Efficient Nearly-Fair Division with Capacity Constraints

Hila Shoshan
Ariel University
Ariel, Israel
hilashoshan0605@gmail.com

Noam Hazon
Ariel University
Ariel, Israel
noamh@ariel.ac.il

Erel Segal-Halevi
Ariel University
Ariel, Israel
erelsgl@gmail.com

Abstract
We consider the problem of fairly and efficiently allocating indivisible items (goods or bads) under capacity constraints. In this setting, we are given a set of categorized items. Each category has a capacity constraint (the same for all agents), that is an upper bound on the number of items an agent can receive from each category. Our main result is a polynomial-time algorithm that solves the problem for two agents with additive utilities over the items. When each category contains items that are all goods (positively evaluated) or all chores (negatively evaluated) for each of the agents, our algorithm finds a feasible allocation of the items, which is both Pareto-optimal and envy-free up to one item. In the general case, when each item can be a good or a chore arbitrarily, our algorithm finds an allocation that is Pareto-optimal and envy-free up to one good and one chore. Full version is available at arXiv [36].

Keywords
Fair division; Indivisible items; Mixed manna; Capacity constraints

ACM Reference Format:
Hila Shoshan, Noam Hazon, and Erel Segal-Halevi. 2023. Efficient Nearly-Fair Division with Capacity Constraints. In Proc.of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023, IFAAMAS, 13 pages.

1 Introduction
The problem of how to fairly divide a set of items among agents with different preferences has been investigated by many mathematicians, economists, political scientists and computer scientists. Most of the earlier work focused on how to fairly divide goods, i.e., items with non-negative utility. In recent years, several works have considered the division of chores, i.e., items with non-positive utility, and a few works also considered the division of a mixture of goods and chores (for example, Aziz et al. [3] and Bérczi et al. [7]). Indeed, items may be considered as goods for one agent and as chores for another agent. For example, consider a project that has to be completed by a team of students. It consists of several tasks, setting a capacity for each category. For example, if the team consists of two students, and there are 5 programming tasks, 6 UI tasks and 4 algorithm tasks, then a capacity of 3 on programming and UI tasks and a capacity of 2 on algorithm tasks would ensure that both students are involved in about the same number of tasks from each category. Clearly, the capacity constraints should be large enough so that all of the items in a given category could be assigned to the agents. An allocation satisfying all capacity constraints is called feasible.

We focus on allocation problems between two agents. This case is practically important. For example, student projects are often assigned to one agent. Considering again the student project example, the mentor of the project may want all students to be involved in all aspects of the project. Therefore, the mentor may partition the project tasks into three categories: programming, UI, and algorithms, setting a capacity for each category. For example, if the team consists of two students, and there are 5 programming tasks, 6 UI tasks and 4 algorithm tasks, then a capacity of 3 on programming and UI tasks and a capacity of 2 on algorithm tasks would ensure that both students are involved in about the same number of tasks from each category. Clearly, the capacity constraints should be large enough so that all of the items in a given category could be assigned to the agents. An allocation satisfying all capacity constraints is called feasible.

Note that, without capacity constraints, if one agent evaluates an item as a good, while another agent evaluates it as a chore, we can simply give it to the agent who evaluates it as a good, as done by Aziz et al. [3]. However, with capacities it may not be possible, which shows that the combination of capacities and mixed valuations is more difficult than each of these on its own.

Two important considerations in item allocation are efficiency and fairness. As an efficiency criterion, we use Pareto optimality (PO), which means that no other feasible allocation is at least as good for all agents and strictly better for some agent. As fairness criteria, we use two relaxations of envy-freeness (EF). The stronger one is envy-freeness up to one item (EF1), which was introduced by Budish [17], and adapted by Aziz et al. [3] for a mixture of goods and chores. Intuitively, an allocation is EF1 if for each pair of agents $i,j$, after removing the most difficult chore (for $i$) from $i$’s bundle, or the most valuable good (for $i$) from $j$’s bundle, $i$ would not be jealous of $j$.

With capacity constraints, an EF1 allocation may not exist. For example, consider a scenario with one category with two items, $o_1$ and $o_2$, and capacity constraint of 1. $o_1$ is a good for both agents (e.g., $u_1(o_1) = u_2(o_1) = 1$), and $o_2$ is a chore for both agents (e.g., $u_1(o_2) = u_2(o_2) = -1$). Clearly, in every feasible allocation, one agent must receive the good and the other agent must receive the chore (due to the capacity constraint), and thus the allocation is not EF1. Therefore, we introduce a natural relaxation of it, which we call envy-freeness up to one good and one chore (EF[1,1]). It means that, for each pair of agents $i,j$, there exists a chore in $i$’s bundle, and a good in $j$’s bundle, such that both are in the same category, and after removing them, $i$ would not be jealous of $j$. In the special case in which, for each agent and category, either all items are goods or all items are chores (as in the student project example above), EF[1,1] is equivalent to EF1. We call this special case a same-sign instance; note that it is still more general than only-goods or only-chores settings.

We focus on allocation problems between two agents. This case is practically important. For example, student projects are often
done in teams of two, and household chores are often carried out by the two partners. Fair allocation among two agents is the focus of various papers on fair division [2, 7, 13, 14, 28, 33, 34, 39].

We prove the existence of PO and EF[1,1] allocations with capacity constraints for two agents with arbitrary (positive or negative) utilities over the items. The proof is constructive: we provide a polynomial-time algorithm that, for two agents, returns an allocation that is both PO and EF[1,1]. In a same-sign instance, the returned allocation is PO and EF[1,1].

Our focus on the case of two agents allows us to simultaneously make two advancements over the state-of-the-art in capacity-constrained fair allocation [10, 21]: First, we handle a mixture of goods and chores, rather than just goods. As we show in Appendix A in the full version [36], standard techniques used for goods are not applicable for mixed utilities. Second, we attain an allocation that is not only fair but also PO. Before this work, it was not even known if a PO and EF[1,1] allocation of goods with capacity constraints always exists.

Our algorithm is based on the following ideas. The division problem can be considered as a matching problem on a bipartite graph, in which one side represents the agents and the other side represents the items. We add dummy items and clones of agents such that in every matching the capacity constraints are guaranteed. We assign a positive weight to each agent. We assign, to each edge between an agent and an item, a weight which is the product of the agent’s weight and the valuation of the agent to the item. A maximum-weight matching in this graph represents a feasible allocation that maximizes a weighted sum of utilities. Every allocation that maximizes a weighted sum of utilities, with positive agent weights, is Pareto-optimal.\(^1\) Our algorithm first computes a maximum-weight matching that is also envy-free (EF) for one of the agents. It then tries to make it EF[1,1], while maintaining it a maximum-weight matching, by identifying pairs of items that can be exchanged between the agents, based on a ratio that captures how much one agent prefers an item relative to the other agent’s preferences. Every exchange of items is equivalent to increasing the jealous agent’s weight and decreasing the other agent’s weight.

2 Related Work

Fair division problems vary according to the nature of the objects being divided, the preferences of the agents, and the fairness criteria. Many algorithms have been developed to solve fair division problems, for details see the surveys of such algorithms [15], [31], [12], [11].

In this paper we consider a new setting, which combines goods, chores, capacity constraints and Pareto-optimal. Note that even ignoring PO, goods, or both, our result is new.

2.1 Mixtures of Goods and Chores

Bérczi et al. [7] present a polynomial-time algorithm for finding an EF[1,1] allocation for two agents with arbitrary utility functions (positive or negative). Chen and Liu [20] proved that the lexicmin solution is EFX (a property stronger than EF[1,1]) for combinations of goods and chores, for agents with identical valuations. Gafni et al. [23] present a generalization of both goods and chores, by considering items that may have several copies. All these works do not consider efficiency. Efficiency in a setting with goods and chores is studied by Aziz et al. [3]. They use the round-robin technique for finding an EF[1,1] and PO division of combinations of goods and chores between two agents. Similarly, Aziz et al. [4] find an allocation that is PROP1 (a property weaker than EF[1,1]) and PO for goods and chores. Aleksandrov and Walsh [1] prove that, with some constraints, EFX and PO allocations always exist for mixed items. However, all of these works do not handle capacity constraints.

2.2 Constraints

When all agents have weakly additive utilities, the round-robin protocol finds a complete EF[1,1] division in which all agents receive approximately the same number of goods [18]. This technique, together with the envy-graph, has been used for finding a fair division of goods under capacity constraints [10]. This work has been extended to heterogeneous capacity constraints [21], and to maximin-share fairness [26].

Fair allocation of goods of different categories has been studied by Mackin and Xia [30]. The constraint is that each agent must receive at least one item per category. Sikdar et al. [37] consider an exchange market in which each agent holds multiple items of each category and should receive a bundle with exactly the same number of items of each category. Nyman et al. [35] study a similar setting (they call the categories “houses” and the items “rooms”), but with monetary transfers (“rent”).

Several other constraints have been considered. For example, Bilò et al. [9] study the fair division of goods such that each bundle needs to be connected on an underlying graph. Igarashi and Peters [27] study PO allocation of goods with connectivity constraints. An overview of the different types of constraints that have been considered can be found in [38].

2.3 Efficiency and Fairness

There are several techniques for finding a division of goods that is EF[1,1] and PO. For example, the Maximum Nash Welfare algorithm selects a complete allocation that maximizes the product of utilities. It assumes that the agents’ utilities are additive, and the resulting allocation is both EF[1,1] and PO [18, 42].

In the context of fair cake-cutting (fair division of a continuous resource), Weller [41] proved the existence of an EF and PO allocation by considering the set of all allocations that maximize a weighted sum of utilities. We adopted this technique for the setting with indivisible items and capacity constraints. Barman et al. [6] present a price-based mechanism that finds an EF[1,1] and PO allocation of goods in pseudo-polynomial time. Similarly, Barman and Krishnamurthy [5] use a price-based approach to show that fair and efficient allocations can be computed in strongly polynomial time. The price-based approach can be seen as a “dual” of our weight-based approach.

Garg et al. [24] present an algorithm for EF[1,1] and PO allocation of chores when agents have bivalued preferences. With general additive preferences, the existence of an EF[1,1] and PO allocation of chores for three agents (without capacity constraints) was proved.
only very recently by Garg et al. [25]. The authors claim that "the case of chores turns out to be much more difficult to work with, resulting in relatively slow progress despite significant efforts by many researchers". Indeed, for four or more agents, existence is still open even for only-chores instances and without capacity constraints.

2.4 Alternative Techniques

Our setting combines a mixture of goods and chores, capacity constraints, and a guarantee of both fairness and efficiency. These three issues were studied in separation, but not all simultaneously. Although previous works have developed useful techniques, they do not work for our setting. For example, using the top-trading graph presented by Bhaskar et al. [8] for dividing chores does not work when there are capacity constraints. The reason is that if we allocate an item to the 'sink' agent (i.e., an agent that does not envy any agent) on the top-trading graph, we may exceed the capacity constraints. As another example, consider the maximum-weighted matching algorithm of Brustle et al. [16]. It is not hard to modify the algorithm to work with chores, but adding capacity constraints on each category might not maintain the EF1 property between the categories. See Appendix A in the full version [36] for more details.

Therefore, in this paper we develop a new technique for finding PO and EF1 (or EF1,1) allocation of the set of items (goods and chores), that also maintains capacity constraints.

Table 1 summarizes some of the previous results mentioned in this section, which are close to our setting.

3 Notations

An instance of our problem is a tuple $I = (N, M, C, S, U)$:

- $N = \{i\}$ is a set of $n$ agents.
- $M = \{o_1, \ldots, o_m\}$ is a set of $m$ items.
- $C = (C_1, C_2, \ldots, C_k)$ is a set of $k$ categories. The categories are pairwise-disjoint and $M = \bigcup C_j$.
- $S = (s_1, s_2, \ldots, s_k)$ is a list of size $k$, containing the capacity constraint of each category. We assume that $\forall j \in [k]: \frac{|C_j|}{n} \leq s_j \leq |C_j|$, $s_j \in \mathbb{N}$. The lower bound is needed to ensure we can divide all the items, and not "throw" anything away, and the upper bound is a trivial bound used for computing the run-time.
- $U$ is an $n$-tuple of utility functions $u_i : M \rightarrow \mathbb{R}$. We assume additive utilities, that is, $u_i(X) := \sum_{o \in X} u_i(o)$ for $X \subseteq M$.

In a general mixed instance, each utility can be any real number (positive, negative or zero). A same-sign instance is an instance in which, for each agent $i \in N$ and category $j \in [k]$, $C_j$ contains only goods for $i$ or only chores for $i$. That is, either $u_i(o) \geq 0$ for all $o \in C_j$, or $u_i(o) \leq 0$ for all $o \in C_j$. Note that, even in a same-sign instance, it is possible that each agent evaluates different categories as goods or chores, and that different agents evaluate the same item differently.

An allocation is a vector $A := (A_1, A_2, \ldots, A_n)$, with $\forall i, j \in [n], i \neq j : A_i \cap A_j = \emptyset$ and $\cup_{i \in [n]} A_i = M$. $A_i$ is called 'agent $i$'s bundle'.

An allocation $A$ is called feasible if for all $i \in [n]$, the bundle $A_i$ contains at most $s_i$ items of each category $C_e$, for each $e \in [k]$.

Definition 3.1 (Due to Aziz et al. [3]). An allocation $A$ is called Envy Free up to one item (EF1) if for all $i, j \in N$, at least one of the following holds:

- $\exists T \subseteq A_i$ with $|T| \leq 1$, s.t. $u_i(A_i \setminus T) \geq u_i(A_j)$.
- $\exists G \subseteq A_j$ with $|G| \leq 1$, s.t. $u_i(A_i) \geq u_i(A_j \setminus G)$.

We also define a slightly weaker fairness notion, that we need for handling general mixed instances, in which an EF1 allocation is not guaranteed to exist, as shown in Introduction.

Definition 3.2. An allocation $A$ is called Envy Free up to one good and one chore (EF1,1) if for all $i, j \in N$, there exists a set $T \subseteq A_i$ with $|T| \leq 1$, and a set $G \subseteq A_j$ with $|G| \leq 1$, such that $G$ and $T$ are of the same category, and $u_i(A_i \setminus T) \geq u_i(A_j \setminus G)$.

The uncategorized setting of Aziz et al. [3] can be reduced to our setting by putting each item in its own category, with a capacity of 1. An allocation is EF1,1 in the categorized instance if-and-only-if it is EF1 (by Definition 3.1) in the original instance.

Throughout the paper, any result that is valid for mixed instances with EF1,1 is also valid for same-sign instances with EF1. This follows from the following lemma.

Lemma 3.3. In a same-sign instance, EF1,1 is equivalent to EF1.

Proof. Suppose that some allocation, $A$, for a same-sign instance is EF1,1. Therefore, for all $i, j \in N$, $\exists T \subseteq A_i$ with $|T| \leq 1$, and $\exists G \subseteq A_j$ with $|G| \leq 1$, such that $G$ and $T$ are of the same category, and $u_i(A_i \setminus T) \geq u_i(A_j \setminus G)$.

If $|G| > 0$ or $|T| > 0$, then $A$ is EF1, by definition. So assume that $|G| = |T| = 1$. Since $G$ and $T$ are in the same category, and in a same-sign instance, for each agent $i \in [n]$ and category $c \in [k]$, $C_c$ contains only goods for $i$ or only chores for $i$, then, for all $j \in [n]$.

If $C_c$ is a category of goods for agent $i$, then $u_i(A_i \setminus T) \geq u_i(A_j \setminus G)$ implies $u_i(A_j) \geq u_i(A_j \setminus G)$, so both allocations are EF1 for agent $i$. If $C_c$ is a category of chores for agent $i$, then $u_i(A_i \setminus T) \geq u_i(A_j \setminus G)$ implies $u_i(A_i \setminus T) \geq u_i(A_j)$, so again both allocations are EF1 for agent $i$.

Remark 3.4. Our new EF1,1 is reminiscent of another guarantee called EF1,1, that is, envy-freeness up to adding a good to one agent and removing a good from another agent [5]. But lemma 3.3 implies that EF1,1 is stronger. The reason is that if there are only goods, it is enough to remove one good from an agent’s bundle, and there is no need to also add a good to the envious agent’s bundle.

EF1,1 can be seen as a generalization of EF1 as defined in [Aziz et al. 2022] to the case of categorized items (you just have to define one category for every item, with an upper bound equal to one).

Remark 3.5. The restriction in Definition 3.2 that $G$ and $T$ should be of the same category is essential for Lemma 3.3. To see this, denote by EF1,1,U the unrestricted variant of EF1,1, allowing to remove one chore and one good from any category. Suppose that there are two categories: one of them contains a good (for both agents) and the other contains a chore (for both agents). If one agent gets the good and the other agent gets the chore, the allocation is EF1,1,U, and it is a same-sign instance, but it is not EF1.

Any EF1,1 allocation is clearly EF1,1,U. Therefore, proving that our algorithm returns an EF1,1 allocation implies two things
Table 1: Summary of some works on fair allocation of indivisible items

| paper | agents | utilities | goods | chores | constraints | fairness | PO | result |
|-------|--------|-----------|-------|--------|-------------|---------|----|--------|
| [7]   | 2      | arbitrary | v     | v      | -           | EF1     | -  | polynomial-time algorithm |
| [20]  | any    | identical | v     | v      | -           | EFX     | -  | the leximin solution       |
| [23]  | any    | leveled   | v     | v      | -           | EFX     | -  | existence proof            |
| [3]   | 2      | arbitrary | v     | v      | -           | EF1     | v  | round-robin technique      |
| [4]   | any    | arbitrary | v     | v      | -           | PROP1   | v  | polynomial-time algorithm  |
| [1]   | any    | tertiary  | v     | v      | -           | EFX     | v  | existence proof            |
| [18]  | any    | weakly    | v     | -      | approximately the same number | EF1     | -  | round-robin protocol       |
| [10]  | any    | additive  | v     | -      | capacity constraints | EF1     | -  | round-robin protocol and envy-graph |
| [21]  | any    | heterogenous | v    | -      | heterogenous capacity constraint | EF1     | -  | polynomial-time algorithm  |
| [26]  | any    | additive  | v     | -      | capacity constraint | MMS     | -  | polynomial-time algorithm  |
| [30]  | any    | heterogenous and combinatorial | v     | -      | each agent gets at least one item per category | egalitarian rank | -  | characterize egalitarian + utilitarian rank-efficiency of categorical sequential allocation mechanisms |
| [9]   | any    | identical | v     | -      | each bundle needs to be connected on an underlying graph | EF1     | -  | polynomial-time algorithm  |
| [27]  | any    | additive  | v     | -      | bundles must be connected in an underlying item graph | EF1     | v  | non-existence on a path graph |
| [18]  | any    | additive  | v     | -      | -           | EF1     | v  | max Nash welfare algorithm |
| [42]  | any    | additive  | v     | -      | each agent has a budget constraint on the total cost of items she receives | 1/4-EF1 | v  | max Nash welfare algorithm |
| [6]   | any    | additive  | v     | -      | -           | EF1     | v  | pseudo-poly. time algorithm |
| [8]   | any    | additive  | -     | v      | -           | EF1     |     | polynomial-time algorithm  |
| [16]  | any    | additive  | v     | -      | -           | EF1     |     | max weighted matching      |
| [19]  | 3      | additive  | v     | -      | -           | EFX     |     | existence proof            |
| [24]  | any    | additive, bivalued | v    | -      | -           | EF1     | v  | polynomial-time algorithm  |

We 2 additive v v capacity constraints EF1 || EF[1,1] v polynomial-time algorithm

at once: in general instances, it returns an EF[1,1,U] allocation; and in same-sign instances, our algorithm returns an EF1 allocation.

Finally, we recall two definitions:

**Definition 3.6.** Given an allocation $A$ for $n$ agents, the envy graph of $A$ is a graph with $n$ nodes, each represents an agent, and there is a directed edge $i \rightarrow j$ iff $i$ envies $j$ in allocation $A$. A cycle in the envy graph is called an envy cycle.

Our efficiency criterion is defined next:

**Definition 3.7.** Given an allocation $A$, another allocation $A'$ is a Pareto-improvement of $A$ if $u_i(A'_i) \geq u_i(A_i)$ for all $i \in N$, and $u_j(A'_j) > u_j(A_j)$ for some $j \in N$.

A feasible allocation $A$ is Pareto-Optimal (PO) if no feasible allocation is a Pareto-improvement of $A$.

4 Finding a PO and EF[1,1] Division

In this section, we present some general notions that can be used for any number of agents.

Then, we present our algorithm that finds in polynomial time a feasible PO allocation with two agents. In any mixed instance, this allocation is also EF[1,1]; in a same-sign instance, it is also EF1, according to Lemma 3.3.

4.1 Preprocessing

We preprocess the instance such that, in any feasible allocation, all bundles have the same cardinality. To achieve this, we add to each category $C_c$ with capacity constraint $c$, some $ns_c - [c]$ dummy items with a value of 0 to all agents. In the new instance, each bundle must contain exactly $c$ items from each category $C_c$. From now on, without loss of generality, we assume that $|M| = m = \sum c \lfloor k \rfloor ns_c$. 209
This implies that, in every feasible allocation \( A \), we have \(|A_i| = m/n\) for all \( i \in [n] \).

### 4.2 Maximizing a Weighted Sum of Utilities

Our algorithm is based on searching the space of PO allocations. Particularly, we consider allocations that maximize a weighted sum of utilities \( w_1 u_1 + w_2 u_2 + \ldots + w_n u_n \), where each agent \( i \) is associated with a weight \( w_i \in [0, 1] \), and \( w_1 + w_2 + \ldots + w_n = 1 \). Such allocations can be found by solving a maximum-weight matching problem in a weighted bipartite graph. We denote the set of all agents’ weights by \( w = (w_1, w_2, \ldots, w_n) \).

**Definition 4.1.** For any \( n \) real numbers (weights) \( w = (w_1, w_2, \ldots, w_n) \), such that, \( \forall i \in [n], w_i \in [0, 1] \), and \( w_1 + w_2 + \ldots + w_n = 1 \), let \( G_w \) be a bipartite graph \( (V_1 \cup V_2, E) \) with \( |V_1| = |V_2| = m \). \( V_2 \) contains all \( m \) items (of all categories, including dummies). \( V_1 \) contains \( \frac{w_i}{w_j} \) copies of each agent \( i \in [n] \). For each category \( c \in [k] \), we choose distinct \( x_c \) copies of each agent and add an undirected edge from each of them to all the \( n_s \) items of \( C_c \). Each edge \( (i, o) \in E, i \in V_1, o \in V_2 \) has a weight \( w(i, o) \), where:

\[
  w(i, o) := w_i \cdot u_i(o)
\]

An allocation is called \( w \)-maximal if it corresponds to a maximum-weight matching among the maximum-cardinality matchings in \( G_w \).

**Proposition 4.2.** Every \( w \)-maximal allocation, where \( w_1, w_2, \ldots, w_n \in (0, 1) \), is PO.

**Proof.** Every \( w \)-maximal allocation \( A = (A_1, A_2, \ldots, A_n) \) maximizes the sum \( w_1 u_1(A_1) + w_2 u_2(A_2) + \ldots + w_n u_n(A_n) \). Every Pareto-improvement would increase this sum. Therefore, there can be no Pareto-improvement, so \( A \) is PO.

### 4.3 Exchanging Pairs of Items

Our algorithm starts with a \( w \)-maximal allocation, and repeatedly exchanges pairs of items between the agents in order to find an allocation that is also \( \text{EF}[1,1] \). To determine which pairs to exchange, we need some definitions and lemmas.

**Definition 4.3.** Given a feasible allocation \( A = (A_1, A_2, \ldots, A_n) \), an exchangeable pair is a pair \( (o_i, o_j) \) of items, \( o_i \in A_i \) and \( o_j \in A_j \), \( i, j \in [n] \), such that \( o_i \) and \( o_j \) are in the same category. This implies that \( A_i \setminus \{o_i\} \cup \{o_j\} \) and \( A_j \setminus \{o_j\} \cup \{o_i\} \) are both feasible. Additionally, in a same-sign instance, for each agent, \( o_i, o_j \) are in the same “type”, that is, both goods or both chores.

In this paper, we work a lot with exchangeable pairs, so we use \( o_i, o_j \in A_i, A_j \) as a shorthand for “\( o_i \in A_i \) and \( o_j \in A_j \)”.

#### 4.3.1 Finding a Fair Allocation

The following two lemmas deal with fairness while exchanging exchangeable pairs in a \( w \)-maximal allocation.

**Lemma 4.4.** Let \( A \) be a \( w \)-maximal feasible allocation, and let \( A' \) be another feasible allocation, resulting from \( A \) by exchanging an exchangeable pair \( (o_i, o_j) \) between some two agents \( i \neq j \). Then there exists some ordering of the agents, \( k_1, \ldots, k_n \), such that for all \( y > x \), the \( \text{EF}[1,1] \) condition is satisfied for agent \( k_y \) with respect to agent \( k_x \) in both allocations \( A \) and \( A' \). That is, \( k_y \) envies \( k_x \) up to one good and one chore in both allocations.

In particular, there is at least one agent (agent \( k_n \)) for whom both \( A \) and \( A' \) are \( \text{EF}[1,1] \).

**Proof.** Let \( A = (A_1, \ldots, A_n) \) and \( A' = (A'_1, \ldots, A'_n) \). Let \( C_c \) be the category that contains both items \( o_i, o_j \). By the pre-processing step, every bundle in \( A \) contains at least one item from \( C_c \). So we can write every bundle \( A_x \), for all \( x \in [n] \), as: \( A_x = B_x \cup \{o_x\} \) for some \( o_x \in C_c \). After the exchange, we have for all \( x \neq i, j \cdot A'_x = A_x = B_x \cup \{o_x\} \), whereas \( A'_i = B_i \cup \{o_j\}, A'_j = B_j \cup \{o_i\} \).

Consider the envy-graph representing the partial allocation \((B_1, B_2, \ldots, B_n)\). We claim that it contains no cycle. Suppose that it contained an envy-cycle. If we replaced the bundles according to the direction of edges in the cycle, we would get another feasible allocation which is a Pareto-improvement of the current allocation, \( A \), which is \( w \)-maximal. Contradiction!

Therefore, the envy-graph of \((B_1, B_2, \ldots, B_n)\) has a topological ordering. Let \( k_1, \ldots, k_n \) be such an ordering, so that for all \( y > x \), agent \( k_y \) prefers \( B_{k_y} \) over \( B_{k_x} \). In both allocations \( A \) and \( A' \), the bundles of both \( k_y \) and \( k_x \) are derived from \( B_{k_y} \) and \( B_{k_x} \) by adding a single good or chore. Therefore, in both \( A \) and \( A' \), the \( \text{EF}[1,1] \) condition is satisfied for agent \( k_y \) w.r.t. agent \( k_x \). In particular, for agent \( k_n \), both these allocations are \( \text{EF}[1,1] \).

□

Lemma 4.4 considered a single exchange. Now, we consider a sequence of exchanges. The following lemma works only for two agents — we could not yet extend it to more than two agents.

**Lemma 4.5.** Suppose there are \( n = 2 \) agents. Suppose there is a sequence of feasible allocations \( A^1, \ldots, A^x \) with the following properties:

- \( \forall j \in [x] \), the allocation \( A^j = (A^j_1, A^j_2) \) is \( \text{w-maximal} \), where \( w = (w_{1,j}, w_{2,j}) \) for some \( w_{1,j}, w_{2,j} \in (0, 1) \).
- \( A^1 \) is \( \text{EF} \) for agent 1 and \( A^x \) is \( \text{EF} \) for agent 2.
- \( \forall j \in [x-1], A^{j+1} \) is obtained from \( A^j \) by a single exchange of an exchangeable pair between the agents.

Then, for some \( j \in [x] \), the allocation \( A^j \) is \( \text{PO} \) and \( \text{EF}[1,1] \).

**Proof.** Every \( A^j \) is \( \text{PO} \) by Proposition 4.2. Therefore, it is never possible for the two agents to envy each other simultaneously. Since at \( A^j \) agent 1 is not jealous and at \( A^j \) agent 2 is not jealous, there must be some \( j \in [x-1] \) in which \( A^j \) is \( \text{EF} \) for 1, and \( A^{j+1} \) is \( \text{EF} \) for 2.

Because \( A^{j+1} \) results from \( A^j \) by exchanging an exchangeable pair between the agents, by Lemma 4.4, there exists an agent \( i \in [2] \) such that both \( A^j \) and \( A^{j+1} \) are \( \text{EF}[1,1] \) for \( i \).

If both are \( \text{EF}[1,1] \) for agent 1, then \( A^{j+1} \) is an \( \text{EF}[1,1] \) allocation. If both are \( \text{EF}[1,1] \) for agent 2, then \( A^j \) is an \( \text{EF}[1,1] \) allocation.

□

To apply Lemma 4.5, we need a way to choose the pair of exchangeable items in each step of the sequence, so that the next allocation in the sequence remains \( w \)-maximal. We use the following definition.

\[\text{In fact, the result holds not only for an exchange of two items, but also for any permutation of \( n \) items of the same category, one item per agent. The proof is the same.}\]
Definition 4.6. For a pair of agents $i, j \in [n]$ s.t. $i \neq j$, and a pair of items $(o_i, o_j)$, the difference ratio, denoted by $r_{ji}(o_i, o_j)$, is defined as:
\[ r_{ji}(o_i, o_j) := \frac{u_j(o_i) - u_j(o_j)}{u_i(o_i) - u_i(o_j)} \]
If $u_j(o_i) = u_j(o_j)$, then the ratio is always $0$. If $u_j(o_i) = u_j(o_j)$ and $u_j(o_i) \neq u_j(o_j)$, then the ratio is defined as $+\infty$ if $u_j(o_i) > u_j(o_j)$, or $-\infty$ if $u_j(o_i) < u_j(o_j)$.

4.3.2 The Properties of a $w$-maximal Allocation

The following lemma is proved in Appendix C in the full version [36].

Lemma 4.7. For any $n$ agents, for any $w = (w_1, w_2, \ldots, w_n)$ such that $w_1, w_2, \ldots, w_n \in (0, 1)$, and an allocation $A = (A_1, \ldots, A_n)$, the following are equivalent:
(i) $A$ is $w$-maximal,
(ii) Every exchange-cycle does not increase the weighted sum of utilities. That is, for all $x \geq 2$, a subset of agents $\{a_1, \ldots, a_x\} \in [n]$, and a set of items $o_1, \ldots, o_x$, such that all are in the same category, and $V \in [x]$, $o_j \in A_{v_j}$:
\[ w_{a_1}u_{a_1}(o_1) + w_{a_2}u_{a_2}(o_2) + \ldots + w_{a_x}u_{a_x}(o_x) \geq w_{a_1}u_{a_1}(o_1') + w_{a_2}u_{a_2}(o_1') + \ldots + w_{a_x}u_{a_x}(o_x') \]

The following lemma follows from Lemma 4.7, but only for two agents.

Lemma 4.8. Suppose there are $n = 2$ agents. For any $w_1, w_2 \in (0, 1)$ and an allocation $A = (A_1, A_2)$, the following are equivalent:
(i) $A$ is $w$-maximal, for $w = (w_1, w_2)$,
(ii) For any exchangeable pair $o_1, o_2 \in A_1, A_2$, exactly one of the following holds:
\[ u_1(o_1) > u_1(o_2) \quad \text{and} \quad w_1/w_2 \geq r_{j1}(o_1, o_2) \quad \text{or} \]
\[ u_1(o_1) = u_1(o_2) \quad \text{and} \quad u_2(o_2) \geq u_2(o_1) \quad \text{or} \]
\[ u_1(o_1) < u_1(o_2) \quad \text{and} \quad u_1/w_2 \leq r_{j1}(o_1, o_2) \]

Proof. The only exchange-cycle in a 2-agents allocation is a replacement of an exchangeable pair $o_1, o_2 \in A_1, A_2$ between the agents. Then, according to Lemma 4.7, for any exchangeable pair $o_1, o_2 \in A_1, A_2$:
\[ w_1u_1(o_1) + w_2u_2(o_2) \geq w_1u_1(o_1') + w_2u_2(o_1') \quad (1) \]
\[ w_1u_1(o_1) - w_2u_2(o_1) \geq w_1u_1(o_2) - w_2u_2(o_2) \quad (2) \]
\[ w_1u_1(o_1) - u_2(o_1) \geq w_1u_2(o_1) - u_2(o_2) \quad (3) \]

The claim in (ii) is an algebraic manipulation of (3), so (ii) $\iff$ (3). And since (i) $\iff$ (3), also (i) $\iff$ (ii).

Lemma 4.9. For any $n$ agents, in any $w$-maximal allocation $A$ (with positive weights), for any $i, j$ and an exchangeable pair $o_i, o_j \in A_i, A_j$, the following implications hold:
\[ u_j(o_i) \geq u_j(o_j) \implies u_i(o_i) \geq u_i(o_j) \]
\[ u_i(o_i) > u_i(o_j) \implies u_i(o_i) > u_i(o_j) \]

Proof. By Lemma 4.7, since $A$ is a $w$-maximal allocation, each exchange-cycle does not increase the sum of the matching. In particular, for $x = 2$, if we define $a_1 = i, a_2 = j, o_1 = o_i, o_2 = o_j$, we have:
\[ w_1u_1(o_1) + w_1u_1(o_j) \geq w_1u_1(o_j) + w_1u_1(o_1) \]
Which is equal to:
\[ w_1[u_1(o_i) - u_1(o_j)] \geq w_1[u_j(o_i) - u_j(o_j)] \]
w_1 and $w_j$ are both positive, so if the left term is positive or non-negative, the right term must be positive or non-negative too, respectively.

Lemma 4.10. Consider a $w$-maximal allocation $A$ and an exchangeable pair $o_i, o_j \in A_i, A_j$, for some $i, j \in [n]$, $o_i$ is called a preferred item in the exchangeable pair $(o_i, o_j)$ if both $u_j(o_i) > u_j(o_j)$ and $u_i(o_i) < u_i(o_j)$.

Lemma 4.11. For any $n$ agents, in any $w$-maximal allocation $A$, if an agent $i$ envisions some agent $j$, then there is an exchangeable pair $o_i, o_j \in A_i, A_j$, and $o_i$ is the preferred item.

Proof. If $j$ envisions $i$, then $u_j(A_i) > u_j(A_j)$. Since both $A_i$ and $A_j$ contain the same number of items in each category, there must be a category in which, for some item pair $o_i, o_j \in A_i, A_j$, agent $j$ prefers $o_j$ to $o_i$. By Lemma 4.9, agent $i$ too prefers $o_i$ to $o_j$. So $o_i$ is a preferred item.

4.3.3 Maintaining the $w$-maximality

The following lemma shows that, by exchanging items, we can move from one $w$-maximal allocation to another $w'$-maximal allocation (for a possibly different weight-vector $w'$). This lemma, too, works only for two agents.

Lemma 4.12. Suppose there are $n = 2$ agents. Let $A$ be a $w$-maximal allocation, for $w = (w_1, w_2)$. Suppose there is an exchangeable pair $o_1, o_2 \in A_1, A_2$ such that:
(1) $u_2(o_1) > u_2(o_2)$, that is, $o_1$ is the preferred item.
(2) Among all exchangeable pairs in which $o_1$ is the preferred item, this pair has a largest difference-ratio $r_{j1}(o_1, o_2)$.

Let $A'$ be the allocation resulting from exchanging $o_1$ and $o_2$ in $A$. Then, $A'$ is $w'$-maximal for some $w' = (w'_1, w'_2)$ with $w'_1 \leq w_1, w'_2 \geq w_2, w'_1 \in (0, 1), w'_2 \in (0, 1)$.

Proof sketch. The lemma can be proved by using Lemmas 4.8, 4.9, the maximality condition in the lemma [condition 2] and Definition 4.6.

The idea of the proof is to define $w'_1, w'_2 \in (0, 1)$ such that $w'_i = r_{j1}(o_1, o_2), w'_1 + w'_2 = 1$. Then, $0 < \frac{w'_1}{w'_2} \leq \frac{w_1}{w_2}$, and $w'_1 \leq w_1, w'_2 \geq w_2$.

Then we look at all the exchangeable pairs $(o^*_1, o^*_2)$ in the new allocation $A'$, resulting from the exchange, and show that they satisfy all the conditions of Lemma 4.8(ii) with $w'_1, w'_2$, which are:
(a) $u_1(o^*_1) > u_1(o^*_2)$ and $r_{j1}(o_1, o_2) \geq r_{j1}(o^*_1, o^*_2)$ or
(b) $u_1(o^*_1) = u_1(o^*_2)$ and $u_2(o^*_2) \geq u_2(o^*_1)$ or
(c) $u_1(o^*_1) < u_1(o^*_2)$ and $r_{j1}(o^*_1, o^*_2) \geq r_{j1}(o^*_1, o^*_2)$

The exchangeable pairs in $A'$ can be divided into four types:
(1) The exchangeable pairs $(o^*_1, o^*_2)$ that have not moved.
(2) The pair $(o_3, o_1)$.
(3) Pairs in the form $(o^*_1, o_1), o^*_1 \in A_1, o^*_1 \neq o_2$.
Throughout this subsection we consider general mixed instances, which are defined as follows. Suppose that $\omega_{2}$ is a function of $w_{1}$, and consider the line $w_{1} + w_{2} = 1$, $w_{1} \geq 0$, $w_{2} \geq 0$, which describes the collection of all pairs of non-negative weights $w_{1,2} \in [0,1]$ whose sum is 1. Each point on this line represents a $w'$-maximal allocation for $w_{1,2}$. In each such allocation, there are no envy-cycles in the envy graph, so there is at most one envious agent.

The algorithm starts with an initial allocation which is a maximum-weight matching in the graph $G_{\omega}$, where $w = (0.5,0.5)$, corresponding to the center of the line. This initial allocation is PO (by Lemma 4.2) and EF for at least one agent. If it is EF for both agents then we are done. Otherwise, depending on the envious agent, the algorithm decides which side of the line to go to. If agent 2 envies, we need to improve 2’s weight, so we go towards (0,1). If agent 1 envies, we need to go towards (1,0). Therefore, as long as the allocation is not EF[1,1], the algorithm swaps an exchangeable pair chosen according to Lemma 4.12, thus maintaining the search space as the space of the $w$-maximal allocations. Note that since the items of the exchanged pair are both in the same category, the capacity constraints are also maintained. Lemma 4.5 implies that some point on the line gives a feasible EF[1,1] and PO division.

Specifically, the exchange pairs are determined as follows. For each item $o$ we can define a linear function $f_{o}(w_{1})$:

$$f_{o}(w_{1}) = w_{1}u_{1}(o) - w_{2}u_{2}(o) = w_{1}u_{1}(o) - (1 - w_{1})u_{2}(o) = w_{1}u_{1}(o) - u_{2}(o) + w_{1}u_{2}(o) = (u_{1}(o) + u_{2}(o))w_{1} - u_{2}(o)$$

If we draw all those functions in one coordinate system, each pair of lines intersects at most once. In total there are $O(m^{2})$ intersections, where $m = \sum_{i=1}^{n} |C_{i}|$, the total number of items, in all categories (including the dummies).

For example, consider the same-sign instance $I = (N,M,C,S,U)$ where $N = \{2\}$, $C = \{C_{1}, C_{2}\}$, $C_{1} = \{o_{1}, o_{2}, o_{3}, o_{4}\}$, $C_{2} = \{o_{5}, o_{6}\}$, $S = \{2,1\}$ and $U$ is shown in Table 2. The corresponding lines for the items are depicted in Figure 1. The meaning of each point of intersection is a possible switching point for these two items between the agents. Clearly, the replacement will only take place between exchangeable pairs, i.e. items in the same category, which are in different agents’ bundles at the time of the intersection. According to Definition 4.6, at each intersection point of the lines of $o_{1}$ and $o_{2}$, $w_{1} = u_{1}(o_{1}) - u_{1}(o_{2}) = r_{2/1}(o_{1}, o_{2})$ holds. The largest $r$ value is obtained on the right side of the graph, and as we progress to the left side its value decreases.

In this example, the algorithm starts with the allocation $A = (\{o_{1}, o_{2}\}$ in the point $(0.5,0.5)$, which is $A_{1} = \{o_{1}, o_{2}, o_{6}\}$, $A_{2} = \{o_{3}, o_{4}, o_{5}\}$. Note that for each category, 1’s items are the top lines. In this initial allocation, 2 envies by more than one item, so we start exchanging items in order to increase $w_{2}$. The first interesting pair (when we go left) is $o_{5}, o_{6}$. It is an exchangeable pair, so we exchange it and update the allocation to $A_{1} = \{o_{1}, o_{2}, o_{5}\}$, $A_{2} = \{o_{3}, o_{4}, o_{6}\}$. This is an EF1 allocation, so we are done.

If at some point there are multiple intersections of exchangeable pairs, we swap the pairs in an arbitrary order.

**Lemma 4.13.** If Algorithm 1 exchanges the last exchangeable pair in the item-pairs list (that is initialized in step 8), then the resulting allocation is envy-free for agent 2.

**Proof.** After the last exchange, there is no exchangeable pair $(o_{1}, o_{2})$, $o_{1}, o_{2} \in A_{1, A_{2}}$ for which $o_{1}$ is the preferred item. Therefore, by Lemma 4.11, agent 2 is not jealous. □

**Theorem 4.14.** Algorithm 1 always returns an allocation that is $w$-maximal with positive weights (and thus PO), and satisfies the capacity constraints. The allocation is EF[1,1], and EF1 for a same-sign instance.

**Proof.** A matching in $G_{\omega}$ graph always gives each agent $s_{i}$ items of category $C_{i}$. Thanks to the dummy items, all possible allocations that satisfy the capacity constraints can be obtained by a matching. The first allocation that the algorithm checks is some $w$-maximal allocation, where $w = (w_{1}, w_{2}, w_{3}, w_{4}) \in (0,1)$, so by Proposition 4.2, this is a PO allocation. At each iteration, it exchanges an exchangeable pair, $(o_{1}, o_{2})$, such that $w_{2}(o_{1}) > w_{2}(o_{2})$, and among all the exchangeable pairs with $w_{2}(o_{1}) > w_{2}(o_{2})$ it has the largest $r_{2/1}(o_{1}, o_{2})$, so by Lemma 4.12, the resulting allocation...
Algorithm 1 Finding an EF[1,1] and PO division for two agents

```
// Step 1: Find a w'-maximal feasible allocation that is EF for some agent.
1: A = (A_1, A_2) ← a w'-maximal allocation, for w_1 = w_2 = 0.5.
2: if A is EF[1,1] then
3: return A
4: end if
5: if A is EF for agent 2 then
6: replace the names of agent 1 and agent 2
7: end if
8: // We can now assume that agent 2 is jealous.
9: // Step 2: Build a set of item-pairs whose replacement increases agent 2’s utility:
10: item-pairs ← all the exchangeable pairs o_1, o_2 ∈ A_1, A_2, for which u_2(o_1) > u_2(o_2).
11: current-pair ← (o_1, o_2) where r_2/1(o_1, o_2) is maximal.
12: // Step 3: Switch items in order until an EF[1,1] allocation is found:
13: while A = (A_1, A_2) is not EF[1,1] do
14: Switch current-pair between the agents.
15: Update item-pairs list and current-pair (Steps 8, 9).
16: end while
17: return A
```

is also w'-maximal for some w' = (w'_1, w'_2), w'_1, w'_2 ≥ 0. In addition, since the items are in the same category, the allocation remains feasible. The first allocation in the sequence is, by step 1, envy-free for agent 1. By Lemma 4.13, the last allocation in the sequence is envy-free for agent 2. So by Lemma 4.5, there exists some iteration in which the allocation is PO and EF[1,1], and EF1 for a same-sign instance.

Theorem 4.15. The runtime of Algorithm 1 is O(m^4).

Proof. Step 1 can be done by finding a maximum weighted matching in a bipartite graph G. Its time complexity is O(|V|^3) (Fredman and Tarjan [22]), where |V| = 2m, the number of vertices in the graph. Thus, O(m^3) is the time complexity of step 1.

At step 2 we go through all the categories c ∈ [k], at each we create groups A_1c, A_2c which contain agent 1’s and agent 2’s items from C_c in A. It can be done in 2^k |C_c| = msc. Now we have |A_1c| = |A_2c| = s_c. Then, we iterate over all the pairs o_1, o_2 ∈ A_1c, A_2c, and add them to the list, which takes s_c^2 time. In total, building item-pairs list is \( \sum_{c \in [k]} (ms_c + s_c^2) = O(\sum_{c \in [k]} ms_c) = O(km^2) \).

The item-pairs list size is \( \sum_{c \in [k]} s_c^2 = O(m^2) \), and then finding its maximum takes O(m^2). In total, step 2 takes O(km^2) time.

The upper bound on the number of iterations in the while loop at step 3 is the number of intersection points between items, which is at most O(m^2). At each iteration we switch one exchangeable pair, (o_1, o_2), and update the pairs-list. The only pairs that should be updated (deleted or added) are those that contain o_1 or o_2. There are at most 2m = O(m) such pairs. Finding the maximum is O(m^2).

In total, step 3 takes O(m^4) time.

Overall, the time complexity of the algorithm is O(m^4) (because m ≥ k necessarily).

5 Conclusion and Future Work

We presented the first algorithm for efficient nearly-fair allocation of mixed goods and chores with capacity constraints. We believe that our paper provides a good first step in understanding fair division of mixed resources under cardinality constraints. Our proofs are modular, and some of our lemmas can be used in more general settings.

5.1 Three or More Agents

The most interesting challenge is to generalize our algorithm to three or more agents. Proposition 4.2 and Lemmas 4.4, 4.7, 4.9, 4.11 work for any number of agents, but the other lemmas currently work only for two agents.

Algorithm 1 essentially scans the space of w-maximal allocations: it starts with one w-maximal allocation, and then moves in the direction that increases the utility of the envious agent. To extend it to n agents, we can similarly start with a w-maximal allocation corresponding to w = (1/n, . . . , 1/n), i.e., identical weights for each of the agents. These weights represent a point in an n-dimensional space. Then, we can exchange items to benefit an envious agent, in order to increase their weight and improve their utility. In case there are several envious agents, we can select one that is at the “bottom” of the envy chain. For example, in the SWAP algorithm of Biswas and Barman [10], the swap is done in a way that benefits the envious agent with the smallest utility. Similarly, in the envy-graph algorithm of Lipton et al. [29], the next item is given to an agent with no incoming edges in the envy-graph (an agent who is not envied by any other agent). The exchanges should be done in an order that preserves the w-maximality and ensures we reach an EF[1,1] allocation. The two main Lemmas that should be extended to ensure the above two conditions are Lemma 4.12 and Lemma 4.5. We have not yet been able to develop such a method and prove its correctness. Finding an EF1+PO allocation for n = 3 agents seems hard even when there is a single category with only goods.

5.2 More General Constraints

Another possible generalization is to more general constraints. Capacity constraints are a special case of matroid constraints, by which each bundle should be an independent set of a given matroid (see [10] for the definitions). Lemmas 4.2, 4.4, 4.5, 4.9 and 4.12 do not use categories, and should work for general matroids. The other lemmas should be adapted.

Finally, we assumed that both agents have the same capacity constraints. We do not know if our results can be extended to agents with different capacity constraints (e.g. agent 1 can get at most 7 items while agent 2 can get at most 3 items). Specifically, the proof of Lemma 4.4 does not work — if (A_1, A_2) is feasible, then (A_2, A_1) might be infeasible.
