How to Fairly Allocate Easy and Difficult Chores

Indian Institute of Technology Kanpur
Algorithmic Game Theory

Group
Manikanta (220409), Havish (220879), Jyothisha (220862)

Instructor
Prof. Sunil Easaw Simon

Course
CS656 — Algorithmic Game Theory

Submitted on
24rd April 2025

Contents

1 Introduction

Fairly allocating indivisible chores among agents with differing preferences is a challenging problem. This report examines the work presented in "How to Fairly Allocate Easy and Difficult Chores," which focuses on the specific setting of bivalued utilities. In this model, chores are classified as either easy (cost 1) or difficult (cost $p > 1$). The paper provides key results on finding allocations that are both efficient (Pareto optimal) and fair (envy-free up to one item and maximin share) within this structured environment.

2 Preliminaries

An instance of the chore allocation problem considered in this paper is formally defined by a set of agents $\mathcal{N} = \{1, \ldots, n\}$, a set of indivisible items $\mathcal{M} = \{c_1, \ldots, c_m\}$ which are referred to as chores, and a utility profile $\mathbf{v} = (v_1, \ldots, v_n)$. Each agent $i \in \mathcal{N}$ has an additive utility function $v_i : \mathcal{M} \to \mathbb{R}_{\leq 0}$, where $v_i(S) = \sum_{c \in S} v_i(c)$ for any bundle of chores $S \subseteq \mathcal{M}$. Since these are chores, utilities are non-positive.

Definition 2.1 (Bivalued Utilities)**.** An agent i has *bivalued utilities* if there exist two distinct negative values, which by scaling we assume to be -1 and $-p$ for some $p > 1$, such that for every chore $c \in \mathcal{M}$, $v_i(c) \in \{-1, -p\}$. The paper primarily focuses on instances where all agents share the same p.

Definition 2.2 (Allocation)**.** An *allocation* $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is a partition of the set of chores $\mathcal{M}$ among the agents, such that $\bigcup_{i \in \mathcal{N}} \mathbf{x}_i = \mathcal{M}$ and $\mathbf{x}_i \cap \mathbf{x}_j = \emptyset$ for all $i \neq j$. Each $\mathbf{x}_i$ is the bundle of chores assigned to agent i.

The paper investigates several standard fairness concepts:

Definition 2.3 (Envy-Freeness (EF))**.** An allocation $\mathbf{x}$ is *envy-free* if no agent strictly prefers another agent's bundle to their own: $v_i(\mathbf{x}_i) \geq v_i(\mathbf{x}_j)$ for all $i, j \in \mathcal{N}$.

Definition 2.4 (Envy-Freeness up to One Item (EF1))**.** An allocation $\mathbf{x}$ is *envy-free up to one item* (EF1) if for every pair of agents $i, j \in \mathcal{N}$, if agent i's bundle is not empty ($\mathbf{x}_i \neq \emptyset$), there exists a chore $c \in \mathbf{x}_i$ such that agent i's utility for their bundle with chore c removed is at least their utility for agent j's bundle: $v_i(\mathbf{x}_i \setminus \{c\}) \geq v_i(\mathbf{x}_j)$.

Definition 2.5 (Maximin Share (MMS))**.** For an agent i, their *maximin share* is the maximum utility $v_i(S)$ they can guarantee themselves from any partition of $\mathcal{M}$ into n bundles, assuming the other $n - 1$ agents will collude to leave agent i with the bundle that minimizes their utility. An allocation $\mathbf{x}$ is an *MMS allocation* if for every agent $i \in \mathcal{N}$, $v_i(\mathbf{x}_i) \geq MMS_i$, where MMS_i is agent i's maximin share value.

Definition 2.6 (Pareto Optimality (PO))**.** An allocation $\mathbf{x}$ is *Pareto optimal* (PO) if there is no other allocation $\mathbf{x}'$ such that $v_i(\mathbf{x}'_i) \geq v_i(\mathbf{x}_i)$ for all $i \in \mathcal{N}$ and $v_j(\mathbf{x}'_j) > v_j(\mathbf{x}_j)$ for at least one agent $j \in \mathcal{N}$.

Definition 2.7 (Same Type)**.** Two agents $i, j \in \mathcal{N}$ are considered of the *same type* if their utility functions are identical for all chores: $v_i(c) = v_j(c)$ for all $c \in \mathcal{M}$. The number of distinct agent types in an instance is denoted by p.

For computational analysis, it is assumed that utility values are rational numbers and that the number of chores m is at least as large as the number of agents n.

3 EF1 + PO for Bivalued Chores

A major contribution of the paper is proving the existence and providing a polynomial-time algorithm for finding an EF1 and PO allocation for chore division with bivalued utilities. The approach is based on the concept of Fisher markets.

3.1 Fisher Markets for Chore Division

In a Fisher market for chores, a price $\mathbf{p}(c) > 0$ is assigned to each chore c. Agent i's pain per buck ratio (PB) for a chore c is $\mathrm{PB}_i(c) = \frac{|v_i(c)|}{\mathbf{p}(c)}$. The minimum pain per buck ratio (MPB) for agent i is $\mathrm{MPB}_i = \min_{c \in \mathcal{M}} \mathrm{PB}_i(c)$. A chore c is referred as an MPB chore for i is $\mathrm{PB}_i(c) = \mathrm{MPB}_i$.

An allocation $\mathbf{x}$ and a price vector $\mathbf{p}$ form a Fisher equilibrium if each agent is allocated only their MPB chores.

Proposition 3.1. *Every equilibrium allocation is Pareto optimal.*

This proposition holds from first welfare theorem.

The algorithm aims to find a fair equilibrium. So, now we define price envy-freeness up to one item. This is similar envy-freeness but here envy is due to difference is prices of bundles.

Definition 3.2 (Price envy-freeness up to one item (pEF1))**.** We say that $(\mathbf{x}, \mathbf{p})$ is price envy-free up to one item (pEF1) if, $\forall\ i, j \in \mathcal{N}$ with $\mathbf{x}_i \neq \emptyset$, there is a chore $c \in \mathbf{x}_i$ such that $\mathbf{p}(\mathbf{x}_i \setminus \{c\}) \leq \mathbf{p}(\mathbf{x}_j)$.

Lemma 3.3. *If $(\mathbf{x}, \mathbf{p})$ is a pEF1 equilibrium, then the allocation $\mathbf{x}$ is EF1.*

Proof. Let i, j be any two agents. By pEF1, there exists some item $c \in x_i$ such that

$$p\big(x_i \setminus \{c\}\big) \ \leq\ p(x_j).$$

Multiplying both sides by MPB_i and using the identity

$$|v_i(S)| \ =\ \mathrm{MPB}_i \ \cdot\ p(S) \ =\ \sum_{c' \in S} \mathrm{PB}_i(c')\, p(c') \quad \text{for any bundle } S,$$

which holds because in an equilibrium, agent i is allocated only her MPB chores, we get

$$|v_i\big(x_i \setminus \{c\}\big)| = \mathrm{MPB}_i\, p\big(x_i \setminus \{c\}\big) \ \leq\ \mathrm{MPB}_i\, p(x_j) \ \leq\ \sum_{c' \in x_j} \mathrm{PB}_i(c') p(c') = |v_i(x_j)|.$$

The second inequality holds because $\mathrm{PB}_i(c') \geq \mathrm{MPB}_i$ for all chores c'. Since utilities for chores are non-positive, $|v_i(S)| = -v_i(S)$. Thus, $-v_i(\mathbf{x}_i \setminus \{c\}) \leq -v_i(\mathbf{x}_j)$, which implies $v_i(\mathbf{x}_i \setminus \{c\}) \geq v_i(\mathbf{x}_j)$. This shows that $\mathbf{x}$ satisfies EF1. □

For a bundle $S \subseteq \mathcal{M}$, we define its *price-one* as $\mathbf{p}_{\text{up to 1}}(S) = \mathbf{p}(S) - \max_{c \in S} \mathbf{p}(c)$ if S is not empty, and 0 if S is empty. The price-one represents the total price of the bundle excluding the most expensive item. We often refer to an agent $\mathrm{ls} \in \mathcal{N}$ as the *least spender* if their bundle has the minimum total price among all agents, i.e., $\mathrm{ls} \in \arg\min_{i \in \mathcal{N}} \mathbf{p}(\mathbf{x}_i)$. With these definitions, we can see that a state $(\mathbf{x}, \mathbf{p})$ is price envy-free up to one item (pEF1) if and only if the price-one of every agent's bundle is less than or equal to the total price of the least spender's bundle, i.e., $\mathbf{p}_{\text{up to 1}}(\mathbf{x}_i) \leq \mathbf{p}(\mathbf{x}_{\mathrm{ls}})$ for all $i \in \mathcal{N}$. We call an agent $i \in \mathcal{N}$ a *violator* if their price-one is strictly greater than the least spender's total price: $\mathbf{p}_{\text{up to 1}}(\mathbf{x}_i) > \mathbf{p}(\mathbf{x}_{\mathrm{ls}})$. Therefore, $(\mathbf{x}, \mathbf{p})$ is pEF1 if and only if no agent is a violator.

Definition 3.4 (MPB alternating path)**.** An *MPB alternating path* of length ℓ from i_ℓ to i_0 is a sequence $i_0 \xleftarrow{c_1} i_1 \xleftarrow{c_2} \cdots \xleftarrow{c_\ell} i_\ell$, where $c_k \in \mathbf{x}_{i_k}$ (agent i_k owns chore c_k) and c_k is an MPB chore for i_{k-1} (agent i_{k-1} considers c_k among her minimum pain-per-buck chores) for $k = 1, \ldots, \ell$. Such a path indicates a sequence of potential chore transfers that maintain the equilibrium property. We write $i_0 \rightsquigarrow i_\ell$ if such a path exists.

3.2 Algorithm

Algorithm 1: EF1 + PO for Bivalued Chores

Phase 1 *Initialization*
- Let $\mathbf{x}$ be an allocation maximizing social welfare $\sum_{i \in \mathcal{N}} v_i(\mathbf{x}_i)$.
- For each $c \in \mathcal{M}$, let $\mathbf{p}_c = p \cdot |\max_{i \in \mathcal{N}} v_i(c)|$
- $k \leftarrow 1$, the number of the current iteration

Phase 2a *Reallocate chores*
- **for** $\ell \in (k-2, k-3, \ldots, 2, 1)$ **do**
  - **while** *true* **do**
    - $i \leftarrow$ an agent from $\arg\max_{i \in H_\ell} \mathbf{p}_{\text{up to } 1}(\mathbf{x}_i)$
    - $j \leftarrow$ an agent from $\arg\min_{j \in H_{\ell+1} \cup \cdots \cup H_{k-1}} \mathbf{p}(\mathbf{x}_j)$
    - **if** $\mathbf{p}_{\text{up to } 1}(\mathbf{x}_i) > \mathbf{p}(\mathbf{x}_j)$ **then**
      - $c \leftarrow$ any item from $\mathbf{x}_i \setminus \text{entitled}(i)$
      - Transfer c from i to j
    - **else**
      - **break**

Phase 2b *Reallocate chores*
- **while** *true* **do**
  - $\text{ls} \leftarrow$ an agent from $\arg\min_{i \in \mathcal{N}} \mathbf{p}(\mathbf{x}_i)$
  - **if** *there is an MPB alternating path* $\text{ls} \xleftarrow{c_1} i_1 \xleftarrow{c_2} \cdots \xleftarrow{c_\ell} i_\ell$ *with* $\mathbf{p}_{\text{up to } 1}(\mathbf{x}_{i_\ell}) > \mathbf{p}(\mathbf{x}_{\text{ls}})$ **then**
    - Choose such a path of minimum length ℓ
    - Transfer c_ℓ from i_ℓ to $i_{\ell-1}$
  - **else**
    - **break**
- **if** $\mathbf{x}$ *satisfies* pEF1 **then**
  - **return** $\mathbf{x}$

Phase 3 *Price reduction*
- $H_k \leftarrow \{i \in \mathcal{N} : \text{there is an agent ls} \in \arg\min_{i \in \mathcal{N}} \mathbf{p}(\mathbf{x}_i) \text{ with ls} \rightsquigarrow i\}$
- ▶ Timestamp: $t_{k,b}$
- $\alpha \leftarrow \min\{\text{PB}_i(c)/\,\text{MPB}_i : i \in H_k, c \in \bigcup_{j \in \mathcal{N} \setminus H_k} \mathbf{x}_j\}$
- **for** $i \in H_k$ **do**
  - $\text{entitled}(i) \leftarrow \mathbf{x}_i$
  - **for** $c \in \mathbf{x}_i$ **do**
    - $\mathbf{p}_c \leftarrow \frac{1}{\alpha} \cdot \mathbf{p}_c$
- ▶ Timestamp: $t_{k,a}$
- $k \leftarrow k + 1$
- Start Phase 2a (i.e. go to line 1)

Theorem 3.5. *Given a chore division problem $I = (\mathcal{N}, \mathcal{M}, \mathbf{v})$ with bivalued utilities, Algorithm 1 finds a PO and EF1 allocation in* $\text{poly}(n, m)$ *time.*

The algorithm maintains an equilibrium $(\mathbf{x}, \mathbf{p})$ throughout its execution. It proceeds in iterations (k). Each iteration consists of Phase 2a, 2b, and 3. Phase 2a and 2b reallocate chores while keeping prices fixed. Phase 3 updates prices for chores held by a specific set of agents H_k. Entitled chores $\text{entitled}(i)$ are those whose prices were reduced while allocated to agent i.

The proof of Theorem 3.5 relies on the following inductive hypotheses holding at time $t_{k,a}$ (after Phase 3 in iteration k) for all $k \geq 1$:

(H1) $H_k \cap H_\ell = \emptyset$ for all $1 \leq \ell < k$.

(H2) During iteration k, each time the algorithm reaches Line 1, there exists a chore $c \in \mathbf{x}_i \setminus \text{entitled}(i)$. All such chores are MPB chores for agent j.

(H3) At time $t_{k,b}$ (before Phase 3), each $i \in H_1 \cup \cdots \cup H_k$ is not a violator, so $\mathbf{p}_{\text{up to 1}}(\mathbf{x}_i) \leq \mathbf{p}(\mathbf{x}_{\text{ls}})$ where ls is the least spender.

(H4) At time $t_{k,a}$, each $i \in H_1 \cup \cdots \cup H_k$ owns every entitled item, $\text{entitled}(i) \subseteq \mathbf{x}_i$.

(H5) When Line 1 is reached during iteration k, α is set to p.

(H6) At time $t_{k,a}$, we have $\mathbf{p}(c) \in \{1, p\}$ for all $c \in \mathcal{M}$. If $\mathbf{p}(c) = 1$, then $c \in \text{entitled}(i)$ for some $i \in H_1 \cup \cdots \cup H_k$.

(H7) At time $t_{k,a}$, we have $\text{MPB}_i = 1$ for all $i \in H_1 \cup \cdots \cup H_k$, and $\text{MPB}_i = 1/p$ for all other agents.

3.2.1 Correctness and Termination

Assuming these hypotheses hold, the algorithm maintains an equilibrium and terminates in polynomial time.

Lemma 3.6. *Throughout the algorithm's execution,* $(\mathbf{x}, \mathbf{p})$ *is an equilibrium.*

Proof. We need to demonstrate that the pair $(\mathbf{x}, \mathbf{p})$ consistently represents a Fisher equilibrium at every step of the algorithm. A Fisher equilibrium requires that each agent is allocated only chores that provide them with their minimum pain-per-buck (MPB).

Let's examine each phase:

Phase 1 (Initialization): The algorithm begins by finding an allocation that maximizes the total social welfare (sum of utilities). For the specific case of bivalued chores, it is a known result that a social welfare maximizing allocation corresponds to a Fisher equilibrium with prices set in a particular way. The prices initialized in Line 1 are precisely those that establish an equilibrium for the initial social welfare maximizing allocation.

Phase 2a (Reallocate chores): In this phase, chores are transferred from an agent i to an agent j (Line 1). A key condition for this transfer is that the chore c being moved must be one that agent i owns but is not entitled to $(\mathbf{x}_i \setminus \text{entitled}(i))$. Furthermore, Hypothesis (H2) (which we prove by induction) guarantees that any such chore c is an MPB chore for the receiving agent j. Since chore c was part of agent i's bundle $\mathbf{x}_i$, and $(\mathbf{x}, \mathbf{p})$ was an equilibrium before the transfer, c must have been an MPB chore for agent i. Therefore, the transfer involves a chore that is an MPB chore for both the agent giving it away and the agent receiving it. Transferring an item that is equally "efficient" in terms of pain per buck for both parties involved in the transfer maintains the equilibrium property for all agents.

Phase 2b (Reallocate chores): Similar to Phase 2a, transfers in Phase 2b (Line 1) occur along an MPB alternating path. By definition of such a path, the chore c_ℓ being transferred from i_ℓ to $i_{\ell-1}$ is an MPB chore for $i_{\ell-1}$. Since c_ℓ was in i_ℓ's bundle $\mathbf{x}_{i_\ell}$, it was also an MPB chore for i_ℓ (because $(\mathbf{x}, \mathbf{p})$ was an equilibrium before this transfer). Again, transferring an item that is an MPB chore for both the giver and receiver ensures that no agent is allocated a non-MPB chore after the transfer, thus preserving the equilibrium.

Phase 3 (Price reduction): In this phase, the prices of chores owned by agents in the set H_k are scaled down by a factor of α (Line 1). The value of α is carefully chosen (Line 1) based on the minimum ratio of $\mathrm{PB}_i(c)/\mathrm{MPB}_i$ for agents in H_k and chores outside H_k. When the price $\mathbf{p}(c)$ of a chore $c \in \mathbf{x}_i$ for $i \in H_k$ is updated to $\mathbf{p}'(c) = \mathbf{p}(c)/\alpha$, the pain-per-buck ratio for agent i for this chore becomes $\mathrm{PB}'_i(c) = |v_i(c)|/(\mathbf{p}(c)/\alpha) = \alpha \cdot (|v_i(c)|/\mathbf{p}(c)) = \alpha \cdot \mathrm{PB}_i(c)$. Since c was an MPB chore for i before the update ($\mathrm{PB}_i(c) = \mathrm{MPB}_i$), the new pain-per-buck is $\alpha \cdot \mathrm{MPB}_i$. The definition of α and the inductive hypotheses (specifically (H5) and (H7)) guarantee that this new value $\alpha \cdot \mathrm{MPB}_i$ becomes the new MPB for agents in H_k. For chores not owned by agents in H_k, their prices remain unchanged, and for agents outside H_k, their bundles and MPB values relative to those prices also remain consistent. Thus, the price update is performed in a way that maintains the property that all allocated chores are MPB chores for their owners.

Since the equilibrium property is established in Phase 1 and maintained through the transfers in Phase 2a and 2b, and the price updates in Phase 3, the allocation and price vector remain an equilibrium throughout the algorithm's execution. □

The algorithm's termination in polynomial time is crucial. Phase 3 can execute at most n times because in each iteration k, the set H_k contains at least one agent (the least spender if the algorithm hasn't terminated), and these sets H_k are disjoint across iterations by (H1). Within each iteration, Phase 2a and 2b must terminate polynomially. Phase 2a involves transfers from agents in H_ℓ to agents in H_t with $t > \ell$. Since items only move from lower-indexed H sets to higher-indexed ones, and there are a finite number of items, Phase 2a terminates polynomially. Phase 2b's termination in polynomial time relies on a potential function argument (similar to algorithm for goods), showing that each transfer strictly decreases a potential function related to agent levels and the set of chores they can transfer. Thus, the algorithm terminates in polynomial time. Upon termination, the allocation is pEF1 (by the condition in Line 1) and thus EF1 (by Lemma 3.3), and it is an equilibrium (by Lemma 3.6), thus PO.

3.2.2 Proof of Hypotheses

The hypotheses are proved by induction on the iteration number k.

Lemma 3.7 (Base Case)**.** *Hypotheses (H1) to (H7) hold for $k = 1$.*

Proof. We need to show that all seven hypotheses are true at the end of the first iteration of the algorithm, specifically at time $t_{1,a}$ (after Phase 3 when $k = 1$).

(H1) ($H_1 \cap H_\ell = \emptyset$ for $\ell < 1$): This hypothesis states that the set H_1 is disjoint from the sets H_ℓ from previous iterations. Since $k = 1$ is the very first iteration, there are no previous iterations (no $\ell < 1$). Thus, this condition is trivially satisfied.

(H2) (Existence of non-entitled item in Phase 2a): This hypothesis applies to transfers within Phase 2a. In the first iteration ($k = 1$), the loop for Phase 2a ('For $\ell \in (k-2, \dots)$') becomes 'For $\ell \in (-1, -2, \dots)$'. This loop is empty, meaning Phase 2a is entirely skipped in the first iteration. Since Phase 2a never executes, the conditions related to transfers within it hold vacuously.

(H3) (Agents in H_1 are not violators at $t_{1,b}$): The algorithm reaches Phase 3 of iteration 1 because Phase 2b terminated. Phase 2b is designed to continue reallocating chores as long as there is a violator reachable from a least spender. When Phase 2b terminates, it guarantees that no agent is a violator. Therefore, at time $t_{1,b}$ (when Phase 2b ends and Phase 3 begins), no agent is a violator. Since H_1 is a subset of all agents, it is true that no agent in H_1 is a violator at $t_{1,b}$.

(H4) (Entitled items are owned by agents in H_1 at $t_{1,a}$): At time $t_{1,a}$, Phase 3 has just completed. In Phase 3, for each agent i in the set H_1, the set of entitled items entitled(i) is explicitly set to

be equal to their current bundle $\mathbf{x}_i$ (Line **??**). For agents not in H_1, entitled(i) remains empty. Therefore, for all agents $i \in H_1 \cup \cdots \cup H_1 = H_1$, their entitled items are exactly the chores in their bundle, meaning entitled(i) $\subseteq \mathbf{x}_i$.

(H5) ($\alpha = p$ in iteration 1): When the algorithm reaches Line 1 in Phase 3 of the first iteration, α is calculated as $\min\{\mathrm{PB}_i(c)/\mathrm{MPB}_i : i \in H_1, c \in \bigcup_{j \notin H_1} \mathbf{x}_j\}$. At the start of Phase 1, prices are initialized. For a chore c, $\mathbf{p}_c = p \cdot \max_l |v_l(c)|$. Since utilities are $\{-1, -p\}$, $\max_l |v_l(c)|$ is either 1 or p. So initial prices are either p or p^2. The minimum pain per buck for any agent i is $\mathrm{MPB}_i = \min_{c'} |v_i(c')|/\mathbf{p}(c')$. Since $v_i(c') \in \{-1, -p\}$ and prices are $\{p, p^2\}$, the possible PB values are $1/p, p/p, 1/p^2, p/p^2$. Assuming $p > 1$, the minimum is $1/p$. So, at $t_{1,b}$, $\mathrm{MPB}_i = 1/p$ for all agents. H_1 consists of agents reachable from a least spender. Analysis of the initial allocation and prices shows that agents in H_1 are those who own chores that are MPB for the least spender. Chores owned by agents not in H_1 cannot be MPB for agents in H_1 (otherwise the owner would be reachable). For an agent $i \in H_1$ (with $\mathrm{MPB}_i = 1/p$) and a chore c owned by $j \notin H_1$, c is not an MPB chore for i. This means $\mathrm{PB}_i(c) > \mathrm{MPB}_i = 1/p$. The possible PB values are $1/p$ and 1. Thus, $\mathrm{PB}_i(c)$ must be 1. Therefore, the minimum ratio $\mathrm{PB}_i(c)/\mathrm{MPB}_i$ is $1/(1/p) = p$. So α is set to p.

(H6) (Prices are $\{1, p\}$ at $t_{1,a}$, price 1 implies entitled): At $t_{1,b}$, prices were $\{p, p^2\}$. In Phase 3, prices of chores owned by agents in H_1 are multiplied by $1/\alpha = 1/p$. As shown in the proof of (H5), chores with price p^2 at $t_{1,b}$ are owned by agents in H_1. These prices become $p^2/p = p$. Chores with price p at $t_{1,b}$ owned by agents in H_1 become $p/p = 1$. Chores owned by agents not in H_1 (which had price p at $t_{1,b}$) remain at price p. Thus, at $t_{1,a}$, all chore prices are either 1 or p. If a chore's price is 1 at $t_{1,a}$, its price must have been p at $t_{1,b}$, and it must have been owned by an agent in H_1 whose price was reduced. By definition in Phase 3, such a chore is included in entitled(i) for its owner $i \in H_1$.

(H7) (MPBs are 1 for H_1, $1/p$ for others at $t_{1,a}$): For agents $i \in H_1$, their chores now have prices $\{1, p\}$. Their value for these chores is either -1 or $-p$. The possible PB values are $|-1|/1 = 1$, $|-p|/1 = p$, $|-1|/p = 1/p$, $|-p|/p = 1$. The minimum of these is 1. So $\mathrm{MPB}_i = 1$ for $i \in H_1$. For agents $j \notin H_1$, the prices of their chores remain p. Their values are -1 or $-p$. The possible PB values are $|-1|/p = 1/p$ and $|-p|/p = 1$. The minimum is $1/p$. So $\mathrm{MPB}_j = 1/p$ for $j \notin H_1$.

All hypotheses hold for $k = 1$. □

Lemma 3.8. *Let t_{mid} denote the time when the algorithm reaches Line 1 in iteration $k+1$, i.e. when Phase 2a ends and Phase 2b begins. At time t_{mid}, no agent in $H_1 \cup \cdots \cup H_k$ is a violator.*

This lemma essentially states that after Phase 2a finishes in any given iteration $k+1$, the agents who were part of the "active" sets H_1 through H_k in previous or the current iteration are no longer violating the pEF1 condition relative to the current least spender. Phase 2a's transfers are designed to resolve specific types of price-one envy among agents in different H_ℓ sets, and this lemma confirms that this phase successfully eliminates violations for the agents in the union of these sets up to H_k.

Lemma 3.9. *During the execution of Phase 2b in iteration $k+1$, no entitled items are transferred. Further, at the end of Phase 2b, no agent $i \in H_1 \cup \cdots \cup H_k$ is a violator.*

This lemma has two main points. First, it guarantees that chores whose prices were reduced in previous Phase 3 steps (entitled items) will not be moved away from their current owners during Phase 2b. This is important for maintaining the structure related to entitled items. Second, it reinforces the outcome of Phase 2a by stating that even after the transfers in Phase 2b, the agents in $H_1 \cup \cdots \cup H_k$ remain non-violators. This is crucial because Phase 3's definition of H_{k+1} and subsequent price adjustments depend on who is a violator at the end of Phase 2b.

Lemma 3.10 (Induction step)**.** *Suppose Hypotheses (H1) through (H7) hold for all iterations ℓ from 1 to k. If Algorithm 1 reaches time $t_{k+1,a}$ (after Phase 3 of iteration $k+1$), then Hypotheses (H1) to (H7) also hold for iteration $k+1$.*

Proof of Lemma 3.10. We proceed by assuming Hypotheses (H1) through (H7) hold for all iterations $1, \ldots, k$. We must show they hold for iteration $k+1$ at time $t_{k+1,a}$.

(H1) ($H_{k+1} \cap (H_1 \cup \cdots \cup H_k) = \emptyset$): The algorithm enters Phase 3 of iteration $k+1$ because Phase 2b has terminated, but the allocation is not yet pEF1. This termination condition implies that there exists at least one violator. By Lemma 3.9, we know that at $t_{k+1,b}$ (the moment Phase 2b ends and Phase 3 begins), no agent within the set $H_1 \cup \cdots \cup H_k$ is a violator. Therefore, any agent who is a violator at this time must necessarily be outside the set $H_1 \cup \cdots \cup H_k$. The set H_{k+1} is defined in Phase 3 as the set of agents reachable from a current least spender via an MPB alternating path. If an agent i were in both $H_1 \cup \cdots \cup H_k$ and H_{k+1}, it would mean an agent from a previous H set is reachable from a least spender at $t_{k+1,b}$. However, since agents in $H_1 \cup \cdots \cup H_k$ are not violators at $t_{k+1,b}$, their inclusion in H_{k+1} would contradict the termination condition of Phase 2b (which stops when reachable violators are resolved). Thus, H_{k+1} must be completely separate from the union of the previous H sets.

(H2) (Existence of non-entitled item in Phase 2a of iter $k+1$): This hypothesis states that during Phase 2a of iteration $k+1$, when the condition for transferring a chore from agent $i \in H_\ell$ to agent $j \in H_{\ell+1} \cup \cdots \cup H_k$ is met, there is always a non-entitled chore available in i's bundle, and any such chore is an MPB chore for j. This was explicitly proven in Lemma **??**, which relies on the inductive hypotheses holding for iteration k.

(H3) (Agents in $H_1 \cup \cdots \cup H_{k+1}$ are not violators at $t_{k+1,b}$): This hypothesis states that at the start of Phase 3 of iteration $k+1$, no agent in the union of all H sets found so far is a violator. Lemma 3.9 directly proves that agents in $H_1 \cup \cdots \cup H_k$ are not violators at $t_{k+1,b}$. By the definition of H_{k+1} and the fact that Phase 2b terminates when no reachable violator exists, it follows that all agents who are part of the structure that leads to Phase 3 (i.e., agents in H_{k+1}) are also non-violators at $t_{k+1,b}$. Therefore, the union $H_1 \cup \cdots \cup H_k \cup H_{k+1}$ contains no violators at $t_{k+1,b}$.

(H4) (Entitled items are owned by agents in $H_1 \cup \cdots \cup H_{k+1}$ at $t_{k+1,a}$): At time $t_{k+1,a}$, Phase 3 of iteration $k+1$ has just finished. For every agent $i \in H_{k+1}$, their entire current bundle $\mathbf{x}_i$ is designated as entitled(i) in Phase 3. For agents $i \in H_1 \cup \cdots \cup H_k$, their entitled items from previous iterations (which were entitled(i) at $t_{\ell,a}$ for $\ell \leq k$) remain in their possession because Lemma 3.9 guarantees that entitled items are not transferred out of $H_1 \cup \cdots \cup H_k$ during Phase 2b of iteration $k+1$. Phase 2a also does not transfer entitled items (by definition of the transfer condition). Thus, all items entitled in previous iterations remain with their owners in $H_1 \cup \cdots \cup H_k$, and the items newly entitled in iteration $k+1$ are owned by agents in H_{k+1}. Therefore, for any agent i in $H_1 \cup \cdots \cup H_{k+1}$, their entitled items are contained within their current bundle $\mathbf{x}_i$.

(H5) ($\alpha = p$ in iteration $k+1$): The value of α is computed in Phase 3 (Line 1) at time $t_{k+1,b}$. α is the minimum ratio $\mathrm{PB}_i(c)/\mathrm{MPB}_i$ for agents $i \in H_{k+1}$ and chores c owned by agents not in H_{k+1}. By Hypothesis (H7) for iteration k, agents outside $H_1 \cup \cdots \cup H_k$ have an MPB of $1/p$. Since H_{k+1} is disjoint from $H_1 \cup \cdots \cup H_k$ (by (H1) for $k+1$), all agents in H_{k+1} have an MPB of $1/p$ at $t_{k+1,b}$. By Hypothesis (H6) for iteration k, prices at $t_{k,a}$ were $\{1, p\}$, where price 1 items were entitled to agents in $H_1 \cup \cdots \cup H_k$. During Phase 2a and 2b, prices do not change. So at $t_{k+1,b}$, prices are still $\{1, p\}$. Chores owned by agents not in H_{k+1} include those owned by agents in $H_1 \cup \cdots \cup H_k$ (which have price 1) and those owned by agents outside $H_1 \cup \cdots \cup H_k \cup H_{k+1}$ (which have price p). Consider an agent $i \in H_{k+1}$ (MPB $1/p$) and a chore c owned by $j \notin H_{k+1}$. Since c is not owned by an agent in H_{k+1}, c cannot be an MPB chore for i (otherwise j would be reachable from i and thus

in H_{k+1}). So $\mathrm{PB}_i(c) > \mathrm{MPB}_i = 1/p$. Given prices are $\{1, p\}$ and values are $\{-1, -p\}$, the possible PB values for i are $1/1 = 1$, $p/1 = p$, $1/p$, $p/p = 1$. The only PB value strictly greater than $1/p$ is 1. Thus, $\mathrm{PB}_i(c) = 1$. The minimum ratio $\mathrm{PB}_i(c)/\mathrm{MPB}_i$ is $1/(1/p) = p$. Therefore, α is set to p.

(H6) (Prices are $\{1, p\}$ at $t_{k+1,a}$, price 1 implies entitled): At $t_{k+1,b}$, prices were $\{1, p\}$ (by (H6) for k). In Phase 3, prices of chores owned by agents in H_{k+1} are multiplied by $1/\alpha = 1/p$. As shown in the proof of (H5) for $k+1$, these chores had price p at $t_{k+1,b}$. Their new price is $p \cdot (1/p) = 1$. Chores owned by agents outside H_{k+1} retain their prices $\{1, p\}$. Thus, at $t_{k+1,a}$, all chore prices are either 1 or p. If a chore has price 1 at $t_{k+1,a}$, it must have either had price 1 at $t_{k+1,b}$ (meaning it was entitled to an agent in $H_1 \cup \cdots \cup H_k$) or its price was p at $t_{k+1,b}$ and it was owned by an agent in H_{k+1} whose price was reduced. In the latter case, by definition in Phase 3, the chore is included in entitled(i) for its owner $i \in H_{k+1}$.

(H7) (MPBs are 1 for $H_1 \cup \cdots \cup H_{k+1}$, $1/p$ for others at $t_{k+1,a}$): For agents $i \in H_{k+1}$, their MPB changes from $1/p$ to 1 because the prices of their chores change from p to 1 (as shown in (H6) for $k+1$). For agents $i \in H_1 \cup \cdots \cup H_k$, their bundles and the prices of their entitled chores (which constitute their entire bundles by (H4)) remain unchanged, so their MPB remains 1. For agents $j \notin H_1 \cup \cdots \cup H_{k+1}$, their bundles and prices remain unchanged, so their MPB remains $1/p$.

All hypotheses hold for iteration $k+1$. □

This concludes the inductive step of the proof. By showing that the hypotheses hold for the base case ($k = 1$) and that if they hold for iterations up to k, they also hold for $k+1$, we establish that these properties are maintained throughout the algorithm's execution. These properties are essential for proving the algorithm's correctness and polynomial time termination.

4 MMS Under Restricted Utilities

In general, it is not always possible to find MMS (maximin share) allocations for any kind of additive utilities. However, earlier work has shown that MMS allocations do exist for special cases, like when utilities are binary or strictly ordered. In this section, we extend these results to more general cases, such as weakly lexicographic and factored personalized bivalued utilities. The main theorem below summarizes the key result.

Theorem 4.1. *In every goods or chore division instance with weakly lexicographic or factored personalized bivalued utilities, an MMS allocation always exists and can be computed in polynomial time.*

We first review two important techniques for finding MMS allocations.

Ordered Instances

A useful trick is to assume that all agents rank the items in the same order. This means, for every agent, the most valuable item is the same, then the second most valuable, and so on. This assumption can be made without loss of generality, and it simplifies the problem. This idea works for both goods and chores.

Lemma 4.2. *Suppose we have a goods or chores division instance $(\mathcal{N}, \mathcal{M}, v)$. We can create a new instance where, for each agent, the utility function is just a reordered version of the original, so that $|v_i'(1)| \geq \ldots \geq |v_i'(m)|$. If $\mathbf{x}'$ is an MMS allocation for this new instance, then there is also an MMS allocation for the original instance, and it can be found in polynomial time.*

From now on, we will assume all instances are ordered in this way, unless stated otherwise. This means for each agent i, $|v_i(1)| \geq \ldots \geq |v_i(m)|$.

Valid Reductions

Another important technique is the idea of **valid reductions**. This means we can sometimes remove an agent and a bundle of items from the problem, and the MMS property will still hold for the remaining agents.

Definition 4.3 (Valid Reduction)**.** Let $(\mathcal{N}, \mathcal{M}, v)$ be a goods or chores division instance, i an agent, and S a subset of items. The pair (i, S) is a valid reduction if:

1. $v_i(S) \geq \text{MMS}_i^n(M)$, and
2. For every other agent $j \neq i$, $\text{MMS}_j^{n-1}(M \setminus S) \geq \text{MMS}_j^n(M)$.

If (i, S) is a valid reduction, we can give bundle S to agent i, and ignore both i and S from now on. In the reduced instance, each remaining agent still gets at least their MMS value.

Lemma 4.4. *For any goods or chores division instance $(\mathcal{N}, \mathcal{M}, v)$, if S is a bundle for agent i such that $v_i(S) \geq \text{MMS}_i^n(M)$, and for every other agent j, there is an MMS n-partition of M for agent j with a bundle S' containing S (for goods) or $S \subseteq S'$ (for chores), then (i, S) is a valid reduction.*

Proof. Fix any agent $j \neq i$. We need to show that $\text{MMS}_j^{n-1}(M \setminus S) \geq \text{MMS}_j^n(M)$. Take an MMS n-partition P_j for agent j as in the lemma. Then, create an $(n-1)$-partition of $M \setminus S$ by removing the bundle S' (which contains S for goods, or $S \subseteq S'$ for chores) from P_j. The value of the worst

bundle in this new partition is at least as large as in the original, so the MMS value does not decrease.

Goods division: Removing a bundle can only make the worst remaining bundle better for agent j. So, the MMS value for $M \setminus S$ with $n-1$ agents is at least as large as before.

Chores division: Removing chores from a bundle can only improve the value for the remaining bundles (since chores are bad). So, again, the MMS value for $M \setminus S$ with $n-1$ agents is at least as large as before.

Therefore, the reduction is valid. □

4.1 Exact MMS Value for Factored Utilities

The goal is to divide items into n bundles so that the smallest bundle (in terms of value) is as large as possible. The algorithm works by always giving the next largest item to the bundle that currently has the lowest total value. This keeps all bundles as balanced as possible throughout the process, which matches the goal of maximizing the minimum bundle value.

Algorithm 2: Compute a maximin n-partition for a factored utility function v

$x \leftarrow (x_i = \emptyset)_{i \in N}$ `// initialize each bundle as empty`
for *each item r in nonincreasing order of $|v(r)|$* **do**
  $k^* \leftarrow \arg\min_{k \in N} \big|v(x_k)\big|$ `// find the bundle with the lowest value`
  $x_{k^*} \leftarrow x_{k^*} \cup \{r\}$ `// add item r to bundle k*`
return x

Lemma 4.5. *Suppose v is a factored utility function over a set M (where all items are either goods or chores). Then Algorithm 2 produces a maximin n-partition of M under v in an efficient manner.*

Proof. Assume, for contradiction, that the allocation x generated by the algorithm is not a maximin partition. Among all possible maximin partitions, select x' that agrees with x on the largest possible prefix of the item order. Let ℓ be the first position where x and x' differ in assigning item r_ℓ to bundles. Let y denote the allocation of the first $\ell - 1$ items, which is identical in both x and x'. Let i be the bundle receiving r_ℓ in x, and i' be the bundle receiving r_ℓ in x', with $i \neq i'$.

Because the algorithm always places the next item in the bundle with the lowest current sum, it follows that $|v(y_i)| \leq |v(y_{i'})|$ at this step.

Now, examine two scenarios:

Case 1: $|v(y_i)| = |v(y_{i'})|$. In this situation, we can form a new partition x'' by swapping the sets of items (after y) between bundles i and i' according to their assignments in x'. This means x''_i gets what $x'_{i'}$ had after y, and $x''_{i'}$ gets what x'_i had after y. The values of these bundles are swapped, so x'' is still a maximin partition, and it matches x on the first ℓ items—contradicting our choice of x'.

Case 2: $|v(y_i)| < |v(y_{i'})|$. Since the algorithm assigns items in order of decreasing absolute value, every item before r_ℓ is at most as large as r_ℓ in value. Thus, $|v(y_i)|$ cannot be much less than $|v(y_{i'})|$; specifically, $|v(y_i)| \geq |v(y_{i'})| - |v(r_\ell)|$. We split into two subcases:

- **Case 2a:** If $|v(x'_i)| < |v(y_i)| + |v(r_\ell)| \leq |v(y_{i'})|$, then swapping r_ℓ and the remaining items in x'_i (after y) between i and i' will not decrease the smallest bundle value, nor increase the largest. This new partition will also match x up to item ℓ, again contradicting our choice of x'.

- **Case 2b:** If $|v(x'_i)| \geq |v(y_i)| + |v(r_\ell)|$, consider the items in $x'_i \setminus y_i$ as $\{s_1, s_2, \ldots\}$, ordered by their appearance. Let t be the smallest index such that adding s_1 through s_t to y_i reaches or exceeds $|v(y_i)| + |v(r_\ell)|$. Before s_t is added, the total is just below this threshold, and since each $|v(s_j)| \leq |v(r_\ell)|$, the gap is at most $|v(r_\ell)|$. When s_t is included, the sum exactly meets $|v(y_i)| + |v(r_\ell)|$, so the total value of $\{s_1, \ldots, s_t\}$ is precisely $|v(r_\ell)|$. Swapping this set with r_ℓ between i and i' yields a new maximin partition that agrees with x on the first ℓ items, again contradicting our initial choice.

In every possible case, we reach a contradiction. Therefore, the allocation x produced by the algorithm must indeed be a maximin partition. □

This algorithm is only guaranteed to work for factored utility functions. For more general utility functions, it may not find the true MMS value.

5 MMS Allocations for Restricted Utility Classes

In general, Maximin Share (MMS) allocations do not always exist for arbitrary additive utilities. However, we can guarantee the existence of MMS allocations for two important subclasses: weakly lexicographic utilities and bivalued utilities. In this section, we present efficient algorithms to compute these allocations using the techniques of valid reductions and ordered instances.

5.1 Weakly Lexicographic Utilities

Weakly lexicographic utilities capture scenarios where agents rank items in order of preference (possibly with ties), and each item is valued more than all less-preferred items combined. This represents a natural preference structure when agents prioritize getting their top choices.

Bad Cuts and Valid Reductions

First, we introduce the concept of a "bad cut," which is crucial for our algorithm.

Definition 5.1 (Bad Cuts)**.** In a goods or chore division instance $I = (\mathcal{N}, \mathcal{M}, v)$, we say that index $k \in [m-1]$ is a **cut** of agent i if $v_i(k) \neq v_i(k+1)$. Further, if k is not a multiple of n, we say that it is a **bad cut** of agent i. Define C_i to be the smallest bad cut of agent i; let $C_i = m$ if agent i does not have any bad cuts.

i	$\lvert v_1\rvert$	$\lvert v_2\rvert$	$\lvert v_3\rvert$	$\lvert v_4\rvert$	$\lvert v_5\rvert$	$\lvert v_6\rvert$	$\lvert v_7\rvert$	$\lvert v_8\rvert$	$\lvert v_9\rvert$	C_i
i_1	81	81	81	$81\rvert_4$	9	9	$9\rvert_7$	1	1	4
i_2	81	81	$81\rvert_3$	9	9	$9\rvert_6$	1	1	1	9
i_3	$729\rvert_1$	81	81	$81\rvert_4$	9	9	$9\rvert_7$	1	1	1

Table 1: An instance with weakly lexicographic utilities. Bad cuts at index k are shown as $\rvert_k$, and non-bad cuts as $\lvert_k$.

Example 1. The ordered instance described in the table consists of $n = 3$ agents and $m = 9$ items. The cuts of i_1, i_2, i_3 are $\{4, 7\}$, $\{3, 6\}$, $\{1, 4, 7\}$ respectively. Here, a cut is considered a bad cut if it is not divisible by $n = 3$. Then, all cuts of i_1 and i_3 are bad while i_2 has no bad cuts. Following the definition of C_i's, we have $C_{i_1} = 4$, $C_{i_2} = 9$, and $C_{i_3} = 1$.

The key insight for weakly lexicographic utilities is that we can identify specific bundles that are guaranteed to be part of some maximin partition. Because of the lexicographic structure, agents care much more about their high-ranked items than their low-ranked ones. By identifying "bad cuts" where preference values change significantly, we can create valid reductions that allow us to solve the problem recursively.

When an agent has a bad cut, it means there's a natural point to divide items that doesn't align with a perfect n-way split. We show that allocating items up to this cut (properly structured) forms a valid reduction - the agent gets enough value, and remaining agents don't lose their fair share.

First, we show that there is always a specific bundle structure in a maximin partition for weakly lexicographic utilities:

Lemma 5.2. *Suppose $I = (\mathcal{N}, \mathcal{M}, v)$ is a goods or chores division problem with weakly lexicographic utilities, and $i \in N$ is any agent. Then, there exists a maximin n-partition for agent i in which one of the bundles is $S = \{1, n+1, 2n+1, \ldots, kn+1\}$, where $k = \lfloor (C_i - 1)/n \rfloor$ and C_i is the smallest bad cut for agent i.*

Proof. Since C_i is the smallest bad cut for agent i, for each block of n items up to C_i, all items in the block have the same utility for i. When forming bundles, the algorithm distributes each block of n items so that every bundle gets one item from each block. After k such blocks, the next $C_i - kn$ items (if any) are assigned one per bundle to $C_i - kn$ bundles. These bundles thus get one extra item compared to the others.

Because of the lexicographic structure, these extra items are more valuable (for goods) or less burdensome (for chores) than all remaining items. The remaining items (from $C_i + 1$ to m) are then distributed among the other bundles. In this way, there is always a bundle consisting of the items at positions $1, n+1, \ldots, kn+1$, which is exactly S. □

Next, we show that picking the agent with the smallest (for goods) or largest (for chores) bad cut allows us to safely allocate their bundle and reduce the problem:

Lemma 5.3. *Let $I = (\mathcal{N}, \mathcal{M}, v)$ be a goods (resp., chores) division instance with weakly lexicographic utilities. If i is the agent with the smallest (resp., largest) bad cut C_i, and $S = \{1, n+1, \ldots, kn+1\}$ as above, then giving S to i is a valid reduction.*

Proof. From the previous lemma, we know S is a bundle in a maximin partition for agent i, so i receives at least their MMS value. For any other agent i', let $k' = \lfloor (C_{i'} - 1)/n \rfloor$ and $S' = \{1, n+1, \ldots, k'n+1\}$. By construction, $k \leq k'$ for goods (so $S \subseteq S'$), and $k \geq k'$ for chores (so $S \supseteq S'$). This means that removing S from the problem does not lower the MMS value for any other agent, so the reduction is valid. □

Algorithm for Computing MMS Allocation

Algorithm 3: MMS Allocation for Weakly Lexicographic Utilities

```
                    /* Input:  Instance (𝒩, ℳ, v) with weakly lexicographic utilities */
                                                          /* Output:  An MMS allocation */
if N = ∅ or M = ∅ then
    return empty allocation
for each agent i ∈ N do
    Find C_i, the smallest bad cut for i
Choose agent i* with the smallest C_{i*} (for goods) or largest C_{i*} (for chores)
k ← ⌊(C_{i*} − 1)/n⌋
S ← { 1, n + 1, 2n + 1, …, kn + 1}
Assign bundle S to agent i*
Recursively solve the problem for (N \ {i*}, M \ S, v)
return the combined allocation
```

This procedure repeatedly applies the above reduction, ensuring that each agent gets at least their MMS value, until all items are allocated.

5.2 Factored Personalized Bivalued Utilities

In this section, we present a valid reduction for factored personalized bivalued utilities. This is another special class of utility functions where each agent has only two possible values for items. For each agent i, items are either high-value (worth p_i) or low-value (worth 1).

Unlike the weakly lexicographic case, the simple approach of choosing the agent with the minimum or maximum bad cut doesn't work here. Why? Because agents might have very different values of p_i, making their "cuts" incomparable. Instead, we need a different measure that accounts for both the location of the cut and the high-value p_i of each agent.

The key insight is to track how the maximin algorithm would distribute items for each agent, and identify a specific pattern of "active" and "idle" bundles that emerges.

Idle Time and Active Bundles

Definition 5.4 (Idle Time)**.** In a goods or chore division instance $I = (\mathcal{N}, \mathcal{M}, v)$ with factored personalized bivalued utilities, we define AC_i as 0 if $C_i = m$ and as $n - (C_i \mod n)$ otherwise. Let the **idle time** of agent i be $T_i^{\text{idle}} = \min\{p_i \cdot AC_i, m - C_i\}$.

i	$\lvert v_1\rvert$	$\lvert v_2\rvert$	$\lvert v_3\rvert$	$\lvert v_4\rvert$	$\lvert v_5\rvert$	$\lvert v_6\rvert$	$\lvert v_7\rvert$	$\lvert v_8\rvert$	$\lvert v_9\rvert$	C_i	AC_i	T_i^{idle}
i_1	2	2	2	2 $\lvert_4$	**1**	**1**	1	1	1	4	2	4
i_2	5 $\lvert_1$	**1**	**1**	1	1	1	1	1	1	1	2	8
i_3	4	4	4	4	4	4	4	4 $\lvert_8$	**1**	8	1	1

Table 2: An instance with personalized bivalued utilities. Bold cells refer to active bundles AC_i, and the underlined ones refer to idle times T_i^{idle}.

Example 2. table:personalized-bival-example presents an instance with personalized bivalued utilities where $p_{i_1} = 2$, $p_{i_2} = 5$ and $p_{i_3} = 4$. The number of *active bundles* and *idle times* are also shown in the table.

Understanding the Algorithm

When allocating items using Algorithm 1 (the standard round-robin algorithm) for an agent with bivalued utilities, the process works in up to three phases:

1. **High-Value Phase:** It first divides the high-value items (value p_i) in round-robin fashion between all n bundles until reaching the bad cut. At this point, $C_i \mod n$ bundles have one extra high-value item.
2. **Low-Value Phase:** The remaining $AC_i = n - (C_i \mod n)$ bundles (called "active bundles") need more items to catch up. So the algorithm allocates low-value items (value 1) to these active bundles for exactly T_i^{idle} steps.
3. **Balanced Phase:** After that, if any items remain, they're divided equally among all bundles.

The term "idle time" represents how many low-value items are needed before all bundles become balanced again. It's either limited by how many low-value items exist ($m - C_i$) or by how many it would take to equalize all bundles ($p_i \cdot AC_i$).

Lemma 5.5. *For a goods or chore division instance $I = (\mathcal{N}, \mathcal{M}, v)$ with factored personalized bivalued utilities and agent $i \in N$, there exists a maximin n-partition of agent i in which one of the bundles is $S = \{1, n+1, \ldots, kn+1\}$, where $k = \lfloor (m - \max\{T_i^{idle} - AC_i, 0\} - 1)/n \rfloor$.*

Proof Sketch. We analyze what Algorithm 1 does when computing a maximin partition for agent i:

Case 1: If agent i has no bad cut, then $T_i^{\text{idle}} = AC_i = 0$, so $k = \lfloor (m-1)/n \rfloor$. The algorithm simply distributes all items in round-robin fashion, so the first bundle is exactly S.

Case 2: If agent i has a bad cut at $C_i = k'n + r$ (with remainder $r \in [n-1]$):

- After distributing the first C_i high-value items, the first bundle has items $\{1, n+1, \ldots, k'n+1\}$.

- If $T_i^{\text{idle}} \leq AC_i$, then there are at most AC_i low-value items after the cut C_i. The algorithm gives these to active bundles, and the first bundle keeps its items, giving $k = k'$.
- If $T_i^{\text{idle}} > AC_i$, we can rearrange how the algorithm assigns the low-value items: first give $\{C_i + 1, \ldots, C_i + AC_i\}$ to active bundles, then give $\{m - (T_i^{\text{idle}} - AC_i) + 1, \ldots, m\}$ to active bundles, reserving the middle items for the final phase. This produces a maximin partition where the first bundle is S.

□

Now we can show that choosing the right agent leads to a valid reduction:

Lemma 5.6. *For a goods (respectively, chore) division instance $I = (\mathcal{N}, \mathcal{M}, v)$ with factored personalized bivalued utilities, the pair (i, S) is a valid reduction when i is an agent with the minimum (resp., maximum) value of $\max\{T_i^{idle} - AC_i, 0\}$ and $S = \{1, n+1, \ldots, kn+1\}$, where $k = \lfloor (m - \max\{T_i^{idle} - AC_i, 0\} - 1)/n \rfloor$.*

Proof Sketch. By the previous lemma, S is one of the bundles in a maximin n-partition of agent i, so $v_i(S) \geq \text{MMS}_i$.

For any other agent i', let $S' = \{1, n+1, \ldots, k'n+1\}$ with $k' = \lfloor (m - \max\{T_{i'}^{\text{idle}} - AC_{i'}, 0\} - 1)/n \rfloor$. By our choice of agent i, we have $\max\{T_i^{\text{idle}} - AC_i, 0\} \leq \max\{T_{i'}^{\text{idle}} - AC_{i'}, 0\}$ for goods, and the opposite for chores.

This means $k \geq k'$ for goods division (so $S \supseteq S'$) and $k \leq k'$ for chore division (so $S \subseteq S'$). Therefore, the pair (i, S) satisfies the conditions for a valid reduction. □

Algorithm for computing MMS allocation

Algorithm 4: MMS Allocation for Factored Personalized Bivalued Utilities

```
/* Input:  Instance (N, M, v) with factored personalized bivalued utilities */
                                                    /* Output:  An MMS allocation */
```

if $N = \emptyset$ ***or*** $M = \emptyset$ **then**
  return empty allocation
for *each agent* $i \in N$ **do**
  Compute C_i (the cut point)
  Compute $AC_i \leftarrow \begin{cases} n - (C_i \bmod n), & C_i < m, \\ 0, & \text{otherwise} \end{cases}$
  Compute $T_i^{\text{idle}} \leftarrow \min\{\, p_i \cdot AC_i,\ m - C_i\}$
Choose agent i with minimum (for goods) or maximum (for chores) value of $\max\{\, T_i^{\text{idle}} - AC_i,\ 0\}$
Compute $k \leftarrow \left\lfloor \dfrac{m - \max\{T_i^{\text{idle}} - AC_i, 0\} - 1}{n} \right\rfloor$
$S \leftarrow \{\, 1,\ n+1,\ 2n+1,\ \ldots,\ kn+1\}$
Allocate bundle S to agent i
Recursively compute an MMS allocation for $(N \setminus \{i\},\ M \setminus S,\ v)$
return the combined allocation

6 Achieving Pareto Optimal MMS Allocations

In this section, we show that for weakly lexicographic as well as for factored bivalued instances, we can compute an allocation that is both MMS and Pareto Optimal (PO) in polynomial time. Our approach uses the fact that if x is an MMS allocation, and x' is a Pareto improvement over x, then x' is also MMS. Thus to find an MMS and PO allocation, we can compute an MMS allocation using Theorem 4.1 and then repeatedly find Pareto improvements until we reach a PO allocation. In this section, we will show that we can in polynomial time find Pareto improvements if they exist, and that we will reach a PO allocation after at most polynomially many Pareto improvements.

The key insight is simple but powerful: if an allocation already gives each agent their maximin share (MMS), then making everyone at least as happy and some strictly happier cannot break the MMS guarantee. This makes intuitive sense - if you're already getting your fair share, and then your allocation improves (or stays the same), you're still getting at least your fair share.

This allows us to separate our task into two steps: first find an MMS allocation (which we already showed how to do in previous sections), then improve it to be Pareto optimal without losing the MMS property.

Finding Pareto Improvements

Aziz et al. (2019) proved that in case of goods division with weakly lexicographic or bivalued utilities, one can efficiently test if a given allocation is Pareto optimal (PO). Further, if it is not PO, a Pareto dominating allocation with special properties always exists and can be computed efficiently. The following lemma states their result for goods division, together with an extension to chore division. While in the case of weakly lexicographic utilities our proof almost mirrors their proof for goods, the ideas needed in the case of bivalued utilities are slightly different for chores. Also, the statement below is their claim for weakly lexicographic utilities, while they make a differently worded claim for bivalued utilities; their proof also shows that this claim holds for bivalued utilities.

Lemma 6.1. *Consider a goods or chore division problem with weakly lexicographic or bivalued utilities. It is possible to efficiently check if a given allocation x is Pareto optimal. Moreover, if x is not Pareto optimal, then there exists a sequence of distinct agents $(i_1, \ldots, i_k, i_{k+1} = i_1)$ and a corresponding sequence of distinct items $(r_1, \ldots, r_k, r_{k+1} = r_1)$ such that:*

1. *For each $t \in \{2, \ldots, k+1\}$, item r_t is assigned to agent i_t in x, and $v_{i_t}(r_t) \geq v_{i_t}(r_{t+1})$.*
2. *At least one of these inequalities is strict.*
3. *If we construct a new allocation x' by giving each item r_{t-1} to agent i_t (for all t), then x' is a Pareto improvement over x.*

Such a Pareto improvement x' can be found in polynomial time.

Proof. We focus on the case of chore division, as the goods case is handled in prior work. Suppose x is not Pareto optimal. Among all possible Pareto improvements, pick one, $\hat{x}$, that is as close as possible to x—meaning the number of items that are assigned differently between x and $\hat{x}$ is minimized.

Let i_1 be an agent who strictly benefits in $\hat{x}$ compared to x. This agent must have given away at least one unwanted chore, say c_1, which is now held by another agent i_2 in $\hat{x}$ but was with i_1 in x. Since i_1 is better off in $\hat{x}$, she must have also offloaded at least one other chore.

We can continue this process: for each $t \geq 2$, agent i_t receives c_{t-1} in $\hat{x}$ and, since her utility is weakly lexicographic and she is not worse off, she must have lost another chore c_t (which she had in x but not in $\hat{x}$), with $v_{i_t}(c_t) \geq v_{i_t}(c_{t-1})$. As there are only finitely many agents, eventually we must revisit an agent, closing a cycle: $i_{t+1} = i_\ell$ for some $\ell < t$.

Now, consider the cycle $(i_\ell, \ldots, i_t, i_{t+1} = i_\ell)$. If for some k in this cycle we have $v_{i_k}(c_k) > v_{i_k}(c_{k-1})$ or $v_{i_{k-1}}(c_{k-1}) > v_{i_{k-1}}(c_k)$, then the cycle meets the requirements of the lemma and we are done. If all the relevant values are equal throughout the cycle, then reassigning the items in the cycle back to their original owners in x yields a new allocation that is at least as good as $\hat{x}$ for all agents and strictly better for at least one, and is closer to x than $\hat{x}$ was. This contradicts our choice of $\hat{x}$ as the closest Pareto improvement.

Therefore, such a cycle always exists, and we can use it to construct a Pareto improvement as described. The process of finding such cycles can be implemented efficiently, ensuring polynomial time. □

Chore division,Bivalued Utilities

Let $I = (\mathcal{N}, \mathcal{M}, v)$ be a chore division instance with bivalued utilities and x be an allocation that is not Pareto optimal. Among all Pareto improvements, choose $\hat{x}$ to be the closest to x in the sense of minimizing $|\cup_{i \in N} x_i \setminus \hat{x}_i|$.

First, we show that there is no agent $i \in N$ for whom $|x_i^a| < |\hat{x}_i^a|$. If this were the case, we could take a chore $c \in \hat{x}_i^a \setminus x_i^a$ and give it back to agent $i \in \hat{x}$. The resulting allocation would still be a Pareto improvement over x (agent i would be strictly better off), and it would be closer to x, contradicting our choice of $\hat{x}$.

Next, we show that $|\cup_{i \in N} x_i^a| < |\cup_{i \in N} \hat{x}_i^a|$, i.e., $\hat{x}$ allocates strictly fewer chores to agents who find them difficult than does x. Note that for any allocation y, the social welfare (sum of utilities of agents) under y is:

$|\cup_{i \in N} y_i^a| \cdot a + |\cup_{i \in N} y_i^b| \cdot b = |\cup_{i \in N} y_i^a| \cdot a + (m - |\cup_{i \in N} y_i^a|) \cdot b = |\cup_{i \in N} y_i^a| \cdot (a - b) + m \cdot b$

Because $\hat{x}$ is a Pareto improvement over x, the social welfare under $\hat{x}$ is strictly higher than the social welfare under x. Since $a - b > 0$ (for chores, $a = -1$ and $b = -p < -1$), it follows that $|\cup_{i \in N} \hat{x}_i^a| > |\cup_{i \in N} x_i^a|$.

Consider a chore $c_1 \in \cup_{i \in N} \hat{x}_i^a \setminus \cup_{i \in N} x_i^a$. Suppose $c_1 \in \hat{x}_{i_2} \cap x_{i_1}$, for some $i_1 \neq i_2$. We can represent this as $i_1 \xrightarrow{c_1}_a i_2$, where the arrow connecting agent i with chore c means that i is giving away c, the entry above indicates $v_i(c)$ while the entry below indicates the allocation in which c is allocated to i. Consider extending this chain as much as possible by alternating $\to_a$ and $\to_b$, to obtain:

$i_1 \xrightarrow{c_1}_a i_2 \xrightarrow{c_2}_b i_3 \xrightarrow{c_3}_a \ldots i_t$

There are two possibilities: either the chain stops at agent i_t for some $t \geq 2$ (and we are unable to extend it further), or an agent repeats at some point (i.e., $i_t = i_\ell$ for some $\ell < t$).

Case 1: The chain stops at agent i_t. First, suppose $\hat{x}_{i_t}^b \neq \emptyset$. Consider any chore $\tilde{c} \in \hat{x}_{i_t}^b$. Consider the allocation obtained by starting from x and cyclically shifting chores as follows: chore c_k is moved to agent i_{k+1} for $k \in [t-1]$, and chore $\tilde{c}$ is moved to agent i_1. Note that agent i_1 loses a b-valued chore and gains an a-valued chore, agents i_2 through i_{t-1} each lose an a-valued chore and gain a b-valued chore, and agent i_t loses a b-valued chore and gains an a-valued chore. Thus, this is the kind of cycle sought in the lemma.

Now, suppose $\hat{x}_{i_t}^b = \emptyset$. In this case, we have shown that $|x_{i_t}^a| < |\hat{x}_{i_t}^a|$, which contradicts our earlier observation.

Case 2: An agent repeats, i.e., $i_t = i_\ell$ for some $\ell < t$. Without loss of generality, assume $\ell = 1$ (otherwise, consider the cycle starting at i_ℓ). If the cycle is of the form $i_1 \xrightarrow{c_1}_a i_2 \xrightarrow{c_2}_b i_3 \ldots \to_b i_1$, then shifting all chores along the cycle (chore c_k is moved to agent i_{k+1} for each k) produces a Pareto

improvement of the desired form. Otherwise, the cycle must be of the form $i_1 \xrightarrow{c_1}_a i_2 \xrightarrow{c_2}_b i_3 \ldots \to_a i_1$. In this case, we cannot directly use this cycle because the conditions of the lemma require agent $i_t = i_1$ to value chore c_t at least as much as chore c_1, but we have $v_{i_1}(c_t) = a > b = v_{i_1}(c_1)$. However, we can use the trick of traversing the cycle twice, which satisfies the conditions of the lemma.

In either case, we find a cycle of agents and chores that satisfies the conditions of the lemma, and the corresponding reallocation yields a Pareto improvement. The fact that we can efficiently find such a cycle follows from standard graph algorithms.

Potential Function and Convergence to PO

To prove that repeated Pareto improvements lead to a PO allocation in polynomial time, we use a potential function approach. For any allocation y, define the potential function $\phi(y) = \sum_{i \in N} \sum_{r \in M} h(i, r)$, where $h(i, r)$ is the index of agent i in the sorted order of agents based on their valuation of item r. Note that $0 \leq \phi(y) \leq m^2$ since each item contributes at most n to the sum and there are m items.

Theorem 6.2. *For goods or chore division with weakly lexicographic or factored bivalued utilities, starting from any MMS allocation, repeatedly applying the Pareto improvement from Lemma 4.14 yields a Pareto optimal allocation in polynomial time.*

Proof. We analyze the convergence for bivalued utilities and weakly lexicographic utilities separately:

For bivalued utilities: Consider any Pareto improvement from x to x' obtained by a cycle of items $(r_1, ..., r_k, r_{k+1} = r_1)$ as in Lemma 4.14. With bivalued utilities, each item has only two possible values for each agent (either 1 and $p > 1$ for goods, or -1 and $-p < -1$ for chores).

When an item is reallocated in the cycle, it moves from an agent who values it less to an agent who values it more. For each such move, the social welfare increases by a discrete amount. Since there are only two possible values and m items, the total number of such improvements is bounded by $O(m)$.

For weakly lexicographic utilities: Here we use the potential function more directly. For any agent i, note that for every item r_t that she loses, she gains a unique item r_{t-1} with $v_i(r_{t-1}) \geq v_i(r_t)$, which implies $h(i, r_{t-1}) \leq h(i, r_t)$ for goods division and the opposite inequality for chore division.

Further, because the Pareto improvement strictly improves the utility of some agent, the inequality for that agent must be strict for at least one item. Each Pareto improvement changes the potential by at least 1, and the potential is bounded between 0 and m^2. Thus, the process must terminate in at most $O(m^2)$ steps.

Therefore, for both utility classes, the algorithm converges to a Pareto optimal allocation in polynomial time. □

Algorithm for MMS+PO Allocation

Algorithm 5: Computing MMS+PO Allocation

```
/* Input:Instance (N, M, v) with weakly lexicographic or factored bivalued utilities */
                                        /* Output:  An allocation that is both MMS and PO */
```

Compute an MMS allocation x using the algorithm from Section 4.2 or 4.3
while x *is not Pareto optimal*. **do**
  Find a cycle of agents $(i_1, \ldots, i_k, i_{k+1} = i_1)$ and a cycle of items $(r_1, \ldots, r_k, r_{k+1} = r_1)$ satisfying Lemma 4.14
  for $t \in \{2, \ldots, k+1\}$ **do**
    Reallocate item r_{t-1} to agent i_t
return x

Corollary 6.3. *In every goods division or chore division instance with weakly lexicographic or factored bivalued utilities, an MMS and PO allocation always exists and can be computed in polynomial time.*

Proof. By Theorem 4.1, we can compute an MMS allocation in polynomial time. By repeatedly applying Pareto improvements according to Lemma 4.14, we reach a PO allocation in at most $O(m^2)$ steps for weakly lexicographic utilities and $O(m)$ steps for bivalued utilities. Since Pareto improvements preserve the MMS property, the final allocation is both MMS and PO. □

Note that none of our results about PO in this section apply to personalized bivalued utilities. This is because with personalized values, the potential function may not have polynomial bounds, making it impossible to guarantee polynomial-time convergence. Obtaining a polynomial-time algorithm for finding an MMS and PO allocation for personalized bivalued instances remains an open problem.

Conclusion

This report has explored the key results presented in the paper "How to Fairly Allocate Easy and Difficult Chores," focusing on the fair allocation of indivisible chores with bivalued utilities. We defined the problem setting, the bivalued utility model, and relevant fairness and efficiency concepts including EF1, PO, and MMS. A major contribution highlighted is the polynomial-time algorithm for finding an allocation that is simultaneously Pareto optimal and envy-free up to one item for bivalued chores. The correctness of this algorithm was examined through its reliance on maintaining equilibrium and satisfying a set of inductive hypotheses across its iterative phases. Furthermore, the paper also establishes the existence and efficient computability of Maximin Share allocations in this same bivalued setting. Collectively, these findings demonstrate that despite the challenges of chore allocation, strong fairness and efficiency guarantees are achievable and computationally tractable when agent preferences exhibit this specific bivalued structure.