This concept generalizes proportionality to the weighted case in cake cutting using a weight profile. This would however require some more preprocessing 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. (Utne, 2022)

7.3 Maximum Nash-Welfare for Mixed Goods

As can be observed in Figure 6.7, the time required to find allocations increases exponentially as the number of variables (agents and/or goods) increases. The worst cases depicted in the figure take a remarkable 2250 seconds to find a single allocation (equivalent to 37.5 minutes). Given that the experiment is only conducted with instances involving a maximum of 6 agents, cutting the cake into a large number of pieces is not a feasible option.

It can also be observed that some instances run substantially faster than others. This could be caused by two main factors.

1. Other work being done on the computer which could have an impact on the

time it takes to find the allocation due to computer resources being busy. 2. Some instances are better suited for MIP solvers in that the valuations allow the solver to converge faster to the optimal solution.

This problem of increased time also comes in addition to the risk that the MIP solver might struggle to actually find the optimal solution as the number of variables increase. AS described in (Caragiannis et al., 2019) the algorithm is also sensitive to the number of bits used to prevent errors due to rounding. The more variables are present, the more places values can be rounded in a non beneficial way causing the solver to miss the optimal solution.

Fortunately, in Figure 6.5, it is observed that the achieved NSW converges rapidly towards the optimum. It should be noted that there is no guarantee that the allocation maximizing the NSW has been found in the experiment, as such an allocation may require a finer granularity than the 100 pieces used in the analysis. However, it is reasonable to assume that the NSW would not differ significantly. Even when the cake is divided into n pieces, over 99% of the estimated optimum is reached by all the analyzed instances.

Additionally, it is noticed that what is more important than simply cutting the cake into many pieces is that the number of pieces should be divisible by the number of agents that need to receive a piece of the cake. It should be acknowledged that there is no way to know exactly how many agents require cake without finding the allocation first, but it is worth knowing that increasing the number of pieces might reduce the achieved NSW.

Additionally it very interesting to see that in Figure 6.6, the number of instances that change who receives items and cake as the number of pieces remain largely the same. This is very interesting as this means that you could theoretically use a small number of pieces to find the "main allocation", and then adjust how much each agent gets of the cake afterwards to find the "sweetspot" for the MNW. Its important to observe that for 2 of the instances with 3 agents, 1 of them keep changing up until 100 pieces, and the other changes once at 85 pieces and again at 97 pieces. Looking at the allocations, this is due to the fact two items of almost identical value was jumping back and forth between two agents. This is not unexpected as there could theoretically be more than one possible allocation that achieves the MNW.

7.4 When does the cake not need ot be cut

In terms of the instances where the cake was not cut, there were a few commonalities. The first was that if one agent valued the cake a lot more than all other agents, this agent was likely to receive the entire cake, which intuitively makes sense. Another commonality was that one or more agents valued the cake at almost 0, reducing the number of agents that "could" receive the cake, increasing the chance that one agent would receive the entire cake.

Other than those two cases, there was no clear pattern in the instances where the cake did not need to be cut. Both cakes of small, large, and random sizes were represented in both the cut and uncut cakes. Reducing the problem from finding a MNW allocation to only determining which agents receive cake is not possible either, as determining the recipients of the cake requires the MNW calculation.

Regarding the exclusion of uncut cakes from the results figures, this was done to reduce clutter since, when the cake is never cut, the NSW never changes.

7.5 Practical vs Optimal Allocation

Among the various algorithms and approaches discussed in this thesis, a significant focus has been placed on the time required to find an allocation. This emphasis stems from the understanding that a perfectly optimal algorithm is of little use if it never completes. The acceptable timeframe for finding an allocation may vary depending on its importance, but the pursuit of faster allocation methods is always a priority.

Another limitation that has not been previously addressed in this thesis is the issue of information constraints. Achieving a perfect allocation necessitates having complete knowledge of the problem at hand. While this may not be a significant challenge when allocating resources to machines, it becomes problematic when human input is involved. For instance, if there are 100 items to allocate, it would be unreasonable to expect each agent to provide precise valuations for all items that accurately reflect their preferences relative to one another. Some algorithms rely on "perfect-allocation oracles," which essentially involve continuously querying agents about all possible combinations of items

or cake pieces—an impractical and burdensome task.

Furthermore, for many real-world applications, having a resource that is infinitely divisible is either unrealistic or unnecessary. Consider the example of splitting a cake with a friend. Counting grams to achieve an exact equal split would be incredibly cumbersome, with minimal impact on how you and your friend feel about the division. Additionally, most divisible resources, such as money or land, have a minimum size (e.g., coins or square meters). This means that even if a resource is divisible, it is not infinitely divisible. Consequently, a degree of approximation, rounding, and algorithmic errors is generally acceptable in real-world applications.

When it comes to selecting an algorithm for a given problem or instance, the attributes of the instance largely dictate which approach will yield the most "fair" allocation. For example, if all agents have equal valuations, using MMS algorithms is highly applicable. On the other hand, if there is a lack of consensus among agents, a simple round-robin allocation method may suffice. (Note: The round-robin algorithm is a greedy approach in which each agent in turn selects the item they value the most from the remaining items.)

Chapter 8

Conclusion

The findings of this study suggest that adapting indivisible algorithms for the allocation of mixed divisible and indivisible goods can be satisfactory for real-life applications, although it may not always achieve strictly optimal allocations. Specifically, when using established MaxiMinShare (MMS) algorithms, distributing one piece of the cake to each agent allows the MMS algorithms to maintain their guarantee with high likelihood.

In cases where agents' valuations are equal, a simple polynomial time algorithm, such as the Longest Processing Time Heuristic, can be employed to find MMS allocations with high guarantees quickly. The experiments conducted revealed that the cake is sufficiently large, the algorithm can find 1-MMS allocations within polynomial time.

Regarding the Maximum Nash Welfare (MNW) for mixed goods, the results clearly demonstrate the impracticality of cutting the cake into numerous indivisible pieces. However, the findings also indicate that the Nash social welfare stabilizes rapidly once the number of cake pieces exceeds the number of agents. Furthermore, the allocation of items tends to remain relatively unchanged beyond this point. This implies that by initially dividing the cake into a few pieces, it can serve as a foundation for further adjustments.

It is important to note that the experiments conducted in this study were limited in terms of the problem size explored due to the extensive time required to run the experiments. This limitation may have influenced the results by potentially overlooking edge cases or other intriguing findings. Therefore, the outcomes should be regarded as a baseline expectation for randomly generated instances, but not necessarily indicative of how the algorithms and approaches would perform in real-life applications.

8.1 Future Work

Further theoretical proofs and studies are required to verify tha validity of the results proposed in this thesis.

An algorithm for finding an approximate MNW allocation for mixed goods without cutting the cake into indivisible pieces.

Another crucial task is to generalize the algorithms to effectively handle heterogeneous cake divisions and/or multiple cakes. By developing algorithms capable of accommodating heterogeneous cake divisions, a more comprehensive solution can be provided that caters to real-world scenarios with diverse preferences and requirements. Additionally, extending the algorithms to handle multiple cakes facilitates resource allocation across different domains, expanding the range of applications.

Bibliography

Amanatidis, G., Aziz, H., Birmpas, G., Filos-Ratsikas, A., Li, B., Moulin, H., ... Wu, X. (2022). *Fair division of indivisible goods: A survey.* arXiv. Retrieved from https://arxiv.org/abs/2208.08782 doi: 10.48550/ARXIV.2208.08782

Amanatidis, G., Markakis, E., Nikzad, A., & Saberi, A. (2017, dec). Approximation algorithms for computing maximin share allocations. *ACM Trans. Algorithms*, *13*(4). Retrieved from https://doi.org/10.1145/3147173 doi: 10.1145/3147173

Ariel Procaccia and Nisarg Shah. (2023). *Spliddit.* https://github.com/jogo279/spliddit. (Accessed: 8.6.2023)

Aumann, Y., Dombb, Y., & Hassidim, A. (2012). *Computing socially-efficient cake divisions.*

Barman, S., Biswas, A., Krishnamurthy, S., & Narahari, Y. (2018, Apr.). Groupwise maximin fair allocation of indivisible goods. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/11463 doi: 10.1609/aaai.v32i1.11463

Bei, X., Li, Z., Liu, J., Liu, S., & Lu, X. (2021). Fair division of mixed divisible and indivisible goods. *Artificial Intelligence*, *293*, 103436. Retrieved from https://www.sciencedirect.com/science/article/pii/S0004370220301831 doi: https://doi.org/10.1016/j.artint.2020.103436

Bei, X., Liu, S., Lu, X., & Wang, H. (2021). *Maximin fairness with mixed divisible and indivisible goods.* Springer Science.

Bhaskar, U., Sricharan, A. R., & Vaish, R. (2022). *On approximate envy-freeness for indivisible chores and mixed resources.* Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Caragiannis, I., Kurokawa, D., Moulin, H., Procaccia, A. D., Shah, N., & Wang, J. (2019, sep). The unreasonable fairness of maximum nash welfare. *ACM Trans. Econ. Comput.*, *7*(3). Retrieved from https://doi.org/10.1145/3355902 doi: 10.1145/3355902

Garg, J., & Taki., S. (2021). *An improved approximation algorithm for maximin shares.* arXiv.

Goldman, J., & Procaccia, A. D. (2015, jan). Spliddit: Unleashing fair division algorithms. *SIGecom Exch.*, *13*(2). Retrieved from https://doi.org/10.1145/2728732.2728738 doi: 10.1145/2728732.2728738

Hall, J., Galabova, I., Feldmeier, M., & Zanetti, F. (2023). https://highs.dev. (Accessed: 12.6.2023)

Hetland, M. L., & Hummel, H. (2022). *Allocations.jl.* (Repository: https://github.com/mlhetland/Allocations.jl)

Kurokawa, D., Procaccia, A., & Wang, J. (2016, Feb.). When can the maximin share guarantee be guaranteed? *Proceedings of the AAAI Conference on Artificial Intelligence*, *30*(1). Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/10041 doi: 10.1609/aaai.v30i1.10041

Nishimura, K., & Sumita, H. (2023). *Envy-freeness and maximum nash welfare for mixed divisible and indivisible goods.*

Spice, B. (2014). *CMU's 'Spliddit' Uses Math to Divide Things Fairly.* https://www.cmu.edu/news/stories/archives/2014/november/november4_spliddit.html. (Accessed: 8.6.2023)

Suksompong, W. (2018, mar). Approximate maximin shares for groups of agents. *Mathematical Social Sciences*, *92*, 40–47. Retrieved from https://doi.org/10.1016/j.mathsocsci.2017.09.004

Utne, S. (2022). *On fair allcoation of mixed goods.* Computer Science Specialization Project.

Woeginger, G. J. (1997). A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, *20*(4), 149-154. Retrieved from https://www.sciencedirect.com/science/article/pii/S0167637796000557 doi: https://doi.org/10.1016/S0167-6377(96)00055-7

Yaari, M., & Bar-Hillel, M. (1984). On dividing justly. *Social Choice and Welfare*. Retrieved from https://doi.org/10.1007/BF00297056

Appendix A

Plots

Below are the full plots from Chapter 6 where the full plots were adjusted to improve visibility/enhance the important parts of the plots. The full plots are included here for completeness.

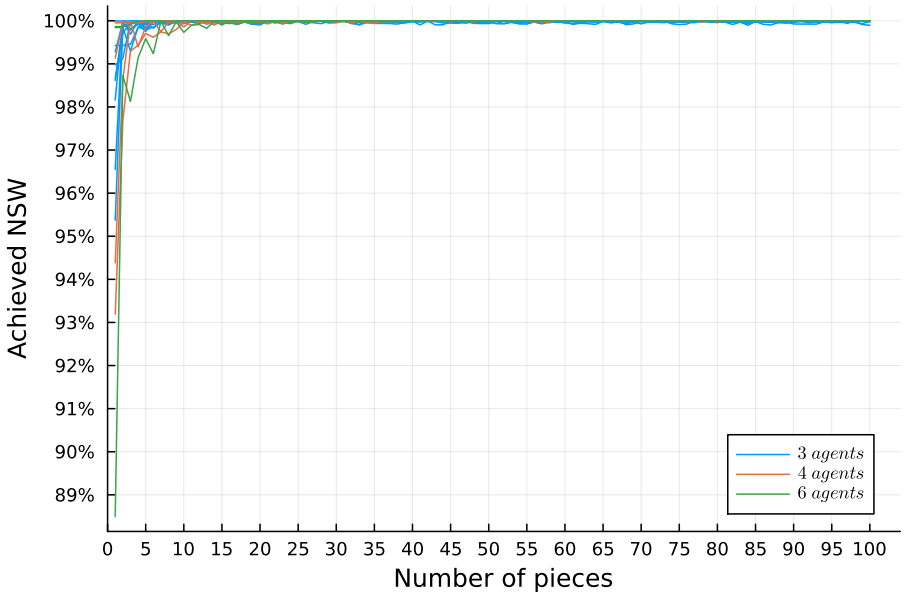

Figure A.1: Progression of NSW as cake is cut into increasing amount of pieces.

Appendix B

Allocations

Equal Valuations

The following are the two instances of which the algorithm for finding 1-MMS allocation with equal valuations achieved a MMS that is lower than what was deemed "acceptable" due to floating point errors. The allocation found by using the MIP solver is also shown for comparison. There were more instances where the MIP solver has a lower than acceptable MMS, refer to Appendix C which contains all logged instances to see these as well.

Instance: 4223.txt

```
Instance with SMALL cake:
  Goods:    Item, Item, Item, Item, Item, Item, Cake
  Agent 1: [0.51, 0.96, 0.07, 1.00, 0.78, 0.51, 0.87] - Mixed MMS: 0.65
  Agent 2: [0.51, 0.96, 0.07, 1.00, 0.78, 0.51, 0.87] - Mixed MMS: 0.65
  Agent 3: [0.51, 0.96, 0.07, 1.00, 0.78, 0.51, 0.87] - Mixed MMS: 0.65
  Agent 4: [0.51, 0.96, 0.07, 1.00, 0.78, 0.51, 0.87] - Mixed MMS: 0.65
  Agent 5: [0.51, 0.96, 0.07, 1.00, 0.78, 0.51, 0.87] - Mixed MMS: 0.65
  Agent 6: [0.51, 0.96, 0.07, 1.00, 0.78, 0.51, 0.87] - Mixed MMS: 0.65

MIP
110.0 ms: MMS=0.9999997901858838, NSW=0.77
  Agent 1: [    ,     ,     ,     , 1.00,     ,     ] - Bundle: 0.78
  Agent 2: [1.00,     ,     ,     ,     ,     , 0.16] - Bundle: 0.65
  Agent 3: [    ,     ,     ,     ,     ,     , 0.75] - Bundle: 0.65
  Agent 4: [    ,     ,     , 1.00,     ,     ,     ] - Bundle: 1.00
  Agent 5: [    , 1.00,     ,     ,     ,     ,     ] - Bundle: 0.96
  Agent 6: [    ,     , 1.00,     ,     , 1.00, 0.09] - Bundle: 0.65

ALGORITHM
0.021 ms: MMS=0.9999997901858854, NSW=0.77
  Agent 1: [    ,     ,     , 1.00,     ,     ,     ] - Bundle: 1.00
  Agent 2: [    , 1.00,     ,     ,     ,     ,     ] - Bundle: 0.96
  Agent 3: [    ,     ,     ,     , 1.00,     ,     ] - Bundle: 0.78
  Agent 4: [1.00,     ,     ,     ,     ,     , 0.16] - Bundle: 0.65
  Agent 5: [    ,     ,     ,     ,     , 1.00, 0.17] - Bundle: 0.65
  Agent 6: [    ,     , 1.00,     ,     ,     , 0.67] - Bundle: 0.65
```

Instance: 4737.txt

```
Instance with SMALL cake:
  Goods:    Item, Item, Item, Item, Item, Cake
  Agent 1: [0.45, 0.53, 0.89, 0.38, 0.42, 0.59] - Mixed MMS: 0.59
  Agent 2: [0.45, 0.53, 0.89, 0.38, 0.42, 0.59] - Mixed MMS: 0.59
  Agent 3: [0.45, 0.53, 0.89, 0.38, 0.42, 0.59] - Mixed MMS: 0.59
  Agent 4: [0.45, 0.53, 0.89, 0.38, 0.42, 0.59] - Mixed MMS: 0.59
  Agent 5: [0.45, 0.53, 0.89, 0.38, 0.42, 0.59] - Mixed MMS: 0.59


MIP Allocation
11.0 ms: MMS=0.9999997999107858, NSW=0.64
  Agent 1: [1.00,     ,     ,     ,     , 0.25] - Bundle: 0.59
  Agent 2: [    , 1.00,     ,     ,     , 0.10] - Bundle: 0.59
  Agent 3: [    ,     ,     ,     , 1.00, 0.29] - Bundle: 0.59
  Agent 4: [    ,     ,     , 1.00,     , 0.36] - Bundle: 0.59
  Agent 5: [    ,     , 1.00,     ,     ,     ] - Bundle: 0.89


ALGORITHM Allocation
0.014 ms: MMS=0.999999799910786, NSW=0.64
  Agent 1: [    ,     , 1.00,     ,     ,     ] - Bundle 0.89
  Agent 2: [    , 1.00,     ,     ,     , 0.10] - Bundle 0.59
  Agent 3: [1.00,     ,     ,     ,     , 0.25] - Bundle 0.59
  Agent 4: [    ,     ,     ,     , 1.00, 0.29] - Bundle 0.59
  Agent 5: [    ,     ,     , 1.00,     , 0.36] - Bundle 0.59
```

Appendix C

Source Code

All source code used for this thesis is added as a separate appendix in a ".zip" file. See the README file in the root of the archive for instructions on how to run the code in order to reproduce/validate the results presented in this thesis.

NTNU
Norwegian University of
Science and Technology