On the PTAS for Maximin Shares in an Indivisible Mixed Manna

Rucha Kulkarni[1], **Ruta Mehta**[1], **Setareh Taki**[1]
[1] University of Illinois at Urbana-Champaign
ruchark2@illinois.edu, rutameht@illinois.edu, staki2@illinois.edu

Abstract

We study fair allocation of indivisible items, both goods and chores, under the popular fairness notion of maximin share (MMS). The problem is well-studied when there are only goods (or chores), where a PTAS to compute the MMS values of agents is well-known.

In contrast, for the mixed manna, a recent result showed that finding even an approximate MMS value of an agent up to any approximation factor in (0,1] is NP-Hard for general instances. In this paper, we complement the hardness result by obtaining a PTAS to compute the MMS value, when its absolute value is at least 1/p times either the total value of all the goods or total cost of all the chores, for some constant p valued at least 1.

1 Introduction

Finding fair and efficient allocations is a fundamental problem in algorithmic game theory. The problem has been extensively studied for divisible resources, phrased as the cake cutting problem; see (Robertson and Webb 1998) for a summary. Here, a division of a *cake* that gave one piece to each of n agents was termed fair if it ensured properties like (a) envy-freeness, meaning every agent values her own piece more than those allocated to other agents, and (b) proportionality, meaning every agent values her piece at least $1/n$ fraction of her total value for the cake.

When there are two agents, the simple cut and choose protocol is known to work since the biblical era, where one agent cuts the cake into two pieces and the other agent gets to choose first. Recent years have seen a surge of works on the fair division of indivisible items, like school/course seats, assets and liabilities, and computing resources on networks, due to their wide applications (Steinhaus 1948; Brams and Taylor 1996; Vossen 2002; Moulin 2004; Etkin, Parekh, and Tse 2007; Budish 2011; Ghodsi et al. 2018). A simple example of allocating a single indivisible item among two agents shows that both envy-free and proportional allocations may not exist. Therefore, Budish (2011) defined the notion of maximin share (MMS) based on the following extension of the cut and choose protocol. If there are n agents, an agent partitions the items into n bundles assuming she will get to choose last. As she may end up with the least valued bundle, naturally, she will partition the items in such a way that the value of the least valued bundle is maximized. This is called her MMS value. An allocation where every agent receives a bundle of at least her MMS value is called an MMS *allocation*.

This problem is well studied for the *good manna* where all items are valued non-negatively by every agent, and for the *bad (chore) manna* where items are valued non-positively by everyone (See Section 1.1 for related work). We consider a *mixed manna* setting, where every item can be positively valued by some agents, and negatively valued by some. A natural starting question in the quest to find MMS allocations is,

Q: Given a mixed manna, what is the MMS value of every agent?

This question is NP-hard, even for the good manna. Note that finding the MMS value of an agent is equivalent to finding an MMS allocation when all agents have valuations identical to this agent. Hence, the problem of finding MMS allocations is also NP-hard, even with identical agents and a good manna. However, (Woeginger 1997) gave a PTAS for this setting. This PTAS was later used in several works to find approximate MMS allocations with non-identical agents (Procaccia and Wang 2014; Kurokawa, Procaccia, and Wang 2016; Amanatidis et al. 2017; Ghodsi et al. 2018; Garg and Taki 2020). The best known approximation result for the MMS problem with nonidentical agents in a chore manna (Huang and Lu 2019) also uses a PTAS for finding MMS values (Jansen, Klein, and Verschae 2016) as a subroutine. The next question then is,

Q: Is there a PTAS to find the MMS values of agents in a mixed manna setting?

Surprisingly, (Kulkarni, Mehta, and Taki 2020a) showed that even in a highly restricted setting of two identical agents where the mixed manna has only two chores, it is NP-hard to find *approximate* MMS values within *any* constant factor. Their reduction indicates that perhaps the bottleneck issue that makes the problem hard, is that the absolute MMS value can be arbitrarily small. Intuitively speaking, as the MMS value approaches arbitrarily close to zero, the problem of finding approximate MMS values approaches that of finding exact MMS values. In the limit where $\mathsf{MMS} = 0$, every approximate MMS value is the exact MMS value. We then ask,

Q: If we had a guarantee that the absolute MMS value is greater than some threshold value, say Δ, then does the problem become tractable?

In this paper, we resolve this question positively for a specific value of Δ. Note that since the MMS problem is scale-free, setting Δ to a fixed constant will not make the problem any easier. Therefore, Δ will have to be instance dependent. To be specific, let v^+ denote the sum of values of an agent for all the items she values positively, and v^- the sum of absolute values of all her negatively valued items (chores).

Theorem 1.1 (Informal). *In the case of identical agents, there is an algorithm that: (a) when $|\mathsf{MMS}| \geq \min\{v^+, v^-\}/\rho$ for some constant $\rho \geq 1$, finds an allocation that gives every agent a bundle of value at least $(1-\varepsilon)$-MMS for any constant $\varepsilon > 0$, and (b) when $|\mathsf{MMS}| < \min\{v^+, v^-\}/\rho$, reports this by returning the trivial allocation where all items are given to one agent. The algorithm runs in time $O(mnL)$, where m, n are the number of items and agents, and L is the bit-length of the input.*

We note that our assumption is weaker than having $\min\{v^+, v^-\}$ being a constant. Also, the extensively studied good manna and chore manna are special cases of this setting, hence any algorithmic results here translate to these settings as well.

One of the key tools used by our algorithm is a carefully designed Integer Program (IP) that can be solved in polynomial-time. IPs have been used to solve related problems in several works. (Woeginger 1997) gave a PTAS using this idea for the machine covering problem, which is equivalent to the MMS problem in a good manna. The MMS problem in a chore manna is equivalent to machine scheduling which has a PTAS using IP for bin packing (the dual problem). Several algorithms for the bin packing problem solve a relaxation of an IP as their main idea (De La Vega and Lueker 1981; Johnson 1982; Karmarkar and Karp 1982). Our approach builds on these, but requires several new ideas to handle both goods and chores simultaneously. Next we briefly describe some of these.

Non-constant variables. The variables of the IP will correspond to subsets of items. We show that we only need to consider subsets with total value at most a particular bound. With a good (chore) manna, this restricts to subsets where the number of items with value at least some fraction of the bound is a constant. While for a mixed manna, subsets of even $O(m)$ size may have a small value due to positive and negative values may cancel each other. Hence, the number of variables of the IP is not a constant for a mixed manna.

We circumvent this issue by reducing the problem to a problem with only goods, where a restricted set of allocations are allowed, called *valid allocations*.

Allow only valid allocations. The next task is to define constraints in the IP that ensure a valid allocation. Towards this, we define a cost function that characterizes valid allocations with a single constraint.

Sign of MMS. Our approach works for both cases $\mathsf{MMS} \geq 0$ (Section 3), and $\mathsf{MMS} < 0$. These are inherently different problems. The $\mathsf{MMS} \geq 0$ problem maximizes the smallest bundle's value, while the negative MMS case minimizes the absolute value of the largest bundle. We show that our IP for the former case can be modified to work for the later case of $\mathsf{MMS} < 0$.[1]

1.1 Related Work

The MMS problem has been extensively studied for the good manna (Kurokawa, Procaccia, and Wang 2016; Ghodsi et al. 2018; Garg, McGlaughlin, and Taki 2018; Kurokawa, Procaccia, and Wang 2018; Barman and Krishna Murthy 2017; Farhadi et al. 2019; Amanatidis et al. 2017; Garg and Taki 2020) and chore manna (Barman and Krishna Murthy 2017; Huang and Lu 2019) settings. With a good manna, there are several algorithms to find allocations that give every agent a bundle worth a constant fraction of their MMS value; the best factor known so far is $(3/4 + 1/(12n))$-MMS, by (Garg and Taki 2020). For the chore manna, (Huang and Lu 2019) give a PTAS to find the MMS values of agents, and an $11/9$ approximate MMS allocation. The study of the mixed manna setting started recently. (Kulkarni, Mehta, and Taki 2020a) gave a PTAS for the special case of the problem with a constant number of agents, when the total value of goods is some factor away from the total absolute value of chores.

2 Preliminaries and Notation

In this section, we formally define mixed instances and other relevant notions of maximin share. We use $[k]$ to denote the set $\{1, 2 \cdots, k\}$, and $(\mathscr{S}_j)_{j\in[k]}$ to denote the (multi-)set $\{\mathscr{S}_1, \mathscr{S}_2 \dots, \mathscr{S}_k\}$.

Definition 2.1. *An MMS instance is a tuple $\langle \mathscr{N}, \mathscr{M}, v\rangle$, where $\mathscr{N}$ is a set of n agents, $\mathscr{M}$ is a set of m indivisible items, and $v : 2^{\mathscr{M}} \to \mathbb{R}$ is the identical additive valuation function of all agents, represented by $v(S) = \sum_{j\in S} v_j$ for $S \subseteq \mathscr{M}$.*

A partition of all items among all agents is termed an *allocation*, denoted by $\mathscr{A} = \{A_1, A_2, \cdots, A_n\}$. Thus, $A_i \cap A_{i'} = \emptyset$ for all distinct i, i' in $\mathscr{N}$, and $\cup_i A_i = \mathscr{M}$.

Definition 2.2 (MMS value). *Given an MMS instance, let $\Pi_n(\mathscr{M})$ be the set of all possible allocations of $\mathscr{M}$ into n sets. The maximin share (MMS) value of an agent, denoted by $\mathsf{MMS}^n(\mathscr{M})$, is defined as,*

$$\mathsf{MMS}^n(\mathscr{M}) = \max_{\mathscr{A}\in\Pi_n(\mathscr{M})} \min_{A_k\in\mathscr{A}} v(A_k) \ .$$

We refer to $\mathsf{MMS}^n(\mathscr{M})$ by MMS when the qualifiers n and $\mathscr{M}$ are clear. Note that MMS can be negative too.

[1]The proof of Theorem 1.1 for the case when $\mathsf{MMS} < 0$ is similar to the case when $\mathsf{MMS} \geq 0$, and in some sense simpler, and is discussed in the full version (Kulkarni, Mehta, and Taki 2020b).

An allocation which gives every agent a set of items worth at least MMS is called an MMS allocation. Note that all agents have the same valuation function v for $\mathscr{M}$, hence the MMS values are same for all agents. Also, the allocation determining the MMS value for any agent is an MMS allocation. Hence when the agents are identical MMS allocations always exist. However, finding an MMS allocation is known to be NP-Hard (Bouveret and Lemaître 2016). Thus, we search for a PTAS to find almost optimal allocations, termed $(1-\varepsilon)$-MMS allocations, defined as follows.

Definition 2.3 ($(1-\varepsilon)$-MMS allocation)**.** *$\mathscr{A}$ is called a $(1-\varepsilon)$-MMS allocation, if for a given $\varepsilon > 0$, for each agent $i \in \mathscr{N}$ we have $v(A_i) \geq (1-\varepsilon)$MMS if MMS ≥ 0, and $v(A_i) \geq (1/(1-\varepsilon))$MMS, if MMS < 0. Equivalently,*

$$v(A_i) \geq \min\{(1-\varepsilon)\mathsf{MMS}, (1/(1-\varepsilon))\mathsf{MMS}\}.$$

Definition 2.4 (MMS problem)**.** *Given an MMS instance $\langle \mathscr{N}, \mathscr{M}, v\rangle$, the MMS problem is to find a $(1-\varepsilon)$-MMS allocation of $\mathscr{M}$ among $\mathscr{N}$.*

Items of a mixed manna can be divided into two sets. *Goods* are the items valued positively according to v. The set of goods is denoted by $\mathscr{M}^+ = \{j \in \mathscr{M} \mid v_j \geq 0\}$. *Chores* are items valued negatively, and the set of chores is termed $\mathscr{M}^- = \{j \in \mathscr{M} \mid v_j \leq 0\}$. We denote by v^+ the sum of values of all goods in the manna. That is, $v^+ = \sum_{j\in\mathscr{M}^+} v_j$. Similarly, we denote by v^- the sum of absolute values of all chores, i.e., $v^- = \sum_{j\in\mathscr{M}^-} |v_j|$.

To circumvent the hardness result of (Kulkarni, Mehta, and Taki 2020a) for the MMS problem for any $\varepsilon \in [0,1)$, we make the assumption that $|\mathsf{MMS}| \geq \min\{v^+, v^-\}/\rho$, for some constant $\rho \geq 1$. As we cannot decide before computing the MMS value if a given instance satisfies this property, we pose the following problem, termed the Bounded MMS problem, denoted by B-MMS.

Definition 2.5 (B-MMS problem)**.** *Given an MMS instance $\langle \mathscr{N}, \mathscr{M}, v\rangle$ and an $\varepsilon > 0$, return a $(1-\varepsilon)$-MMS allocation if MMS $\geq \min\{v^+, v^-\}/\rho$ for some constant $\rho \geq 1$, else report MMS $< \min\{v^+, v^-\}/\rho$ by returning the trivial allocation where one agent gets all items $\mathscr{M}$.*

In this paper, we give a polynomial time algorithm that solves the B-MMS problem. In other words, we provide a PTAS to find the MMS values of agents in a mixed manna, when the absolute MMS values are higher than $\min\{v^+, v^-\}/\rho$, for some constant $\rho \geq 1$.

The following lemma by (Kulkarni, Mehta, and Taki 2020a) shows an easy way to decide the sign of MMS, allowing us to design separate approaches for the negative and non-negative MMS cases.

Lemma 2.1. *$v(\mathscr{M}) \geq 0$ iff MMS ≥ 0.*

For solving the B-MMS problem, we first find the sign of MMS using Lemma 2.1, then apply the appropriate algorithm for that case.

3 Algorithm for B-MMS when MMS ≥ 0

In this section, we describe a PTAS for the B-MMS problem for the MMS ≥ 0 case. Some proofs are omitted from this section due to simplicity and they are available in the full version (Kulkarni, Mehta, and Taki 2020b). The main parts of the PTAS are explained and solved in separate subsections.

3.1 Reducing B-MMS to GC-MMS

We first reduce the given B-MMS problem to a new problem with only goods called the Goods manna Constrained MMS problem, denoted by GC-MMS. At a high level, this is similar to the MMS problem, but it computes optimal allocations over a restricted set of partitions, called *valid allocations* (described shortly).

The intuition behind defining GC-MMS problem is as follows: Suppose we replace every chore by $n-1$ goods, each of value equal to the absolute value of the chore. Lets call these good-copies of the chores. Every time we want to assign a chore $j \in \mathscr{M}$ to some agent, we instead assign one of the $n-1$ good-copies of j to the remaining $n-1$ agents (one copy for each $n-1$ agents). This adds exactly $|v_j|$ value to every bundle and therefore keeps their relative order the same. Once we do this for every chore, the value added to each bundle is exactly v^-, and is the same for every partition in $\Pi^n(\mathscr{M})$. Therefore, if we restrict the allocations in the new setting to allow an agent to get at most one good-copy of any chore, then MMS allocations in the two settings are equivalent.

Following this intuition, we define a GC-MMS instance and valid allocations as follows.

Definition 3.1 (GC-MMS instance)**.** *A tuple $\langle \mathscr{N}, \mathscr{G}, (\mathscr{S}_j)_{j\in[m^-]}, u\rangle$, where $\mathscr{N}$ is a set of agents, $\mathscr{G}$ is a set of goods, $(\mathscr{S}_j)_{j\in[m^-]}$ are $m-$ sets of goods, each containing $(n-1)$ identical copies of a good, and $u : \mathscr{M} \cup (\mathscr{S}_j)_{j\in[m^-]} \to \mathbb{R}_+$ is the identical valuation function of the agents in $\mathscr{N}$ for all items $\mathscr{G} \cup (\mathscr{S}_j)_{j\in[m^-]}$.*

Definition 3.2 (Valid allocation)**.** *Given a GC-MMS instance $\langle \mathscr{N}, \mathscr{G}, (\mathscr{S}_j)_{j\in[m^-]}, u\rangle$, an allocation $\mathscr{A}$ is* valid *if no agent receives more than 1 item from any set $\mathscr{S}_j \in (\mathscr{S}_j)_{j\in[m^-]}$, i.e., for all $i \in \mathscr{N}$, $j \in [m^-]$, $|\mathscr{A}_i \cap \mathscr{S}_j| \leq 1$.*

The GC-MMS problem asks to find a *valid allocation* that maximizes the value of the smallest bundle, i.e., an MMS allocation over the valid allocations. We abuse notation to denote both the problem and the value by GC-MMS, and formally define them as follows.

Definition 3.3 (GC-MMS value)**.** *Given a GC-MMS instance $\langle \mathscr{N}, \mathscr{G}, (\mathscr{S}_j)_{j\in[m^-]}, u\rangle$, let $\mathscr{F}$ be the set of all* valid *allocations. The GC-MMS value of the instance, denoted by GC-MMS, is defined as follows.*

$$\mathsf{GC\text{-}MMS} = \operatorname*{argmax}_{\mathscr{A}\in\mathscr{F}} \min_{A\in\mathscr{A}} v(A)$$

Since it is NP-hard to compute GC-MMS (even when $(\mathscr{S}_j)_{j\in[m^-]} = \emptyset$), define the following approximate version of the problem.

Definition 3.4 (GC-MMS problem)**.** *Given a GC-MMS instance $\langle \mathscr{N}, \mathscr{G}, (\mathscr{S}_j)_{j\in[m^-]}, u\rangle$ and $\varepsilon > 0$, return a valid allocation $\mathscr{A}$ such that $\min_{A\in\mathscr{A}} v(A) \geq (1-\varepsilon')$GC-MMS.*

Next we show that the B-MMS problem can be reduced to GC-MMS problem such that a PTAS for the latter gives a PTAS for the former.

Given an instance $\langle \mathcal{N}, \mathcal{M}, v\rangle$ we define the corresponding GC-MMS instance $\langle \mathcal{N}, \mathcal{G}, (\mathcal{S}_j)_{j\in[m^-]}, u\rangle$ as follows: The set of agents is unchanged, $\mathcal{G} = \mathcal{M}^+$, $m^- = |\mathcal{M}^-|$, and for all $j \in \mathcal{M}^-$, define $\mathcal{S}_j$ to be a set of $(n-1)$ goods represented as $\mathcal{S}_j := \{(j,k)|k \in [n-1]\}$ – $\mathcal{S}_j$ consists of good-copies of chore j. Finally, define $u(j) = v(j)$ for all $j \in \mathcal{G}$ and $u((j,k)) = -v(j)$ for all $j \in \mathcal{M}^-$ and $k \in [n-1]$.

Lemma 3.1. *Allocations of* B-MMS *are in one-to-one correspondence with valid allocations of* GC-MMS*, such that if allocation B^π of the former maps to allocation C^π of the later then $u(C_i) = v(B_i) + v^-$, $\forall i \in \mathcal{N}$.*

Proof. Given a B-MMS allocation B^π, add good-copies of each chore to agents who did not receive the chore in B^π, and discard all chores. This gives a GC-MMS allocation C^π. The reverse allocation is obtained by similarly discarding all good-copies and assigning the corresponding chore to the agent who did not receive any good-copy.

Every agent $i \in \mathcal{N}$ receives in C_i all the goods assigned to her in B_i. Every chore that was assigned to her in B_i is discarded in C_i. Due to this, her value increases by the absolute value of chores allotted to her in B_i. Further, for every chore not assigned to her, she receives a good-copy of it in C_i. As for all $j \in C, u(j) = |v(j^*)|$ for the corresponding j^* in $\mathcal{M}$, each good-copy increases her value by the absolute value of the corresponding chore. Her total valuation increases by the absolute value of all chores not assigned to her as well. Hence, the difference $u(C_i) - v(B_i)$ is exactly, $\sum_{j\in B_i} v(j) + \sum_{j\notin B_i} v(j) = v^-$. ☐

Corollary 3.1. GC-MMS*, relates to the* MMS *value of the* B-MMS *problem as,*

$$\text{GC-MMS} = \text{MMS} + v^-. \tag{1}$$

Equation (1) allows to relate the approximation parameters of B-MMS and GC-MMS allocations as follows.

Theorem 3.1. *If* $\text{MMS} \geq v^-/\rho$*, then a* $(1 - \frac{\varepsilon}{(1+\rho)})$GC-MMS *allocation gives a* $(1-\varepsilon)$-MMS *allocation, and therefore a* PTAS *for* GC-MMS *gives a* PTAS *for the* B-MMS *problem.*

Proof. Let $\varepsilon' = \frac{\varepsilon}{(1+\rho)}$. Take the $(1-\varepsilon')$GC-MMS allocation, say C^π, and consider the corresponding allocation B^π of the B-MMS instance as described in the proof of Lemma 3.1. From Lemma 3.1, the smallest bundle in B^π has value $(1-\varepsilon')\text{GC-MMS} - v^-$.

If $\text{MMS} \geq v^-/\rho$, we have, $(1-\varepsilon')\text{GC-MMS} - v^- \geq (1-\varepsilon')(\text{MMS} + v^-) - v^- \geq (1-\varepsilon')(\text{MMS} + \rho\text{MMS}) - \rho\text{MMS} = (1-(1+\rho)\varepsilon')\text{MMS} = (1-\varepsilon)\text{MMS}$. Therefore, B^π is a $(1-\varepsilon)$-MMS allocation

Since ρ and ε are constants in the B-MMS problem, ε' is also a constant. Therefore, a PTAS for GC-MMS is indeed a PTAS for the B-MMS problem as well. ☐

Due to the above theorem, it suffices to obtain a PTAS for the GC-MMS problem.

3.2 Algorithm for GC-MMS

Algorithm 1 for GC-MMS will perform a search for the highest value μ for which we get an allocation that gives every agent at least a μ-valued bundle. For this we perform a search on a multiplicative grid over all possible values of GC-MMS, obtained as follows. First, we have $v^-/\rho \leq \text{MMS} \leq v(\mathcal{M})/n = (v^+ - v^-)/n$. Combined with Equation (1), we get $v^- + v^-/\rho \leq \text{GC-MMS} \leq (v^+ - v^-)/n + v^-$.

In each iteration of the search, it first checks if there is an item with value more than μ. First, there will be no such chore. Because if there was one, say c, we have $c > \mu \geq \text{GC-MMS} = \text{MMS} + v^-$ which implies $\text{MMS} < c - v^- \leq 0$.

If there is a good j with $v(j) \geq \mu$, we have $\mu - v^- \geq \text{GC-MMS} - v^- = \text{MMS(B-MMS instance)}$. Using this we find B^π, a solution of the B-MMS instance as follows: assign good j and all the chores to an agent, and remove the agent and her bundle. The following allocation for the resulting instance is feasible, and has equal or higher MMS value. From any MMS allocation of the B-MMS instance, (a) remove all chores and add them to the part containing the good j, and (b) remove all goods except j from this part and arbitrarily distribute among the remaining parts. The MMS value of the resulting instance is not lower, and therefore it suffices to find it's $(1-\varepsilon)$-MMS allocation using the PTAS of (Woeginger 1997). Algorithm 1 returns allocation C^π corresponding to this B-MMS allocation.

If every item has value at most μ, the algorithm applies a subroutine Exists-GC-MMS, for which we prove in Section 3.3,

Theorem 3.2. Exists-GC-MMS$(\langle \mathcal{N}, \mathcal{G}, (\mathcal{S}_j)_{j\in[m^-]}, u\rangle, \varepsilon, \mu)$ *returns a tuple* $(\mathcal{A}, flag)$ *with* $flag = true$ *and* $u(A) \geq (1-\varepsilon)\mu$, $\forall A \in \mathcal{A}$*, whenever* $\mu \leq$ GC-MMS*. And it runs in* $O(mn)$ *time.*

If Exists-GC-MMS returns a false flag, the Algorithm resets $\mu \leftarrow (1-\varepsilon)\mu$ and starts the next iteration, else returns the allocation obtained and stops. Theorem 3.2 implies the following.When Algorithm 1 stops, say for a value μ^*, we know $\text{GC-MMS} \leq \mu^*/(1-\bar{\varepsilon})$, from the false flag returned in the previous iteration. From this iteration's output, we have a $(1-\bar{\varepsilon})^2$GC-MMS allocation. Fixing $\bar{\varepsilon}$ as $\varepsilon'/2$ gives,

Lemma 3.2. *If* $\text{GC-MMS} \geq (1+1/\rho)v^-$*, Algorithm 1 returns a* $(1-\varepsilon')$GC-MMS *allocation.*

We are now ready to show Theorem 1.1 for the case when $\text{MMS} \geq 0$.

Theorem 3.3. *There is an algorithm to solve the* B-MMS *problem for the case* $\text{MMS} \geq 0$*, that runs in time* $O(mnL)$*, where L is the number of bits needed to represent function v.*

Proof. From Theorem 3.1, it suffices to get a PTAS for the corresponding GC-MMS problem. By Lemma 3.2 Algorithm 1 does solve a GC-MMS problem. The while loop of the algorithm runs for $\frac{1}{\bar{\varepsilon}} \log\left(\frac{v^+ + (n-1)v^-}{n} - \frac{(1+\rho)v^-}{\rho}\right) \leq \frac{2(1+\rho)}{\varepsilon} L$ many times. By Theorem 3.2 and (Woeginger 1997), every iteration of the while loop takes at most $O(mn)$ time, and therefore the overall running time is $O(mnL)$. ☐

Algorithm 1: Algorithm for GC-MMS

```
Input  : ⟨𝒩, 𝒢, (𝒮_j)_{j∈[m⁻]}, u⟩, ε' > 0
Output: (1 − ε')GC-MMS allocation if
        GC-MMS ≥ (1 + 1/ρ)v⁻
ε̄ ← ε'/2; μ ← v⁺/n + (1 − 1/n)v⁻
while μ ≥ (1 + 1/ρ)v⁻ do
    if ∃ j ∈ 𝒢 : u(j) ≥ μ then
        𝒜 = (A_1, ..., A_n), A_n ← {j}
        (A_1, ..., A_{n−1}) ← (1 − ε̄)-MMS partition of
          ⟨𝒩 \ {n}, 𝒢, u⟩        // use PTAS of
          (Woeginger 1997)
        A_i ← A_i ∪ {(j, i)} for all i ∈ [n − 1] and
          j ∈ [m⁻]
        return 𝒜
    (𝒜, flag) ←
      Exists-GC-MMS(⟨𝒩, 𝒢, (𝒮_j)_{j∈[m⁻]}, u⟩, ε̄, μ)
    If flag then return 𝒜
    else μ ← (1 − ε̄)μ
𝒜 = (A_1, ..., A_n) where A_i = {(j, i) : ∀ j ∈ [m⁻]} for
  i ∈ [n − 1], A_n = 𝒢        // agents 1 to n−1
  each get one good-copy of all
  chores.
return 𝒜
```

The next section shows Theorem 3.2.

3.3 Algorithm for Exists-GC-MMS

At a high level, we first map the set of items in the Exists-GC-MMS instance to multi-sets of numbers corresponding to their values (scaled to have $\mu = 9\lceil\frac{1}{\varepsilon}\rceil^2$ for technical reasons). *Valid partitions* of these numbers are defined analogously like valid allocations of the GC-MMS items. We then classify the values as BIG or SMALL. The key component of the algorithm is an IP to find a valid partition of the BIG values such that (a) every part has value at least $9(\lceil\frac{1}{\varepsilon}\rceil^2 - \lceil\frac{1}{\varepsilon}\rceil)$, and (b) there are enough SMALL values to greedily allocate over this partition and have every part valued at least $9\lceil\frac{1}{\varepsilon}\rceil^2$. We now discuss the details of the algorithm formally.

Exists-GC-MMS has two steps 1) Pre-processing and 2) Main Algorithm.

Pre-processing.(Algorithm 2, line 1) Let $E := \lceil\frac{1}{\bar{\varepsilon}}\rceil$. Note that E is a constant integer that only depends on $\bar{\varepsilon}$ and not on parameters in the GC-MMS instance. Scale the valuations v by $9E^2/\mu$. Let $\mathscr{V}^g = (g_j)_{(j\in[m^+])}$ and $\mathscr{V}^c = \cup_{j\in[m^-]} C_j$, where $C_j = (c_j^k)_{k\in[n-1]}$ be multi-sets of numbers corresponding to scaled valuations, respectively of $\mathscr{M}^+$ and $(\mathscr{S}_j)_{j\in[m^-]}$. Let $\mathscr{T} = \mathscr{V}^g \cup \mathscr{V}^c$.

This completes the pre-processing step. The following lemmas characterize partitions of $\mathscr{T}$ that correspond to approximately optimal GC-MMS allocations.

Definition 3.5 (Valid Partition of $\mathscr{T}$)**.** *We call a partition $P = (P_1, \ldots, P_n)$ of values in $\mathscr{T}$* valid if each P_k contains at most one element from each C_j, i.e., $|P_k \cap C_j| \leq 1$ for all $k \in [n]$ and $j \in [m^-]$.

It is easy to see that each valid partition of $\mathscr{T}$ is equivalent to a valid allocation in its corresponding GC-MMS instance. With the scaling step, this directly implies,

Lemma 3.3. *Given a* GC-MMS *instance $\langle \mathscr{N}, \mathscr{G}, (\mathscr{S}_j)_{j\in[m^-]}, u\rangle$, if $\mu \leq$* GC-MMS *then there is a valid partition of $\mathscr{T}$ where the sum of values in each part is at least $9E^2$.*

As $E = \lceil 1/\bar{\varepsilon}\rceil$, we can show that a part of value at least $9E^2 - 9E$ will correspond to a bundle of value at least $(1-\bar{\varepsilon})\mu$. We use this and Lemma 3.3 to show the next lemma.

Lemma 3.4. *A valid partition of $\mathscr{T}$ where the sum of values in each part is at least $9(E^2 - E)$ is equivalent to a valid allocation for its corresponding* GC-MMS *instance where each bundle has value at least $(1-\bar{\varepsilon})\mu$.*

Main Algorithm. Call a valid partition of $\mathscr{T}$ *optimal* if the sum of values in each part is at least $9(E^2 - E)$. This step returns an optimal partition if $\mu \leq$ GC-MMS, else correctly reports $\mu >$ GC-MMS by returning $flag = false$. Note that Algorithm 1 runs Exists-GC-MMS only if every item has value at most μ. Hence, after scaling by $9E^2$, we can assume $t \leq 9E^2$, $\forall t \in T$. The key of the algorithm is an IP. We first explain the IP.

Notation. We define SMALL and BIG values in $\mathscr{T}$. Call a value $t \in \mathscr{T}$ SMALL if $t < 3E$ and BIG if $t \geq 3E$. For each $T \subseteq \mathscr{T}$ let SMALL(T) be the set of all small values in T and BIG(T) be the set of all big values in T. We call a set $C_j \in \mathscr{V}^c$ *small* if it contains SMALL values and *big* otherwise[2]. Let $\sigma, \sigma^+, (n-1)\cdot\sigma^-$ respectively be the sum of all values in SMALL$(\mathscr{T})$, SMALL$(\mathscr{V}^g)$ and SMALL$(\mathscr{V}^c)$, , i.e., $\sigma := \sum_{t\in \mathsf{SMALL}(\mathscr{T})} t$, $\sigma^+ := \sum_{t\in\mathsf{SMALL}(\mathscr{V}^g)} t$ and $\sigma^- := (\sum_{t\in\mathsf{SMALL}(\mathscr{V}^c)} t)/(n-1)$. Note that σ^- is equal to the sum of values obtained by picking one value from each small C_j, and $\sigma = \sigma^+ + (n-1)\sigma^-$.

Next, we know that every BIG value will be in the range $[3E, 9E^2]$. For all integers r in $[3E, 9E^2]$, let n_r^+, n^- respectively be the number of values in BIG$(\mathscr{V}^g)$ and the number of sets C_j with integral part of values r. Thus, $(n-1)n_r^- + n_r^+$ items j in $\mathscr{V}^g \cup \mathscr{V}^c$ have $\lfloor j \rfloor = r$.

We now define notation to represent a subset of BIG values and their sum. Let X denote a part in a partition of $\mathscr{T}$. We define the *type* of X by $\tau(X) = \langle \underline{\tau}(X), \overline{\tau}(X)\rangle = (\underline{\tau}_{3E}, \ldots, \underline{\tau}_{9E^2}, \overline{\tau}_{3E}, \ldots, \overline{\tau}_{9E^2})$; here $\underline{\tau}_r, \overline{\tau}_r$ are resp. the number of values in BIG$(X \cap \mathscr{V}^g)$ and BIG$(X \cap \mathscr{V}^c)$ with integer part r. Let SIZE$(\tau(X)) := \sum_r r(\underline{\tau}_r + \overline{\tau}_r)$ be the total sum of these rounded values in BIG$(\mathscr{T} \cap X)$.

Using this notation, we design an IP to find an assignment of BIG values in an optimal partition. First, observe that every BIG value is at most $9E^2$. Thus, if an optimal partition has some part valued more than $18E^2$, we can remove values until the size of this set is in the range $[9E^2, 18E^2]$. Finding a *partial* allocation of BIG values that assigns at least

[2]Note that each C_j, $j \in [m^-]$ contains $n-1$ equal values. i.e, for each C_j, either SMALL$(C_j) = \emptyset$ or SMALL$(C_j) = C_j$.

$9E^2$ value to all parts suffices to solve Exists-GC-MMS, as we can arbitrarily add the unallocated values. Thus, we will only consider types whose size $\textsc{Size}(.)$ is at most $18E^2$.

The variables of the IP correspond to all types τ that satisfy (i) $\textsc{Size}(\tau) \le 18E^2$, (ii) $\underline{\tau}_r \le n_r^+$, (iii) $\overline{\tau}_r \le n_r^-$. Let $\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(\Gamma)}$ be an enumeration of all variables. Intuitively, we consider types that represent valid allocations of items corresponding to the BIG values in a GC-MMS instance. Every IP variable takes an integer value equal to the number of times the corresponding type is selected. This in turn represents the number of parts in the output allocation that have a subset of BIG items as represented by this type.

Lemma 3.5. *The number of IP variables* Γ *is* $O(1)$.

Proof. Every type with size at most $18E^2$ can have at most $6E$ BIG values, as every BIG value is at least $3E$. Each value is one of $[3E, 9E^2]$, a constant sized set. Hence, the number of types $\bar{\tau}$ and $\underline{\tau}$ are each at most $(9E^2 - 3E + 1)^{6E}$. The total number of types at most twice this value, hence a constant as E is a constant. The number of variables of the IP is at most the number of types with size at most $18E^2$, hence is constant. □

Before defining the IP, we define two *cost* functions for every type. These are used to define constraints to allocate SMALL items.

First, define $c(\tau(X)) := \max\{0, 9E^2 - 6E - \textsc{Size}(\tau(X))\}$. The intuition for this function is as follows. Our aim is to create an optimal partition. If the sum of BIG values $\textsc{Size}(\tau(X)) < 9(E^2 - E)$, we must add values from $\mathsf{SMALL}(\mathscr{T})$ to X. The required sum from SMALL, is at least $9E^2 - 9E - \textsc{Size}(\tau(X))$. However, $\mathsf{SMALL}(\mathscr{T})$ does not have arbitrarily precise values. As every SMALL value is at most $3E$, we may have to add SMALL items until the net value of the part becomes $3E$ more than required, i.e., $9E^2 - 6E$. Hence the cost function $c(\tau(X))$ is defined as specified.

The second cost function captures the value that must be added to a part from $\mathsf{SMALL}(\mathscr{V}^g)$. If a part has $c(\tau) > 0$, we can add at most value σ^- to the part from $\mathsf{SMALL}(\mathscr{V}^c)$. Hence, the minimum value from $\mathsf{SMALL}(\mathscr{V}^g)$ is $\sigma^+(\tau(X)) := \max\{0, c(\tau(X)) - \sigma^-\}$.

Using these notions, we define the following IP for finding an allocation of BIG values.

$$\sum_{j=1}^{\Gamma} x_j = n; \;\; x_j \in \{0\} \cup \mathbb{N}, \forall j \in [\Gamma] \tag{2}$$

$$\sum_{j=1}^{\Gamma} \underline{\tau}_r^{(j)} x_j \le n_r^+ \forall r \in [3E, 9E^2] \tag{3}$$

$$\sum_{j=1}^{\Gamma} \overline{\tau}_r^{(j)} x_j \le (n-1) n_r^-, \forall r \in [3E, 9E^2] \tag{4}$$

$$(a) \sum_{j=1}^{\Gamma} c(\tau^{(j)}) x_j \le \sigma; \; (b) \sum_{j=1}^{\Gamma} \sigma^+(\tau^{(j)}) x_j \le \sigma^+ \tag{5}$$

Algorithm 2: Exists-GC-MMS

Input : $\langle \mathscr{N}, \mathscr{G}, (\mathscr{S}_j)_{j \in [m^-]}, u \rangle$, $\bar{\varepsilon}$, μ
Output: $(\mathscr{A}, \text{True})$ if there exists a $(1-\bar{\varepsilon})$-GC-MMS allocation $\mathscr{A}$ and $(\emptyset, \text{False})$ otherwise

$\mathscr{V}^g \leftarrow \{g_1, \ldots g_{m^+}\}, g_j = u(j) \cdot \left(\frac{9E^2}{\mu}\right), j \in \mathscr{G}$
  $\mathscr{V}^c \leftarrow \bigcup_{j \in [m^-]} C_j, C_j := \{c_j^1, \ldots, c_j^{n-1}\}$,
  $c_j^k = u(j,k), \forall (j,k) \in \mathscr{S}_j, \forall \mathscr{S}_j \in (\mathscr{S}_j)_{j \in [m^-]}$;
  $\mathscr{T} \leftarrow \mathscr{V}^g \cup \mathscr{V}^c$
if *IP has a solution X for $\mathscr{T}$* **then**
  $j \leftarrow 1$
  for *all $i : x_i \ne 0$* : **do**
    Create x_i parts P_j to P_{j+x_i}
    Add BIG values to each P_k, $k \in [j, j+x_i]$ as per $\tau^{(i)}$; $j \leftarrow j + x_i + 1$
  while $\exists k : \sum_{j \in P_k} j < (9E^2 - 9E)$ **do**
    while $\sum_{j \in P_k} j < (9E^2 - 9E)$ **do**
      If $P_k \cap C_j = \emptyset$ for any $j \in [m^-]$ **then** add one value from C_j to P_k
      else $P_k \leftarrow P_k \cup$ any $j \in \mathsf{SMALL}(\mathscr{V}^g)$
  while *there is an unallocated value k from C_j for any $j \in [m^-]$* **do**
    Add k to any $P_i : P_i \cap C_j = \emptyset$
  Add remaining unallocated values arbitrarily
  $\mathscr{A} \leftarrow$ allocation corresponding to $P = (P_1, \cdots, P_n)$ // use Lemma 3.3
  return $(\mathscr{A}, \mathit{True})$
return $(\emptyset, \mathit{False})$

The Exists-GC-MMS algorithm is as follows. After applying the pre-processing step, it defines and solves the above IP. If the IP has a solution, then first it considers the items from the GC-MMS instance that correspond to the BIG values in $\mathscr{T}$. The algorithm partitions all these items in n bundles by creating n subsets, with x_i subsets corresponding to type $\tau^{(i)}$. After this, it considers the subsets of BIG items that do not have total sum of values at least $9E^2 - 9E$. To each of these, it first adds the SMALL items corresponding to the small C_j subsets, by adding at most one item from each subset C_j, in any order. If upon adding these, the value of the set is still not $9E^2 - 9E$, it adds items corresponding to the $\mathsf{SMALL}(\mathscr{V}^g)$ set, until the total sum of values is at least $9E^2 - 9E$. The algorithm returns the tuple $(\mathscr{A}, true)$, where $\mathscr{A}$ is the allocation formed by this process. If the IP does not have a solution, it returns the tuple $(\emptyset, flag = false)$.

Algorithm 2 formally describes Exists-GC-MMS. We now analyze the correctness of Exists-GC-MMS.

Lemma 3.6. *If* $\mu \le$ GC-MMS, *then IP has a solution.*

Proof. As $\mu \le$ GC-MMS, from Lemma 3.3, there is a valid partition of $\mathscr{T}$ with sum of values of each part at least $9E^2$. Let this partition be T^{IP}. Let $\tau^i = \tau(T_i^{\text{IP}})$ be the type of each part, and $\tau^{\text{IP}} = [\tau^1 \cdots, \tau^n]$, be the multi-set of types of all parts.

Constraints (2), (3) and (4) hold for τ^{IP} by definition of a valid partition. For any $\tau^i \in \tau^{\text{IP}}$ with $c(\tau^i) = 0$, we have $\sum_{t \in \mathsf{SMALL}(T_i^{\text{IP}})} t \geq c(\tau^i) = 0$, and for any $\tau^i \in \tau^{\text{IP}}$ with $c(\tau^i) > 0$, we have $\sum_{t \in \mathsf{SMALL}(T_i^{\text{IP}})} t \geq 9E^2 - \sum_{t \in \mathsf{BIG}(T_i^{\text{IP}})} t \geq 9E^2 - 6E - \textsc{Size}(\tau^i) \geq c(\tau^i)$. The second inequality holds because the SIZE function rounds down all values, and there are at most $6E$ BIG values in each T_i^{IP}. By adding the above inequality for all $\tau^i \in \tau^{\text{IP}}$, we obtain (5) of the IP.

Since each T_i^{IP} is a subset of a valid part, its corresponding type τ^i has at most one value from each C_j. Therefore, for any $\tau^i \in \tau^{\text{IP}}$ with $\sigma^+(\tau^i) > 0$ we have, for $\sum_{t \in \mathsf{SMALL}(\tau^i \cap \mathscr{V}^g)} t \geq c(\tau^i) - \sum_{t \in \mathsf{SMALL}(\tau^i \cap \mathscr{V}^c)} t \geq c(\tau^i) - \sigma^- \geq \sigma^+(\tau^i)$. Moreover, for any $\tau^i \in \tau^{\text{IP}}$ with $\sigma^+(\tau^i) = 0$ we have, for $\sum_{t \in \mathsf{SMALL}(\tau^i \cap \mathscr{V}^g)} t \geq \sigma^+(\tau^i) = 0$. By adding the above inequality for all $T_i^{\text{IP}} \in T^{\text{IP}}$ we get constraint (5). Thus, T^{IP} is a solution of the IP. □

Lemma 3.7. *If the IP has a solution, then the allocation returned by* Exists-GC-MMS *is an allocation that gives every agent a bundle of value at least* $(1-\bar{\varepsilon})\mu$.

Proof. Let τ^{sol} be the solution of the IP and P^{sol} be the partition of the values formed by Exists-GC-MMS after finding τ^{sol}. We show that each part of P^{sol} has value at least $(9E^2 - 9E)$. From Lemma 3.4, we get that in $\mathscr{A}$, every agent gets a bundle of value at least $(1-\bar{\varepsilon})\mu$.

After assigning BIG values to P_i as per the type τ^i, suppose there are parts with value less than $9E^2 - 9E$.

Consider any such part P. The algorithm first adds SMALL values from $\mathscr{V}^g$. As τ^{sol} satisfies constraint (5) (a) of the IP, then $c(\tau(P)) \leq \sigma^+$. That is, the value to add to P so that the sum of values in P is at least $9E^2 - 9E$ is at most the sum of all SMALL values. We first add values from $\mathsf{SMALL}(\mathscr{V}^c)$. Suppose after receiving one value from each set in $\mathscr{V}^c$, P still has value less than $(9E^2 - 9E)$. As τ^{sol} satisfies constraint (b) of (5) of the IP, the total cost from $\mathsf{SMALL}(\mathscr{V}^c)$ for all parts together is at most σ^+. As the cost function is monotonic with number of parts, the total cost from $\mathsf{SMALL}(\mathscr{V}^c)$ for P also is at most $\mathsf{SMALL}(\mathscr{V}^c)$. Hence, there are enough values in $\mathsf{SMALL}(\mathscr{V}^g)$ to add to P_i to increase its value to at least $(9E^2 - 9E)$.

After adding values to P, its total value is at most $9E^2 - 6E$, as every item has value at most $3E$. Thus, the value added to it from SMALL values is at most $c(\tau(P))$. The total cost of the remaining parts is $\sum_{P' \neq P} c(\tau(P)) = \sum_{P \in P^{sol}} c(\tau(P)) - c(\tau(P)) \leq \sigma^+ -$ (the sum of SMALL values assigned to P), which is exactly the total value of unassigned SMALL values. Hence, constraint 5 (a) is satisfied for the smaller set $\tau^{sol} \backslash \tau(P)$. Similarly, we can show constraint 5 (b) also is satisfied. The initial constraints 2, 4 and 3 are satisfied for $\tau^{sol} \backslash \tau(P)$ by the validity of τ^{sol}. Hence $\tau^{sol} \backslash \tau(P)$ is a solution to the IP for the smaller case after removing P and its assigned values. By induction, we can assign values to every part until all parts are satisfied. Adding any unallocated values arbitrarily in Line 13 only increases the value of each bundle.

Hence, P^{sol} has every bundle of value at least $9E^2 - 9E$. From Lemma 3.4, the corresponding allocation $\mathscr{A}$ gives every agent a bundle of value at least $(1-\bar{\varepsilon})\mu$. □

Lemma 3.8. Exists-GC-MMS *runs in time* $O(mn)$.

Proof. The time to run Exists-GC-MMS is asymptotically equal to the time for constructing and solving the IP. Lenstra's algorithm (Lenstra Jr 1983) takes time exponential in the number of variables, $O(2^{1/\bar{\varepsilon}^2}) = O(2^{4/\varepsilon'^2})$ here, and polynomial in the largest coefficient of any variable in all inequalities, $m^+ + (n-1)m^- = O(mn)$ here. Note that σ and σ^+ are at most $n \cdot 9E^2$. Hence, the IP requires $O(2^{1/\varepsilon^2} mn) = O(mn)$ time. □

Lemmas 3.6, 3.7 and 3.8 together prove Theorem 3.2.

Acknowledgements

Rucha Kulkarni and Ruta Mehta thank the support of NSF Grant CCF-1750436 (CAREER). Setareh Taki is partially supported by NSF Grant CCF-1942321 (CAREER).

References

Amanatidis, G.; Markakis, E.; Nikzad, A.; and Saberi, A. 2017. Approximation Algorithms for Computing Maximin Share Allocations. *ACM Trans. Algorithms* 13(4): 52:1–52:28.

Barman, S.; and Krishna Murthy, S. K. 2017. Approximation algorithms for maximin fair division. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, 647–664. ACM.

Bouveret, S.; and Lemaître, M. 2016. Characterizing conflicts in fair division of indivisible goods using a scale of criteria. *Autonomous Agents and Multi-Agent Systems* 30(2): 259–290.

Brams, S. J.; and Taylor, A. D. 1996. *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press.

Budish, E. 2011. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy* 119(6): 1061–1103.

De La Vega, W. F.; and Lueker, G. S. 1981. Bin packing can be solved within 1+ ε in linear time. *Combinatorica* 1(4): 349–355.

Etkin, R.; Parekh, A.; and Tse, D. 2007. Spectrum sharing for unlicensed bands. *IEEE Journal on selected areas in communications* 25(3): 517–528.

Farhadi, A.; Ghodsi, M.; Hajiaghayi, M. T.; Lahaie, S.; Pennock, D. M.; Seddighin, M.; Seddighin, S.; and Yami, H. 2019. Fair Allocation of Indivisible Goods to Asymmetric Agents. *J. Artif. Intell. Res.* 64: 1–20.

Garg, J.; McGlaughlin, P.; and Taki, S. 2018. Approximating Maximin Share Allocations. In *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Garg, J.; and Taki, S. 2020. An Improved Approximation Algorithm for Maximin Shares. In *Proceedings of the 21st ACM Conference on Economics and Computation*, 379–380.

Ghodsi, M.; Hajiaghayi, M.; Seddighin, M.; Seddighin, S.; and Yami, H. 2018. Fair Allocation of Indivisible Goods: Improvements and Generalizations. In *Proceedings of the 2018 ACM Conference on Economics and Computation*.

Huang, X.; and Lu, P. 2019. An algorithmic framework for approximating maximin share allocation of chores. *CoRR* abs/1907.04505.

Jansen, K.; Klein, K.; and Verschae, J. 2016. Closing the Gap for Makespan Scheduling via Sparsification Techniques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP*, volume 55, 72:1–72:13.

Johnson, D. S. 1982. The NP-completeness column: An ongoing guide. *Journal of Algorithms* 3(4): 381–395.

Karmarkar, N.; and Karp, R. M. 1982. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 312–320. IEEE.

Kulkarni, R.; Mehta, R.; and Taki, S. 2020a. Approximating Maximin Shares with Mixed Manna. *CoRR* abs/2007.09133. URL https://arxiv.org/abs/2007.09133.

Kulkarni, R.; Mehta, R.; and Taki, S. 2020b. On the PTAS for Maximin Shares in an Indivisible Mixed Manna URL http://ruchark2.web.engr.illinois.edu/MMS_mixed_id.pdf.

Kurokawa, D.; Procaccia, A. D.; and Wang, J. 2016. When Can the Maximin Share Guarantee Be Guaranteed? In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, 523–529. AAAI Press.

Kurokawa, D.; Procaccia, A. D.; and Wang, J. 2018. Fair Enough: Guaranteeing Approximate Maximin Shares. *J. ACM* 65(2): 8:1–8:27.

Lenstra Jr, H. W. 1983. Integer programming with a fixed number of variables. *Mathematics of operations research* 8(4): 538–548.

Moulin, H. 2004. *Fair division and collective welfare*. MIT press.

Procaccia, A. D.; and Wang, J. 2014. Fair enough: Guaranteeing approximate maximin shares. In *Proceedings of the fifteenth ACM conference on Economics and computation*, 675–692. ACM.

Robertson, J.; and Webb, W. 1998. *Cake-cutting algorithms: Be fair if you can*. CRC Press.

Steinhaus, H. 1948. The problem of fair division. *Econometrica* 16: 101–104.

Vossen, T. 2002. *Fair allocation concepts in air traffic management*. Ph.D. thesis, Supervisor: MO Ball, University of Martyland, College Park, Md.

Woeginger, G. J. 1997. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters* 20(4): 149–154.