

Maximin Share Fair Allocation of Indivisible Chores: Beyond Additive Valuations*

Bo Li¹

Fangxiao Wang²

Yu Zhou¹

¹Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China

comp-bo.li@polyu.edu.hk, csyzhou@comp.polyu.edu.hk

² School of Information and Communication Engineering, UESTC, Chengdu, China

2018011216019@std.uestc.edu.cn

April 29, 2022

Abstract

In this work, we study the maximin share (MMS) fair allocation of indivisible chores. For additive valuations, Huang and Lu [EC, 2021] designed an algorithm to compute a $11/9$ -approximate MMS fair allocation, and Feige et al. [WINE, 2021] proved that no algorithm can achieve better than $44/43$ approximation. Beyond additive valuations, unlike the allocation of goods, very little is known. We first prove that for submodular valuations, in contrast to the allocation of goods where constant approximations are proved by Barman and Krishnamurthy [TEAC, 2020] and Ghodsi et al [AIJ, 2022], the best possible approximation ratio is n . We then focus on two concrete settings where the valuations are combinatorial. In the first setting, agents need to use bins to pack a set of items where the items may have different sizes to different agents and the agents want to use as few bins as possible to pack the items assigned to her. In the second setting, each agent has a set of machines that can be used to process a set of items, and the objective is to minimize the makespan of processing the items assigned to her. For both settings, we design constant approximation algorithms.

Key Words: Indivisible Chores, Maximin Share Fairness, Subadditive Valuations, Bin Packing, Job Scheduling.

1 Introduction

Since the introduction of maximin share (MMS) fairness by Budish (2011), MMS has been extensively used as a criterion to evaluate the fairness of an allocation when m indivisible items are allocated to n heterogeneous agents. MMS fairness is traditionally defined for the allocation of goods as a relaxation of proportionality (PROP). A PROP allocation requires that the utility of each agent is no smaller than the average utility when all items are assigned to her. PROP is too demanding when the items are indivisible since such an allocation does not exist even when there is a single item and

*The authors are ordered alphabetically. Part of this work was done when the second author was a research assistant with The Hong Kong Polytechnic University.

two agents; the agent who is not assigned the item has utility 0 which is arbitrarily smaller than her proportionality. Motivated by this strong impossibility result, Budish (2011) proposed to use MMS to replace proportionality. Informally, the maximin share (MMS) of an agent is the maximum value she can guarantee if she is to partition the items into n bundles but is the last one to pick a bundle. An allocation is MMS fair if every agent's value is no smaller than her MMS.

For the allocation of goods, although MMS significantly weaken the fairness requirement, it is first shown by Kurokawa et al. (2018) that there are instances where no allocation is MMS fair to all agents. Accordingly, designing (efficient) algorithms to compute approximately MMS fair allocations steps into the center of the field of algorithmic fair division. Kurokawa et al. (2018) first proved there exists a $2/3$ -approximate MMS fair allocation for additive valuations, and then Amanatidis et al. (2017) designed a polynomial time algorithm with the same approximation guarantee. Later, Ghodsi et al. (2021) improved the approximation ratio to $3/4$, and Garg and Taki (2021) further improved it to $3/4 + 1/(12n)$. On the negative side, Feige et al. (2021) proved that no algorithm can ensure better than $39/40$ approximation. Beyond additive valuations, Barman and Krishnamurthy (2020) initiated the study of approximate MMS fair allocation with submodular valuations, and proved that a 0.21 -approximate MMS fair allocation can be computed by the round-robin algorithm. Ghodsi et al. (2022) improved the approximation ratio to $1/3$, and moreover, they gave constant and logarithmic approximation guarantees for XOS and subadditive valuations, respectively. The approximations for XOS and subadditive valuations are recently improved in Seddighin and Seddighin (2022).

As we have seen, the majority of efforts is devoted to the allocation of indivisible goods, and the parallel problem of chores (where agents have costs or dis-utilities to complete the received items) has received less attention. Aziz et al. (2017) first pointed out this issue, and proved that the round-robin algorithm ensures 2 approximation for additive valuations for indivisible chores. Barman and Krishnamurthy (2020) and Huang and Lu (2021) respectively improved the approximation ratio to $4/3$ and $11/9$. Recently, Feige et al. (2021) proved that with additive valuations, no algorithm can beat the approximation of $44/43$. However, very little is known beyond additive valuations. In this work, we aim at understanding the extent to which MMS fairness can be satisfied when the valuations are not additive. Our main results are summarized as follows.

1.1 Main Results

In this work, we initiate the study of MMS fair allocation of indivisible chores with subadditive valuations. We first observe that it is simple to achieve n -approximate MMS fairness by allocating all items to one agent. Surprisingly, we show that n -approximation is the best possible even when the valuations are submodular. This result exhibits a significant difference between the allocations of chores and goods, since for goods with submodular (even XoS) valuations, we always have constant-approximate MMS fair allocations (Barman and Krishnamurthy, 2020; Ghodsi et al., 2022).

Result 1 For any number n of agents, there is an instance with submodular valuations where no allocation can be better than n -approximate MMS fair.

Due to this strong impossibility result, we turn to study two concrete settings where the agents have specific combinatorial valuations. The first setting deals with a bin packing problem, where the items have sizes and the sizes can be different to different agents. The agents have bins that can be used to pack the items allocated to them with the goal of using as few bins as possible. We call this valuation *bin-packing*. For bin-packing valuations, we show that no algorithm can be better than 2 -approximation, and design a polynomial time algorithm that returns a 2 -approximate MMS fair allocation. Moreover, the approximation can be improved to $3/2$ when the maximin share gets large.

Result 2 With bin-packing valuations, the best possible approximation of MMS fairness is 2, and an allocation matching this approximation ratio can be found in polynomial time.

The second setting deals with a job scheduling problem, where a set of jobs needs to be processed by the agents. The agents are heterogeneous and thus each job may be of different lengths to different agents. Each agent controls a set of machines with possibly different speeds. Upon receiving a set of jobs, an agent's value is determined by the corresponding minimum completion time of processing the jobs using her own machines (i.e., makespan). The corresponding valuation is called *job-scheduling*. As will be clear, job-scheduling is a generalized setting of additive valuations, which takes group-wise fairness into consideration.

Result 3 With job-scheduling valuations, a 2-approximate MMS fair allocation can be found in polynomial time.

1.2 More Related Works

We refer the readers to Amanatidis et al. (2022); Aziz et al. (2022) for detailed surveys on fair division of indivisible items. Besides MMS fairness, proportionality up to one item (PROP1) and proportionality up to any item (PROPX) are also studied as relaxations of proportionality. For goods, PROP1 allocations always exist but PROPX allocations may not exist Barman and Krishnamurthy (2019); Brânzei and Sandomirskiy (2019); Aziz et al. (2020). For chores, both PROP1 and PROPX allocations exist and can be found efficiently Moulin (2018); Li et al. (2022). As far as we know, all the above works assume the agents have additive valuations, and little is known beyond additive valuations. In another parallel line of research envy-freeness and its relaxations, namely envy-free up to one item (EF1) and envy-free up to one any item (EFX), are very widely studied. It is shown in Lipton et al. (2004) and Bhaskar et al. (2021) for goods and chores, respectively, that an EF1 allocation exists for the monotone combinatorial valuations. However, the existence of EFX allocations is still unknown even with additive valuations. Therefore, approximation algorithms are proposed in Amanatidis et al. (2020); Zhou and Wu (2021) for additive valuations and in Plaut and Roughgarden (2020); Chan et al. (2019) for subadditive valuations.

2 Preliminaries

For any integer $k \geq 1$, let $[k] = \{1, \dots, k\}$. In a fair allocation instance, we have n agents denoted by $N = [n]$ and m items denoted by $M = [m]$. Each agent i has a valuation function over the items, $v_i : 2^M \rightarrow \mathbb{R}^+ \cup \{0\}$. The items are chores, and particularly, upon receiving $S \subseteq M$, $v_i(S)$ represents the effort or cost agent i needs to spend on completing the chores in S . The valuation functions are normalized and monotone, i.e., $v_i(\emptyset) = 0$ and $v_i(S_1) \leq v_i(S_2)$ for any $S_1 \subseteq S_2 \subseteq M$. Note that no bounded approximation can be achieved for general valuation functions, and we provide one such example at the end of this section. Thus we restrict our attention to the following three classes.

- Subadditive: A valuation function v_i is subadditive if for any $S_1, S_2 \subseteq M$, we have

$$v_i(S_1 \cup S_2) \leq v_i(S_1) + v_i(S_2).$$

- Submodular: A valuation function v_i is submodular if for any $S_1 \subseteq S_2 \subseteq M$ and $e \in M \setminus S_2$, we have

$$v_i(S_2 \cup \{e\}) - v_i(S_2) \leq v_i(S_1 \cup \{e\}) - v_i(S_1).$$

- Additive: A valuation function v_i is additive if for any two sets $S \subseteq M$, we have

$$v_i(S) = \sum_{j \in M} v_i(\{j\}).$$

It is widely known that any additive valuation function is also submodular, and any submodular valuation function is also subadditive.

An allocation is an n -partition of the items $\mathbf{X} = (X_1, \dots, X_n)$ where X_i contains the items allocated to agent i such that $X_i \cap X_j = \emptyset$ and $\bigcup_{i \in N} X_i = M$. For any set S and integer k , let $\Pi_k(S)$ be the set of all k -partitions of S . Then the maximin share (MMS_i) of agent i is defined as

$$\text{MMS}_i = \min_{\mathbf{X} \in \Pi_n(M)} \max_{j \in N} v_i(X_j).$$

Note that the computation of MMS_i is NP-hard even when the valuations are additive, which can be verified by a reduction from the Partition problem. Given an n -partition of M , $\mathbf{X} = (X_1, \dots, X_n)$, if $v_i(X_i) \leq \text{MMS}_i$ for all i , then $\mathbf{X}$ is called an *MMS-defining partition* for agent i . Note that the original definition of MMS_i for chores is defined with non-positive valuations, where the minimum valued bundle is maximized. In this work, to simplify the notions, we choose to use non-negative numbers (representing costs), and thus the definition is equivalently changed to be the maximum valued bundle is minimized. To be consistent with the literature, we still call it maximin share.

Definition 1 (α -MMS) An allocation $\mathbf{X} = (X_1, \dots, X_n)$ is α -approximate maximin share (α -MMS) fair if $v_i(X_i) \leq \alpha \cdot \text{MMS}_i$ for all agents $i \in N$. The allocation is MMS fair if $\alpha = 1$.

Given the definition of MMS, we have the following simple properties.

Lemma 1 For any agent i with subadditive valuation $v_i(\cdot)$, we have the following bounds for MMS_i ,

1. $\text{MMS}_i \geq \max_{j \in M} v_i(\{j\});$
2. $\text{MMS}_i \geq \frac{1}{n} \cdot v_i(M).$

Proof The first claim is straightforward, since in any MMS-defining partition of agent i , the item with largest value must appear in one of the bundles, and by the monotonicity of the valuations, the value of this bundle is at least $\max_{j \in M} v_i(\{j\})$.

To see the second claim, let $\mathbf{X} = (X_1, \dots, X_n)$ be an MMS-defining partition for agent i , then $v_i(X_j) \leq \text{MMS}_i$ for all j . Summing up all these inequalities, we have

$$v_i(M) \leq \sum_{j \in N} v_i(X_j) \leq n \cdot \text{MMS}_i,$$

where the first inequality is by the subadditivity of the valuations. ■

Before the end of this section, we present an example with non-subadditive valuations where the approximation ratio is unbounded. Note that there exist simpler examples, but we choose the following one because it represents a particular combinatorial structure – minimum spanning tree.

Example Let $G = (V, E)$ be a graph shown in the left sub-figure of Figure 1, where the vertices V are the items that are to be allocated, i.e., $M = V$. There are two agents $N = \{1, 2\}$ who possess different weights on the edges as shown in the middle and right sub-figures of Figure 1. The valuation functions are measured by the minimum spanning tree in their received subgraphs. Particularly, for any $S \subseteq V$, $v_i(S)$ equals the weight of the minimum spanning tree on $G[S]$ – the induced subgraph of S in G – under agent i 's weights. Thus, $\text{MMS}_i = 0$, for both $i = 1, 2$, where an MMS-defining partition for agent 1 is $\{v_1, v_2\}$ and $\{v_3, v_4\}$ and that for agent 2 is $\{v_1, v_4\}$ and $\{v_2, v_3\}$. However, it can be verified that no matter how the vertices are allocated to the agents, there is one agent whose value is at least 1, which implies that no bounded approximation is possible for general valuations.

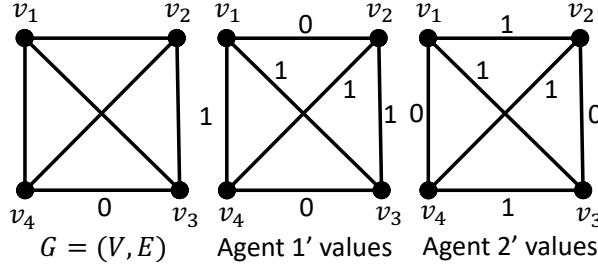


Figure 1: An Instance with Unbounded Approximation Ratio

3 Subadditive Valuations

By the second claim of Lemma 1, if the valuations are subadditive, allocating all items to a single agent ensures an approximation of n . We call this algorithm All-or-Nothing, which is somewhat the most unfair algorithm. Surprisingly, such an unfair algorithm achieves the optimal approximation ratio of MMS even if the valuations are submodular.

Theorem 1 *For any $n \geq 2$, there is an instance with submodular valuations for which no allocation is better than n -MMS.*

Proof For any fixed $n \geq 2$, we construct the following instance with n agents and $m = n^n$ items via an n -dimensional coordinate system. Let each item correspond to a point in the system and

$$M = \{(x_1, x_2, \dots, x_n) \mid x_i \in [n] \text{ for all } i \in [n]\}.$$

For each agent $i \in [n]$, we define n covering planes $\{C_{il}\}_{l \in [n]}$ and for each $l \in [n]$,

$$C_{il} = \{(x_1, x_2, \dots, x_n) \mid x_i = l \text{ and } x_j \in [n] \text{ for all } j \in [n] \setminus \{i\}\}.$$

Note that $\{C_{il}\}_{l \in [n]}$ forms an exact cover of the points in M , i.e., $\bigcup_l C_{il} = M$ and $C_{il} \cap C_{iz} = \emptyset$ for all $l \neq z$. For any set of items $S \subseteq M$, $v_i(S)$ equals the minimum number of planes in $\{C_{il}\}_{l \in [n]}$ that can cover S . Therefore, $v_i(S) \in [n]$ for all S . We first show $v_i(\cdot)$ is submodular for every i . For any $S \subseteq T \subseteq M$ and any $e \in M \setminus T$, if e is not in the same covering plane for any point in T , e is not in the same covering plane for any point in S . Thus, $v_i(T \cup \{e\}) - v_i(T) = 1$ implies $v_i(S \cup \{e\}) - v_i(S) = 1$, and accordingly,

$$v_i(T \cup \{e\}) - v_i(T) \leq v_i(S \cup \{e\}) - v_i(S).$$

Since $\{C_{il}\}_{l \in [n]}$ is an exact cover of M , $\text{MMS}_i = 1$ where the MMS defining partition is simply $\{C_{il}\}_{l \in [n]}$. Then to prove the theorem, it suffices to show that for any allocation of M , there is at least one agent whose value is n . For the sake of contradiction, we assume there is an allocation $\mathbf{X} = (X_1, \dots, X_n)$ where every agent has value at most $n - 1$, which means for every agent $i \in [n]$, there exists a plane C_{il_i} with $l_i \in [n]$ such that $X_i \cap C_{il_i} = \emptyset$. Consider the point $\mathbf{b} = (l_1, l_2, \dots, l_n)$. It is clear that $\mathbf{b} \in C_{il_i}$ and thus $\mathbf{b} \notin X_i$ for all i . Thus $\mathbf{b}$ is not allocated to any agent, which is a contradiction with $\mathbf{X}$ being an allocation and thus there must be an agent whose value is n . ■

4 Bin Packing Model

Although Theorem 1 shows that there is no algorithm that can be better than n -MMS for all submodular valuations, it does not wipe out the possibility of beating n -approximation for specific subadditive valuations. In this section and the next, we propose two models where agents have concrete combinatorial valuations and design constant-approximate MMS fair algorithms.

The first model encodes a bin packing problem, where the items have sizes and need to be packed into bins by the agents. Specifically, each item $j \in M$ has size $s_{i,j} \geq 0$ to agent i , and note that j may be of different sizes to different agents. For a set of items S , $s_i(S) = \sum_{j \in S} s_{i,j}$. Each agent $i \in N$ has unlimited number of bins with the same capacity c_i , where it is assumed that $c_i \geq \max_{j \in M} s_{i,j}$. Given a set of items $S \subseteq M$, the value of agent i , $v_i(S)$, is determined by the minimum number of bins (with capacity c_i) that can pack all items in S , which involves solving a classic bin packing problem and thus the computation of $v_i(S)$ is NP-hard. Accordingly, MMS_i is essentially the minimum number k_i such that the items can be partitioned into n bundles and the items in each bundle can be packed into no more than k_i bins. Then we have the following simple lemma.

Lemma 2 $\text{MMS}_i \cdot c_i \geq \frac{s_i(M)}{n}$ for every $i \in N$

Proof From the definition of maximin share, we know that each agent i can use at most $n \cdot \text{MMS}_i$ bins to pack all the items. This implies that the total capacity of $n \cdot \text{MMS}_i$ bins is no smaller than the total size of all the items. That is, $n \cdot \text{MMS}_i \cdot c_i \geq s_i(M)$, which completes the proof. ■

4.1 Lower Bound

We first prove the lower bound for the bin-packing model.

Theorem 2 *For the bin-packing model, no algorithm performs better than 2-MMS.*

Proof We first recall the impossibility instance given by Feige et al. (2021). In this instance there are three agents and nine items as arranged in a three by three matrix. The three agents' valuations are shown in the matrices V_1, V_2 and V_3 .

$$V_1 = \begin{pmatrix} 6 & 15 & 22 \\ 26 & 10 & 7 \\ 12 & 19 & 12 \end{pmatrix} \quad V_2 = \begin{pmatrix} 6 & 15 & 23 \\ 26 & 10 & 8 \\ 11 & 18 & 12 \end{pmatrix} \quad V_3 = \begin{pmatrix} 6 & 16 & 22 \\ 27 & 10 & 7 \\ 11 & 18 & 12 \end{pmatrix}$$

Feige et al. (2021) proved that for this instance the MMS value of every agent is 43, however, in any allocation, at least one of the three agents gets value no smaller than 44.

We can adapt this instance to the bin-packing model and obtain a lower bound of 2. In particular, we also have three agents and nine items. The numbers in matrices V_1, V_2 and V_3 are the sizes of the items to each agent 1, 2 and 3. Let the capacities of the bins of each agent be $c_i = 43$ for all i . Accordingly, we have $\text{MMS}_i = 1$. Since in any allocation, there is at least one agent who gets items with total size no smaller than 44, for this agent, she has to use 2 bins to pack the assigned items, which finishes the proof. ■

4.2 Upper Bound

Next, we design an algorithm to compute an allocation that matches the approximation 2, and thus is the best possible approximation algorithm. Note that although the computation of each MMS_i and $v_i(S)$ is NP-hard, our algorithm runs in polynomial time without explicitly computing these values.

We first show that if we can solve the problem for the special case where all agents have the same order of items by ranking their sizes, we can also solve the general case. Similar lemmas have been widely used for designing algorithms to compute MMS fair allocations with additive valuations Barman and Krishnamurthy (2020); Huang and Lu (2021). Although this lemma does not hold for arbitrary subadditive valuations, it holds for the bin-packing model.

We call a bin-packing instance *identical ordering* (IDO) if $s_{i,1} \geq s_{i,2} \geq \dots \geq s_{i,m}$ holds for every agent i . Note that although the agents have the same order of items by their sizes, the cardinal numbers can still be significantly different. Given a general bin-packing instance $I = (N, M, \{v_i\}_{i \in N}, \{s_i\}_{i \in N})$, we can construct an IDO instance as follows. First, note that for every agent i , there exists a permutation $\sigma_i : [m] \rightarrow [m]$ such that for all $g, g' \in [m]$ with $g < g'$, $s_{i,\sigma_i(g)} \geq s_{i,\sigma_i(g')}$. Using these permutations, we can construct the IDO instance by setting $s'_{i,g} = s_{i,\sigma_i(g)}$ for every agent $i \in N$ and item $g \in M$. In other words, for every agent i , the size of the g -th item in the IDO instance is equal to the g -th largest size of items in the original instance. Note that the construction of the IDO instance runs in $O(nm \log m)$ time.

Lemma 3 *If there exists an allocation $\mathcal{A}' = (A'_1, \dots, A'_n)$ in the IDO instance I' , such that $v'_i(A'_i) \leq \alpha \cdot \text{MMS}_i$ for all $i \in N$, then there exists an allocation $\mathcal{A} = (A_1, A_2, \dots, A_n)$ in the original instance I such that $v_i(A_i) \leq \alpha \cdot \text{MMS}_i$ for all $i \in N$.*

Next, we present the main algorithm (Algorithm 1) that computes 2-MMS allocations for IDO bin-packing instances. The basic idea of the algorithm is similar to that of the bag filling algorithm, i.e., incrementally adding small items into a bag until the bag is large enough for all the agents, and allocating the bag to one of the agents who are the last to think the bag large enough. The major differences include the way we distinguish between small and large items, and the way we initialize and fill the bag. In the following, we prove the approximation of 2, and in Appendix A, we show how to modify Algorithm 1 so that the approximation ratio improves to $3/2$ when MMS values get large.

Theorem 3 *Algorithm 1 computes a 2-MMS allocation for all IDO bin-packing instances in polynomial time.*

Before the proof of Theorem 3, we first introduce some notations and technical lemmas. In the bin-packing model, we say an item j is large for an agent i if the size of the item exceeds half the capacity of the agent's bins, i.e., $s_{i,j} > 1/2 \cdot c_i$; Otherwise, we say item j is small for agent i . We denote by H_i and L_i respectively the sets of large and small items for agent i . Formally,

$$H_i = \{j \in M \mid s_{i,j} > \frac{c_i}{2}\}, \quad L_i = \{j \in M \mid s_{i,j} \leq \frac{c_i}{2}\}.$$

We also split all the items into $\lceil \frac{m}{n} \rceil$ sets, where the first set contains the n largest items, the second set contains the next n largest items, and so on. That is,

$$G_i = \{(i-1)n + 1, \dots, i \cdot n\} \text{ for } i \in [1, \lceil \frac{m}{n} \rceil], \quad G_{\lceil \frac{m}{n} \rceil} = \{\lfloor \frac{m}{n} \rfloor \cdot n + 1, \dots, m\}.$$

Note that G_i s are well-defined since the instances we consider here are identical ordering.

Algorithm 1: Modified Bag Filling

```

1 Input: An IDO instance over  $n$  agents and  $m$  indivisible chores with size  $s_{i,j}$  for each agent
    $i \in N$  and chore  $j \in M$ . The valuation  $v_i$  for each agent  $i$  is determined by the bin packing
   problem, which is non-negative, monotone, and subadditive.
2 Output: An allocation  $\mathcal{A} = (A_1, A_2, \dots, A_n)$  such that  $v_i(A_i) \leq 2 \cdot \text{MMS}_i$  for every  $i \in N$ .
3 while  $|N| > 1$  do
4   Initialize  $B = \emptyset$ ,  $k = \max\{i \in [1, \lceil \frac{m}{n} \rceil] \mid \exists j \in N : G_i \cap H_j \neq \emptyset\}$ .
5   for  $i = 1$  to  $k$  do
6     Choose the largest item  $g$  in  $G_i$ .
7      $B \leftarrow B \cup \{g\}$ ,  $G_i \leftarrow G_i \setminus \{g\}$ .
8   end
9   if  $s_i(B) > \frac{s_i(M)}{n}$  for every agent  $i$  in  $N$  then
10    Arbitrarily choose one agent  $j$  in  $N$  who thinks the smallest item in  $B$  is large.
11     $A_j \leftarrow B$ ,  $N \leftarrow N \setminus \{j\}$ .
12  end
13  else
14    while  $s_i(B) \leq \frac{s_i(M)}{n}$  for some agent  $i \in N$  and  $L_i \neq \emptyset$  do
15      Choose the smallest item  $g$  in  $L_i$ .
16       $B \leftarrow B \cup \{g\}$ ,  $L_i \leftarrow L_i \setminus \{g\}$ .
17      if  $s_j(B) > \frac{s_j(M)}{n}$  for every agent  $j$  in  $N$  then
18         $A_i \leftarrow B$ ,  $N \leftarrow N \setminus \{i\}$ .
19      end
20    end
21  end
22  if  $s_i(B) \leq \frac{s_i(M)}{n}$  for some agent  $i$  in  $N$  then
23     $A_i \leftarrow B$ ,  $N \leftarrow N \setminus \{i\}$ .
24  end
25 end
26 Allocate the remaining items to the last agent.

```

The algorithm consists of two procedures, i.e, bag initialization (Steps 4 to 11) and bag filling (Steps 13 through 23). In the bag initialization procedure, for each G_i that contains large items, its largest item (which must be large for some agent) is put into the bag. This guarantees that the bag contains enough large items for every agent while some agent can use no more than MMS bins to pack all the items in it. If all the agents think the bag large enough, the bag will be allocated to one of the agents who think the smallest item in the bag is large. Otherwise, the algorithm enters the next procedure.

In the bag filling procedure, if there exists an agent who thinks the bag not large enough, she will incrementally fill the bag with her small items until the bag becomes large enough for her or she has

no small items. If all the agents think the bag large enough, the bag will be allocated to the agent who adds the last item into the bag. If some agents think the bag not large enough while none of them have small items, the bag will be arbitrarily allocated to one of these agents.

Lemma 4 *The agent who is allocated a bag during the bag initialization procedure can use no more than MMS bins to pack the items in the bag.*

Proof Clearly, all the items in the bag are large for the agent i who receives the bag, since the smallest one is already large for her. From the definition of maximin share, we know that agent i can use at most $n \cdot MMS_i$ bins to pack all the items in M . This implies that there are at most $n \cdot MMS_i$ large items for agent i since each bin can hold at most one large item. From the structure of G s, we know that one of every n such large items is put into the bag. Therefore, at most MMS_i large items of agent i are put into the bag, which require no more than MMS_i bins to pack, thus completing the proof. ■

Lemma 5 *The agent who is allocated a bag during the bag filling procedure can use no more than $2MMS$ bins to pack the items in the bag.*

Proof Before proving the lemma, we first show that if the total size of items for an agent i is at most $\frac{s_i(M)}{n}$, the agent can use no more than $2MMS_i$ bins to pack all the items. This is because in the worst case where only half capacity of each bin is taken up, the $2MMS_i$ bins can still hold items with total size of $MMS_i \cdot c_i$, which is at least $\frac{s_i(M)}{n}$ according to Lemma 2. This also implies that at least one bin is taken up no more than half of its capacity.

Now, we're ready to prove Lemma 5. There are two cases where the agent i is allocated the bag B . First, after agent i adds a small item into the bag, all the agents think the bag large enough. Denote g as the item added into the bag, we have $s_i(B \setminus \{g\}) \leq \frac{s_i(M)}{n}$ and $s_i(\{g\}) \leq \frac{c_i}{2}$. As we've shown, these items can be packed using $2MMS_i$ bins. Second, agent i thinks the bag not large enough (i.e., $s_i(B) \leq \frac{s_i(M)}{n}$) while she has no small items. Clearly, she can also use $2MMS_i$ bins to pack all the items, which completes the proof. ■

Now, we are ready to prove Theorem 3.

Proof of Theorem 3 Clearly, Algorithm 1 runs in polynomial time. For all the agents except the last one, Lemma 4 and Lemma 5 have proved that they can use no more than $2MMS$ bins to pack all the items allocated to them. To complete the proof, we now show that the last agent i can use no more than $2MMS_i$ bins to pack all the remaining items.

Considering two cases, first, agent i thinks all the bags allocated to other agents are large enough, i.e., $s_i(A_j) > \frac{s_i(M)}{n}$ for all $j \in N \setminus i$. Denote the remaining items by M' , then we have

$$s_i(M') = s_i(M) - \sum_{j \in N \setminus i} s_i(A_j) < s_i(M) - (n-1) \frac{s_i(M)}{n} = \frac{s_i(M)}{n}.$$

As proved in Lemma 5, agent i can use no more than $2MMS_i$ bins to pack all the items in M' . Second, at least one bag allocated to other agents is not large enough for agent i , i.e., $s_i(A_j) \leq \frac{s_i(M)}{n}$ for some $j \in N \setminus i$. This implies that before A_j is allocated, there is already no small item for agent i . Therefore, all the remaining items are large for agent i . From the structure of G s, we know that at most MMS_i of agent i 's large items are left to the last round. Therefore, in this case, agent i can use no more than MMS_i bins to pack all the remaining large items. ■

5 Job Scheduling Model

Our second model considers a job scheduling environment, where the items are jobs that need to be processed by the agents. Similar to the bin packing model, each item $j \in M$ has size $s_{i,j} \geq 0$ to agent $i \in N$. For a set of items S , $s_i(S) = \sum_{j \in S} s_{i,j}$. And each agent $i \in N$ exclusively controls a set of k_i machines $P_i = [k_i]$ with possibly different speed $\rho_{i,j}$ for $j \in P_i$. Upon receiving a set of items $S \subseteq M$, agent i 's value, $v_i(S)$, is determined by the minimum competition time of processing S using her own machines P_i (i.e., *makespan*). Formally,

$$v_i(S) = \min_{(T_1, \dots, T_{k_i}) \in \Pi_{k_i}(S)} \max_{l \in [k_i]} \frac{\sum_{j \in T_l} s_{i,j}}{\rho_{i,l}}.$$

Note that the computation of $v_i(S)$ is NP-hard. Intuitively, the value of MMS_i is obtained by partitioning the items into $n \cdot k_i$ bundles, and allocating to k_i different types of machines (with different speeds) where each type has n identical machines so that the makespan is minimized.¹ Note that when each agent controls a single machine, i.e., $k_i = 1$ for all i , the problem degenerates to the additive valuation case, and thus by Feige et al. (2021) no algorithm can be better than $44/43$ -approximation.

Next we focus on the upper bound. We can prove a result similar with Lemma 3 in the bin packing model. That is, it suffices for us to design algorithms for the IDO instances where $s_{i,1} \geq s_{i,2} \geq \dots \geq s_{i,m}$ holds for any agent i , and the algorithms can be easily converted to handle the general case by the reduction similar to Algorithm 6 (presented in the appendix). Since the proof is almost the same as that of Lemma 3, we omit the details to avoid redundancy.

Now, we can focus on computing approximate MMS allocations for IDO instances of the job scheduling model. We allocate jobs among agents in a round-robin fashion (Algorithm 2): the agents take turns picking the largest job among the remaining jobs until no job remains. The algorithm is quite direct but surprisingly returns us a 2-MMS allocation. The technical contribution here is to show the returned allocation is indeed a 2-MMS allocation. That is, each agent i can distribute all the jobs allocated to her among her machines while ensuring that the maximum completion time doesn't exceed $2 \cdot \text{MMS}_i$.

Theorem 4 *Algorithm 2 computes a 2-MMS allocation for every IDO job-scheduling instance in polynomial time.*

To prove Theorem 4, we present an algorithm (Algorithm 3) for each agent to distribute the jobs allocated to her among her machines. Here, we consider the capacity of a machine instead of its speed. Specifically, for each agent $i \in N$ and each machine $j \in P_i$, we set the capacity of the machine $c_{i,j} = \text{MMS}_i \cdot \rho_{i,j}$. Each agent assigns as much large jobs as possible to her machines with larger capacities, while ensuring that the workload (i.e., total size of the jobs) on each machine is no more than twice its capacity. In this way, for each agent i , the maximum completion time (i.e., *makespan*) of her machines doesn't exceed $2 \cdot \text{MMS}_i$. Therefore, all we need to do now is to show that for each agent, all the jobs allocated to her are distributed to her machines at the end of Algorithm 3.

¹An alternative way to explain the scheduling model is to view each agent i as a group of k_i small agents and the value of MMS_i as the *collective maximin share* for these k_i small agents in the group. We believe this notion of collective maximin share is of independent interest as a groupwise fairness notion. We remark that this notion is different with the groupwise (and pairwise) maximin share defined in Barman et al. (2018) and Caragiannis et al. (2019), where the max-min value is defined for each single agent. In our definition, however, a set of agents share the same value for the items assigned to them.

Algorithm 2: Round-Robin

1 **Input:** An IDO instance over n agents and m indivisible jobs with size $s_{i,j}$ for each agent $i \in N$ and job $j \in M$. The valuation v_i for each agent i is determined by the job scheduling problem, which is non-negative, monotone, and subadditive.
2 **Output:** An allocation $\mathcal{A} = (A_1, A_2, \dots, A_n)$ such that $v_i(A_i) \leq 2 \cdot \text{MMS}_i$ for every $i \in N$.
3 Initialize allocation $\mathcal{A} = (A_1, \dots, A_n)$ with $A_i = \emptyset$ for every $i \in N$.
4 **while** $M \neq \emptyset$ **do**
5 **for** $i = 1$ **to** m **do**
6 Pick $g \in \arg \max_{g' \in M} s_{i,g'}$.
7 $A_i \leftarrow A_i \cup \{g\}$, $M \leftarrow M \setminus \{g\}$.
8 **end**
9 **end**

Lemma 6 *For each agent i , Algorithm 3 allocates all the jobs in A_i among the machines P_i .*

Proof of Lemma 6 We will prove the lemma for an agent i , an analogous proof can be established for any other agent. Recall that A_i is the set of jobs that Algorithm 2 allocates to agent i , $P_i = \{1, 2, \dots, k_i\}$ is the set of agent i 's machines. We assume, for sake of contradiction that A_i is not empty after allocating the jobs A_i to the machines P_i by Algorithm 3. We will show that this assumption contradicts the definition of maximin share.

We denote $T = (T_1, T_2, \dots, T_{k_i})$ as the allocation returned by Algorithm 3 and $t_{k_i+1}^*$ as the largest job that is not allocated. We use t_j^* to denote the largest job in T_j for $j \in [1, k_i]$, then we can define $T' = (T'_1, T'_2, \dots, T'_{k_i})$, where $T'_j = T_j \setminus t_j^* \cup t_{j+1}^*$ for $j \in [1, k_i]$. In the partition that defines agent i 's maximin share, every job $g \in M$ must be assigned to one machine j . We assume, without loss of generality that $c_{i,1} \geq c_{i,2} \geq \dots \geq c_{i,k_i}$, then we have $s_{i,g} \leq c_{i,j} \leq c_{i,1}$ for every $g \in M$. From Algorithm 3, we know that $s_i(T_1 \cup t_2^*) > 2c_{i,1}$, then we have $s_i(T'_1) = s_i(T_1 \setminus t_1^* \cup t_2^*) > c_{i,1}$.

We assume as our induction hypothesis that $s_{i,t_j^*} \leq c_{i,j}$ for $j \in [1, l]$ where l is less than k_i . Similar to the claim that $s_i(T'_1) > c_{i,1}$, we have $s_i(T'_j) > c_{i,j}$ for every $j \in [1, l]$. Summing these l inequalities, we have $\sum_{j=1}^l s_i(T'_j) > \sum_{j=1}^l c_{i,j}$. Since Algorithm 2 allocates jobs to agents in a round-robin fashion, we know that for any job g in $\cup_{j=1}^l T'_j$, there are n jobs right before g in M (i.e., jobs $g-n, g-n+1, \dots, g-1$) that are not smaller than g . Therefore, denote S as the set of jobs in M that are before job t_{l+1}^* , we have

$$s_i(S) \geq \sum_{g \in \cup_{j=1}^l T'_j} n \cdot s_{i,g} = n \cdot \sum_{j=1}^l s_i(T'_j) > n \cdot \sum_{j=1}^l c_{i,j}.$$

Due to the pigeonhole principle, in the partition that defines agent i 's maximin share, at least one bundle contains a subset of S whose total size is greater than $\sum_{j=1}^l c_{i,j}$. Denote S' as the subset of S in such a bundle, we know that at least a job g in S' is assigned to a machine j with $j \geq l+1$, since the completion time of processing the jobs in S' using only the first l machines must exceed MMS_i . Hence, we have $s_{i,g} \leq c_{i,j} \leq c_{i,l+1}$. Since any job in S is not smaller than job t_{l+1}^* , we have $s_{i,t_{l+1}^*} \leq s_{i,g} \leq c_{i,l+1}$.

Therefore, we have $s_{i,t_j^*} \leq c_{i,j}$ for every $j \in [1, k_i]$. Similarly, we have $s_i(T'_j) > c_{i,j}$ for every $j \in [1, k_i]$ and $s_i(M) > n \cdot \sum_{j=1}^{k_i} c_{i,j}$. Due to the pigeonhole principle, in the partition that defines

agent i 's maximin share, the size of at least one bundle exceeds $\sum_{j=1}^{k_i} c_{i,j}$. This contradicts with the definition of maximin share, since the completion time of processing the jobs in such a bundle must exceed MMS_i . ■

Algorithm 3: Job Scheduling

```

1 Input: A set of jobs  $A = (1, 2, \dots, t)$  with sizes  $s_1 \geq s_2 \geq \dots \geq s_t$ , and a set of machines
    $P = (1, 2, \dots, k)$  with capacities  $c_1 \geq c_2 \geq \dots \geq c_k$ .
2 Output: An allocation  $T = (T_1, T_2, \dots, T_k)$  of the jobs among the machines, the set of
   remaining jobs  $G$ .
3 Initialize  $T = (T_1, T_2, \dots, T_k)$  with  $T_i = \emptyset$  for every machine  $i \in P$ .
4 for  $i = 1$  to  $k$  do
5   while  $\sum_{g \in T_i} s_g \leq 2c_i$  and  $A \neq \emptyset$  do
6     Pick the job  $g$  with the largest size in  $A$ .
7      $T_i \leftarrow T_i \cup \{g\}$ ,  $A \leftarrow A \setminus \{g\}$ .
8   end
9 end
10  $G = A$ .
```

Proof of Theorem 4 Clearly, Algorithm 2 runs in polynomial time. From Algorithm 3 and Lemma 6, we can know that each agent can process all the jobs that Algorithm 2 allocates to her while ensuring the maximum completion time of her machines doesn't exceed 2MMS_i . This implies that $v_i(A_i) \leq 2\text{MMS}_i$ holds for every agent $i \in N$, thus completing the proof. ■

Remark Note that although the proof of Lemma 6 requires the value of each MMS_i which is NP-hard to compute, Algorithm 2 runs in polynomial time, which means there exists a way for each agent i to complete the jobs allocated to her in $2 \cdot \text{MMS}_i$ time. To explicitly construct such a way, in Appendix B, we show how to modify Algorithm 3 so that it returns an allocation that satisfies the requirement in polynomial time (without computing the MMS_i values).

6 Conclusion

In this work, we study the fair allocation problem of allocating indivisible chores when the valuations are subadditive and fairness is measured by MMS. We first show that no algorithm can ensure better than n -MMS even when the valuations are submodular. Then we considered two specific combinatorial valuations, namely, bin packing and job scheduling. For bin packing valuations, we show that the best possible approximation algorithm is round robin which achieves 2-MMS. For job scheduling valuations, we show a 2-MMS allocation can be found efficiently, and inherit the lower bound of $44/43$ from additive valuations.

Our work uncovers several future directions. One immediate direction is to design better approximation algorithms or lower bound instances for the job scheduling valuations. In this work, we restricted us on the case of related machines, it is intriguing to consider the general model of unrelated machines. As we mentioned, the notion of collective maximin share fairness in the job scheduling model can be viewed as a group-wise fairness notion (for both goods and chores), which could be studied of independent interest. Finally, we can investigate other combinatorial valuations that can better characterize real-world problems.

References

- G. Amanatidis, E. Markakis, A. Nikzad, and A. Saberi. Approximation algorithms for computing maximin share allocations. *ACM Trans. Algorithms*, 13(4):52:1–52:28, 2017.
- G. Amanatidis, E. Markakis, and A. Ntokos. Multiple birds with one stone: Beating $1/2$ for EFX and GMMS via envy cycle elimination. *Theor. Comput. Sci.*, 841:94–109, 2020.
- G. Amanatidis, G. Birmpas, A. Filos-Ratsikas, and A. A. Voudouris. Fair division of indivisible goods: A survey. *CoRR*, abs/2202.07551, 2022.
- H. Aziz, G. Rauchecker, G. Schryen, and T. Walsh. Algorithms for max-min share fair allocation of indivisible chores. In *AAAI*, pages 335–341. AAAI Press, 2017.
- H. Aziz, H. Moulin, and F. Sandomirskiy. A polynomial-time algorithm for computing a pareto optimal and almost proportional allocation. *Oper. Res. Lett.*, 48(5):573–578, 2020.
- H. Aziz, B. Li, H. Moulin, and X. Wu. Algorithmic fair allocation of indivisible items: A survey and new questions. *CoRR*, abs/2202.08713, 2022.
- S. Barman and S. K. Krishnamurthy. On the proximity of markets with integral equilibria. In *AAAI*, pages 1748–1755, 2019.
- S. Barman and S. K. Krishnamurthy. Approximation algorithms for maximin fair division. *ACM Trans. Economics and Comput.*, 8(1):5:1–5:28, 2020.
- S. Barman, A. Biswas, S. K. K. Murthy, and Y. Narahari. Groupwise maximin fair allocation of indivisible goods. In *AAAI*, pages 917–924. AAAI Press, 2018.
- U. Bhaskar, A. R. Sricharan, and R. Vaish. On approximate envy-freeness for indivisible chores and mixed resources. In *APPROX-RANDOM*, volume 207 of *LIPIcs*, pages 1:1–1:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- S. Brânzei and F. Sandomirskiy. Algorithms for competitive division of chores. *CoRR*, abs/1907.01766, 2019.
- E. Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.
- I. Caragiannis, D. Kurokawa, H. Moulin, A. D. Procaccia, N. Shah, and J. Wang. The unreasonable fairness of maximum nash welfare. *ACM Trans. Economics and Comput.*, 7(3):12:1–12:32, 2019.
- H. Chan, J. Chen, B. Li, and X. Wu. Maximin-aware allocations of indivisible goods. In *IJCAI*, pages 137–143. ijcai.org, 2019.
- U. Feige, A. Sapor, and L. Tauber. A tight negative example for MMS fair allocations. In *WINE*, volume 13112 of *Lecture Notes in Computer Science*, pages 355–372. Springer, 2021.
- J. Garg and S. Taki. An improved approximation algorithm for maximin shares. *Artif. Intell.*, 300: 103547, 2021.

- M. Ghodsi, M. T. Hajiaghayi, M. Seddighin, S. Seddighin, and H. Yami. Fair allocation of indivisible goods: Improvement. *Math. Oper. Res.*, 46(3):1038–1053, 2021.
- M. Ghodsi, M. T. Hajiaghayi, M. Seddighin, S. Seddighin, and H. Yami. Fair allocation of indivisible goods: Beyond additive valuations. *Artif. Intell.*, 303:103633, 2022.
- X. Huang and P. Lu. An algorithmic framework for approximating maximin share allocation of chores. In *EC*, pages 630–631. ACM, 2021.
- D. Kurokawa, A. D. Procaccia, and J. Wang. Fair enough: Guaranteeing approximate maximin shares. *J. ACM*, 65(2):8:1–8:27, 2018.
- B. Li, Y. Li, and X. Wu. Almost proportional allocations for indivisible chores. *TheWebConf*, 2022.
- R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *EC*, pages 125–131. ACM, 2004.
- H. Moulin. Fair division in the age of internet. *Annu. Rev. Econ.*, 2018.
- B. Plaut and T. Roughgarden. Almost envy-freeness with general valuations. *SIAM J. Discret. Math.*, 34(2):1039–1068, 2020.
- M. Seddighin and S. Seddighin. Improved maximin guarantees for subadditive and fractionally sub-additive fair allocation problem. 2022.
- S. Zhou and X. Wu. Approximately EFX allocations for indivisible chores. *CoRR*, abs/2109.07313, 2021.

A Bin Packing Model: Allocating Items among Bins

In this section, we improve on Algorithm 1 and present an algorithm (Algorithm 4) that computes allocations where each agent i only needs $\lceil \frac{3}{2} \cdot \text{MMS}_i \rceil$ bins to pack all the items allocated to her.

The basic idea corresponds to the Algorithm 1, first we allocate the large items into the empty bins as the bottom, and then we allocate the small items to the bins to fill the bins, and remove the small item which used to be added to make the bin overflow into single bin. Finally, to decrease the number of bins, we combine every two bins which is smaller than $\frac{1}{2}c_i$ into one bin.

Algorithm 4: Bin Packing in Polynomial Time

```

1 Input: An ordered instance over unlimited number of bins with capacity  $c_i$  and indivisible
   item set  $A_i$  whose costs,  $s_i : 2^{A_i} \mapsto \mathbb{R}_+$ , are nonnegative, monotone.
2 Output: A bin allocation  $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$  such that  $s_i(T_p) \leq c_i$  for all  $p \in [k]$  and
    $k = 2\text{MMS}_i$ .
3  $L_i^{A_i} = L_i \cap A_i, H_i^{A_i} = H_i \cap A_i$ .
4 while  $H_i^{A_i} \neq \emptyset$  do
5   | Choose item  $g \in H_i^{A_i}$  arbitrarily and choose  $T_l = \emptyset$ .
6   |  $H_i^{A_i} \leftarrow H_i^{A_i} \setminus \{g\}, T_l = T_l \cup \{g\}$ .
7 end
8  $L = \{l | s_i(T_l) < s_i\}$ .
9 while  $L_i^{A_i} \neq \emptyset$  do
10  | Choose item  $g \in L_i^{A_i}$  arbitrarily and choose  $l \in L$  arbitrarily.
11  | if  $T_l \cup g > c_i$  then
12  |   | Choose  $T_{l'} = \emptyset$  arbitrarily.
13  |   |  $L_i^{A_i} \leftarrow L_i^{A_i} \setminus \{g\}, L \leftarrow L \setminus \{l, l'\}, T_{l'} = T_{l'} \cup \{g\}$ .
14  | else
15  |   |  $L_i^{A_i} \leftarrow L_i^{A_i} \setminus \{g\}, T_l = T_l \cup \{g\}$ .
16  | end
17 end
18 while There are  $T_{l_1} \neq \emptyset$  and  $T_{l_2} \neq \emptyset$  that  $s_i(T_{l_1}) \leq \frac{c_i}{2}, s_i(T_{l_2}) \leq \frac{c_i}{2}$  do
19  |  $T_{l_1} = T_{l_1} \cup T_{l_2}, T_{l_2} = \emptyset$ .
20 end

```

Lemma 7 Every agent i can use at most $\lceil \frac{3}{2} \cdot \text{MMS}_i \rceil$ bins to pack all her items.

Proof Firstly, we allow every bin can take one small item more than c_i , we call the bin filled. As Lemma 4 shows, there are less than MMS_i bins to take the big items. And when MMS_i bins are filled, the size is more than $\text{MMS}_i \cdot c_i$, it can take a set of items which size is less than $\text{MMS}_i \cdot c_i$ and a extra small item. As Lemma 5 shows, it is enough to take all the small items. So there will be at most MMS_i filled bins to hold the allocation.

We remove the small item in each filled bin, put every two removed small items into a new bin, there be $\lceil \frac{\text{MMS}_i}{2} \rceil$ bins to take the extra items. So there are $\lceil \frac{3}{2} \cdot \text{MMS}_i \rceil$ bins in total. ■

Remark 1 Here we need to pointhich satisfies out there are other algorithms which can achieve 2-MMS and be simpler than Algorithm 1. But our algorithm wins at the situation when the agent's

$\text{MMS} \geq 2$, the approximation ratio is less than 2. In fact, the allocation computed by Algorithm 1 only needs $\lceil \frac{3}{2}\text{MMS} \rceil$ bins to take. As we discussed above, the approximation ratio is $\frac{3}{2}\text{MMS}$ for agent which MMS is even and the approximation ratio is $\frac{3\text{MMS}+1}{2\text{MMS}}$ for agent which MMS is odd.

B Job Scheduling: Allocating Jobs among Machines

In this section, we show how every agent can allocate the jobs allocated to her among her machines, without knowing the actual values of the machines' capacities. The algorithm (Algorithm 5) achieves this by employing Algorithm 3 as a subroutine. It starts by setting the thresholds c_i s of machines, making sure c_i less than each agent's capacity $c_{i,j}$. Then the thresholds increase as not all the items can be allocated. In such case, it is needed that $c_i \leq c_{i,j}$. As mentioned above, Algorithm 5 can efficiently find the thresholds which equals to $c_{i,j}$.

Algorithm 5: Job Schedule in Polynomial Time

- 1 **Input:** An ordered instance over $|Q_i|$ indivisible jobs and k_i machines $\{d_1, d_2, \dots, d_{k_i}\}$ with speed $\rho_{i,1}, \rho_{i,2}, \dots, \rho_{i,k_i}$ in descending order, whose costs, $s_i : 2^{[m]} \mapsto \mathbb{R}_+$, are nonnegative, monotone.
 - 2 **Output:** An allocation $\mathcal{T} = \{T_1, T_2, \dots, T_{k_i}\}$ such that $s_i(T_p)/\rho_{i,p} \leq 2\text{MMS}_i$ for all $p \in [k_i]$.
 - 3 Initialize set of agents $A = [k_i]$, set of goods $G = Q_i$, and $T_i = \emptyset$ for all $i \in A$. Set $\delta \in (0, 1)$ to be an arbitrarily small constant.
 - 4 $c_1 = \max_{j \in [m]} s_{i,j}$.
 - 5 $c_p = c_1 \cdot \frac{\rho_{i,p}}{\rho_{i,1}}$ for any $p \in [k_i]$.
 - 6 **while** $G \neq \emptyset$ **do**
 - 7 For all $p \in [k_i]$, update $c_p \leftarrow (1 + \delta) \cdot c_p$.
 - 8 $(T_1, T_2, \dots, T_{k_i}, G) \leftarrow \text{Bag-Filling}(c_1, c_2, \dots, c_{k_i})$.
 - 9 **end**
-

C Missing Proofs

Algorithm 6: α -MMS Allocation for General Bin-Packing Instances

- 1 **Input:** Instance $I = (N, M, \{v_i\}_{i \in N}, \{s_i\}_{i \in N})$ with an allocation $\mathcal{A}' = (A'_1, \dots, A'_n)$ for the IDO instance $I' = (N, M, \{v'_i\}_{i \in N}, \{s'_i\}_{i \in M})$ such that $v'_i(A'_i) \geq \alpha \cdot \text{MMS}_i$ for all $i \in N$.
 - 2 **Output:** An allocation $\mathcal{A} = (A_1, A_2, \dots, A_n)$ such that $v_i(A_i) \geq \alpha \cdot \text{MMS}_i$ for all $i \in N$.
 - 3 For all $i \in N$ and $g \in A'_i$ set $p_g := i$.
 - 4 Initialize allocation $\mathcal{A} = (A_1, A_2, \dots, A_n)$ with $A_i = \emptyset$ for all $i \in N$, and initialize $R \leftarrow M$.
 - 5 **for** $g = m$ **to** 1 **do**
 - 6 Pick $k \in \arg \min_{g' \in R} \{s_{p_g, g'}\}$.
 - 7 Update $A_{p_g} \leftarrow A_{p_g} \cup \{k\}$ and $R \leftarrow R \setminus \{k\}$.
 - 8 **end**
-

Proof of Lemma 3 We show that given I and $\mathcal{A}'$, Algorithm 6 finds $\mathcal{A}$ in polynomial time. Clearly, Algorithm 6 runs in polynomial time. Now, we will show that Algorithm 6 finds the required allocation $\mathcal{A}$. Considering the g -th iteration of the second for-loop (Steps 5 to 8), suppose that item k_g is allocated to agent i , we have $k \in A'_i$ and $k_g \in A_i$. Before k_g is allocated, exactly $g - 1$ items have been allocated, since an item is removed from the set R after it is allocated. Therefore, k_g is among the top g smallest items for agent i . Note that g is the item with the exactly g -th smallest size for agent i , hence we have $s'_i(g) \geq s_i(k_g)$ for any $g \in A'_i$ and $k_g \in A_i$. This implies that it requires no more bins for agent i to pack items in A_i than those in A'_i , i.e., $v_i(A_i) \leq v'_i(A'_i)$. Note that the maximin share depends on the sizes of the items but not on the order, which means that the maximin share of agent i in I' is the same as her maximin share in I . Hence, the condition that $v'_i(A'_i) \leq \alpha \cdot \text{MMS}_i$ gives us $v_i(A_i) \leq \alpha \cdot \text{MMS}_i$ for all $i \in N$. ■