PROPm Allocations of Indivisible Goods to Multiple Agents

Artem Baklanov[1] , **Pranav Garimidi**[2] , **Vasilis Gkatzelis**[3] and **Daniel Schoepflin**[3]
[1]HSE University, Russian Federation
[2]Columbia University
[3]Drexel University
apbaklanov@hse.ru, pg2682@columbia.edu, {gkatz,schoep}@drexel.edu

Abstract

We study the classic problem of fairly allocating a set of indivisible goods among a group of agents, and focus on the notion of approximate proportionality known as PROPm. Prior work showed that there exists an allocation that satisfies this notion of fairness for instances involving up to five agents, but fell short of proving that this is true in general. We extend this result to show that a PROPm allocation is guaranteed to exist for all instances, independent of the number of agents or goods. Our proof is constructive, providing an algorithm that computes such an allocation and, unlike prior work, the running time of this algorithm is polynomial in both the number of agents and the number of goods.

1 Introduction

The fair allocation of scarce resources to a group of competing agents is a fundamental problem in both computer science and economics. A particularly natural and well-studied setting is the fair allocation of indivisible goods to agents with additive valuations. Under additive valuations, an agent i has a value v_{ij} for each good j and her value for a bundle of goods S is equal to the sum of the values over each good $j \in S$, i.e., $v_i(S) = \sum_{j \in S} v_{ij}$. An indivisible good cannot be split and shared by more than one agent so achieving "fairness" with indivisible goods is often a difficult task. Even determining the appropriate definition of fairness can be non-trivial.

One standard notion of fairness is *proportionality*. An allocation of a set of goods M to n agents is proportional if each agent i receives a set of goods S_i for which she has value $v_i(S_i) \geq \frac{1}{n} v_i(M)$. In words, proportionality requires that every agent obtains at least a $1/n$ fraction of her total value. Unfortunately, when items are indivisible achieving proportionality may not be possible. For instance, when allocating a single indivisible good there is no way to provide *any* positive value to anyone other than the one agent that receives the good. In fact, this example shows that one cannot even guarantee any *multiplicative* approximation of proportionality. On the other hand, this instance does not rule out the existence of allocations satisfying *additive* relaxations of proportionality.

Three notable additive relaxations of proportionality are PROP1, PROPx, and PROPm. Each of these notions requires that agent i must receive value no less than $\frac{1}{n} v_i(M) - d_i$ for some appropriately defined $d_i \geq 0$. The least demanding of these notions is PROP1, wherein d_i is the *largest* value that agent i has for any item allocated to another agent [Conitzer *et al.*, 2017]. On the other extreme, for PROPx d_i is the *smallest* value that agent i has for any item allocated to another agent [Moulin, 2019]. PROP1 is known to be easy to satisfy and provides weak guarantees, while PROPx is overly demanding and known to not always exist. In the case of PROPm, d_i corresponds to the *maximin* value that agent i has among items allocated to other agents [Baklanov *et al.*, 2020]. PROPm sits somewhere between PROP1 and PROPx, and it is the focus of this work.

Baklanov *et al.* [2020] demonstrated that there always exists a PROPm allocation for problem instances with up to five agents. They also demonstrate that many other alternative relaxations of proportionality (e.g., letting d_i be the value of the minimax value item, the median value item, and the average value item) fail to exist even for instances of three agents. PROPm then seems to be a rather unique notion of approximate proportionality in that it strikes a balance between providing non-trivial guarantees and seemingly being plausible to exist in general cases. However, the techniques used to prove this existence result required extensive case analysis, suggesting that they would not be useful toward an analogous proof for instances with many agents. Two natural questions then arise from [Baklanov *et al.*, 2020]: Are PROPm allocations always guaranteed to exist for any number of agents? If so, can they be efficiently computed? In this work, we answer both these questions in the affirmative.

2 Related Work

As discussed above, it is impossible to guarantee any multiplicative approximation of proportionality in the indivisible items setting. The first additive approximation, "proportionality up to the most valued item" (PROP1), was originally proposed by Conitzer *et al.* [2017] where the authors demonstrated that there always exists a Pareto optimal allocation that is also PROP1. On the other hand, Moulin [2019] showed that if we instead consider "proportionality up to the least valued item" (PROPx) we can no longer guarantee existence. Moreover, Aziz *et al.* [2020] demonstrated that PROPx allocations may not exist even for instances with only three agents.

Another standard notion of fairness is "envy-freeness"

wherein an agent is said to be envy-free if she has weakly higher value for the set of goods she receives than the set of goods any other agent receives. The instance with the single indivisible items, discussed above, verifies that envy-freeness may not be achievable either, so prior work has focused on notions of approximate envy-freeness, namely "envy-freeness up to the most valued item" (EF1) [Budish, 2011] and "envy-freeness up to the least valued item" (EFx) [Caragiannis *et al.*, 2019]. Similar to PROP1, EF1 allocations are known to exist for any number of agents [Lipton *et al.*, 2004]. On the other hand, the existence or non-existence of EFx allocations has not been proven in general, and it is one of the main open problems in fair division.

Plaut and Roughgarden [2018] demonstrated that EFx allocations always exist for two agents (even with combinatorial valuations) and Chaudhury *et al.* [2020a] established the existence of EFx allocations for instances with three agents with additive valuations. Extending the results in [Chaudhury *et al.*, 2020a] to more than three agents remains a challenging problem as the proof relies on complex case analysis, much like the proof of existence of PROPm allocations for up to five agents in [Baklanov *et al.*, 2020]. Central to many of the proofs of existence for EF1 and EFx is a variation of a procedure of Lipton *et al.* [2004] known as "envy-cycle elimination" (see, e.g., [Plaut and Roughgarden, 2018; Chaudhury *et al.*, 2020b; Oh *et al.*, 2019; Amanatidis *et al.*, 2020]) wherein a graph representing a given allocation is constructed and an alternative allocation is produced by propagating changes along the edges of the graph. Our algorithm for generating PROPm allocations has a very similar flavor, beginning from a partial allocation and using a graph analysis to imply a set of changes sufficient to arrive at a PROPm allocation.

Even if some fairness notion is shown to be achievable, it is still crucial to study the computational tractability of finding a solution that satisfies it. Aziz *et al.* [2020] provided a strongly polynomial-time algorithm producing a PROP1 and Pareto efficient allocation even in the presence of chores (i.e., some goods can have negative value). For EF1 allocations, Caragiannis *et al.* [2019] showed that maximizing the Nash social welfare (the geometric mean of the values of the agents) produces an allocation that is EF1 and Pareto efficient. On the other hand, Lee [2017] demonstrated that computing this is intractable. However, the work of Barman *et al.* [2018] provided an alternative pseudo-polynomial time algorithm that computes an EF1 and Pareto optimal allocation. For EFx, the picture is much less clear. The algorithmic result in [Plaut and Roughgarden, 2018] relies on computing the allocation optimizing the leximin objective which may take exponential time and the result for three agents in [Chaudhury *et al.*, 2020a] leads only to a pseudo-polynomial time algorithm. For PROPm, the existing results in [Baklanov *et al.*, 2020] are constructive but may require exponential time in the number of items, even for just five agents. On the other hand, in this work we demonstrate that PROPm allocations for any number of agents can, indeed, be computed in time polynomial in the number of agents and items – a major improvement over [Baklanov *et al.*, 2020].

3 Our Results

Prior to this work, we knew that an allocation satisfying PROPm always exists for instances involving up to five agents. In this paper, we significantly extend this result by providing an algorithm that computes a PROPm allocation for any number of goods and agents. Moreover, our algorithm operates in time polynomial in both the number of agents and items, unlike the algorithm proposed in [Baklanov *et al.*, 2020], which was not polynomial even for a fixed number of agents. In light of these results, PROPm stands out as a rare example of a quite non-trivial fairness notion for which we get universal existence and polynomial-time computability.

Our algorithm employs a useful observation from [Baklanov *et al.*, 2020] (see Observation 3 in Subsection 6.2 in this paper) which characterizes the conditions under which an instance can be split into agent- and item-disjoint subproblems which can, effectively, be solved completely separately, yielding a full solution for the initial instance. To produce such sub-problems, we consider a novel graph representation of our instance and search for paths through the graph. These paths imply a series of gradual modifications leading to the final decomposition of each problem instance into sub-problems. We consider this algorithm to be of both practical and theoretical interest.

4 Preliminaries

We study the problem of allocating a set M of m indivisible items (or goods) to a set of n agents $N = \{1, 2, \ldots, n\}$. Each agent i has a value $v_{ij} \geq 0$ for each good j and her value for receiving some subset of goods $S \subseteq M$ is additive, i.e., $v_i(S) = \sum_{j \in S} v_{ij}$. For ease of presentation, we normalize the valuations so that $v_i(M) = 1$ for all $i \in N$. We also assume that $v_{ij} \leq 1/n$ for all $i \in N, j \in M$, because any item j with $v_{ij} > 1/n$ could be assigned to i and reduce the problem to finding a PROPm allocation of $M \backslash \{j\}$ to $N \backslash \{i\}$.[1] We let $m_i(S) = \min_{j \in S}\{v_{ij}\}$ denote the value of the least valuable good for agent i in bundle of goods S.

An allocation $X = (X_1, X_2, \ldots, X_n)$ is a partition of the goods into bundles such that X_i is the bundle allocated to agent i. We use $d_i(X) = \max_{i' \neq i}\{m_i(X_{i'})\}$ to denote the value of the *maximin good of agent i in X*, and we say that an agent i is **PROPm-satisfied** by X if $v_i(X_i) + d_i(X) \geq 1/n$. An allocation X is PROPm if it PROPm-satisfies every agent.

The goal of our algorithm is to use these bundles to decompose the problem into smaller sub-problems, and compute a PROPm allocation using a divide & conquer approach. A **sub-problem** $(\mathcal{A}, N')$ is a pair consisting of a set of bundles $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ and a subset of agents $N' \subseteq N$. In other words, a sub-problem "matches" a group of agents with a group of bundles, and our goal is going to be to do so in a way that computing a PROPm allocation for each sub-problem yields a PROPm allocation for the original problem. The value of an agent i for a set of bundles $\mathcal{A}$ is $v_i(\mathcal{A}) = \sum_{A_j \in \mathcal{A}} v_i(A_j)$. We call a sub-problem $(\mathcal{A}, N')$ **proportional** if $v_i(\mathcal{A})/|N'| \geq 1/n$ for all $i \in N'$.

[1] This fact is proven as Lemma 2 in [Baklanov *et al.*, 2020].

Given a set of bundles $\mathcal{A}$ and a set of agents N', a **decomposition** is a division of these agents and bundles into (bundle and agent) disjoint sub-problems. For example consider a set of five agents $N' = \{1, 2, 3, 4, 5\}$ and a set of five bundles $\mathcal{A} = \{A_1, A_2, A_3, A_4, A_5\}$. One possible decomposition of (A, N') would be into the disjoint sub-problems $(\{A_1, A_2, A_3\}, \{1, 2, 3\})$ and $(\{A_4, A_5\}, \{4, 5\})$. We say that a decomposition for $(\mathcal{A}, N')$ is **proportional** if all of its included sub-problems are proportional. As we show later on, as long as a decomposition is proportional, we can focus on solving each of its sub-problems recursively without worrying about the allocation beyond that sub-problem.

Consider, again, the example above of five agents $\{1, 2, 3, 4, 5\}$ and five bundles $\{A_1, ..., A_5\}$. For these five agents assume agents 1, 2, and 3 have the same valuation function and assume agents 4 and 5 have the same valuation functions. Let the valuation functions for agents 1, 2, and 3 be $v(A_1) = \frac{1}{4}$ and $v(A_2) = v(A_3) = v(A_4) = v(A_5) = \frac{3}{16}$ and let the valuation function for agents 4 and 5 be $v(A_1) = v(A_2) = v(A_3) = \frac{1}{6}$, $v(A_4) = \frac{3}{20}$, and $v(A_5) = \frac{7}{20}$. Since valuation functions are additive, we then have that $v_i(A_1 \cup A_2 \cup A_3)/3 \geq 1/5$ for $i \in \{1, 2, 3\}$ and $v_i(A_4 \cup A_5)/2 \geq 1/5$ for $i \in \{4, 5\}$. Thus, the decomposition of (A, N') described above $D = (\{A_1, A_2, A_3\}, \{1, 2, 3\}), (\{A_4, A_5\}, \{4, 5\}))$ is a proportional decomposition. We will see that this means that we can solve the two sub-problems of allocating items in $A_1 \cup A_2 \cup A_3$ to agents 1, 2, and 3 such that they are PROPm-satisfied with respect to $A_1 \cup A_2 \cup A_3$ and allocating items in $A_4 \cup A_5$ to agents 4, 5 such that they are PROPm-satisfied with respect to $A_4 \cup A_5$ to produce an allocation where every agent is PROPm-satisfied in the original problem.

5 PROPm Algorithm

Our algorithm begins by choosing some arbitrary agent $i \in N$ to serve as the "divider" (we henceforth use i to refer to the divider agent and $N^{-i} = N \setminus \{i\}$ to refer to the set of all other agents). The divider agent partitions the items into n bundles, and then the algorithm proceeds to evaluate the other agents' preferences over these bundles to decide which one the divider should receive. Once the divider's bundle has been determined, the initial problem is decomposed into smaller sub-problems that are solved recursively.

5.1 Stage 1: The Divider Partitions the Goods

In order to partition the goods, the divider (agent i) first sorts them in non-decreasing order of value, from i's perspective, and indexes them accordingly. Then, the first bundle S_1 corresponds to the longest prefix of goods in this ordering such that $v_i(S_1) \leq 1/n$. Observe that, by construction, $v_i(S_1) + v_{ij} > 1/n$ for all $j \in M \setminus S_1$. Moreover, there is at least $(n-1)/n$ total value remaining for i outside S_1. We construct S_2 by taking the longest prefix of goods in $M \setminus S_1$ such that i has value less than or equal to $1/(n-1) \cdot v_i(M \setminus S_1)$ for receiving all of them. Similarly, we let S_k be the longest prefix of goods in $M \setminus (\cup_{j=1}^{k-1} S_j)$ such that the divider's value for these items remains less than or equal to $1/(n-k+1) \cdot v_i(M \setminus (\cup_{j=1}^{k-1} S_j))$.

5.2 Stage 2: Decomposing into Sub-problems

Using disjoint bundles $S_1, S_2, \ldots, S_n$ from the divider's partition, we now decompose the problem into sub-problems, eventually solving them recursively. Specifically, we carefully choose one of these n bundles, say S_t, and allocate it to the divider. We then recursively allocate the items of bundles $S_1, \ldots, S_{t-1}$ to some group N_L of $t-1$ agents, and the items of bundles $S_{t+1}, \ldots, S_n$ to some group N_R of $n-t$ agents.

As the pseudocode of Algorithm 1 shows, the decomposition process works in a sequence of (up to) n iterations, indexed by $t \in \{1, 2, \ldots, n\}$. At the beginning of every iteration t, the algorithm has already identified a proportional decomposition D involving $t-1$ agents and the bundles $S_1, S_2, \ldots, S_{t-1}$. At the end of step t, either the proportional decomposition D has been updated to also include the bundle S_t and a total of t agents (possibly different than the $t-1$ ones that were participating in it at the beginning of the round), or the bundle S_t has been assigned to the divider agent, and the remaining problem has been decomposed into a list of proportional sub-problems. Throughout the execution of the algorithm, N_R is used to denote the set of agents that are not participating in the proportional decomposition D.

The first thing that the algorithm does in each iteration t is to evaluate c, the number of agents from N_R whose average value for the first t bundles is more than $1/n$. If c is equal to 0, this means that all the agents in N_R essentially "prefer" sharing the last $n-t$ bundles instead of the first t bundles. If this is the case, then the algorithm allocates bundle S_t to the divider agent. It then recursively solves the proportional decomposition D, whose sub-problems involve $t-1$ agents and the first $t-1$ bundles, and also recursively solves the sub-problem involving the remaining $n-t$ agents (i.e., those in N_R) and the items from the last $n-t$ bundles, S_{t+1} to S_n.

On the other hand, if the value of c is positive, this suggests that there are agents in N_R that "prefer" to share the first t bundles rather than the last $n-t$ bundles. Intuitively, this suggests that the first t bundles are "over-demanded", so our algorithm calls UPDATEDECOMPOSITION, a crucial subroutine, to update decomposition D. As we discuss in subsection 5.3, a single execution of this subroutine can have one of two possible outcomes: i) either the value of c decreases by 1, or ii) the number of agents in the decomposition (denoted $|D.agents|$ for notational simplicity) increases by 1. The algorithm keeps calling this subroutine until either the decomposition grows to include bundle S_t and t agents, or c drops to 0. In the former case, it continues to the next iteration (i.e., $t \leftarrow t+1$), otherwise, it assigns S_t to the divider and recursively solves the remaining sub-problems. Figure 1 shows an example of the algorithm running on a sample instance.

5.3 The UPDATEDECOMPOSITION Subroutine

The UPDATEDECOMPOSITION subroutine plays a central role in our PROPm algorithm, and it achieves the desired update of the existing proportional decomposition D at iteration t by propagating changes on a carefully constructed graph. Note that whenever we call this subroutine, the value of c is positive, so there exists at least one agent $k \in N_R$, i.e., not participating in D, for whom $v_k(S_1 \cup \cdots \cup S_t)/t > 1/n$.

Algorithm 1: PROPm Algorithm

```
Let S_1, S_2, ..., S_n be the bundles the divider produces
Let D be an, initially empty, decomposition
N_R ← N^{-i}
for t = 1 to n do
    c ← |{k ∈ N_R : v_k(S_1 ∪ ··· ∪ S_t)/t > 1/n}|
    while c > 0 and |D.agents| < t do
        D ← UPDATEDECOMPOSITION
        N_R ← subset of N^{-i} not participating in D
        c ← |{k ∈ N_R : v_k(S_1 ∪ ··· ∪ S_t)/t > 1/n}|
    if |D.agents| < t then
        Allocate S_t to the divider agent (agent i)
        Recursively solve all sub-problems of D
        Recursively solve (S_{t+1} ∪ ··· ∪ S_n, N_R)
        Return the combined allocation
```

Given the decomposition of disjoint sub-problems D, we construct a directed "sub-problem graph" $G = (V, E)$, where each vertex in V corresponds to a sub-problem in D and and an edge (u, w) between two vertices $u, w \in V$ exists if and only if the corresponding sub-problems, $(\mathcal{A}_u, N_u)$ and $(\mathcal{A}_w, N_w)$ satisfy the following condition: there exists some agent $k \in N_u$ who satisfies $\frac{v_k(\mathcal{A}_w)}{|N_w|} \geq \frac{1}{n}$. In other words, such an edge exists if and only if removing some agent from N_w and replacing her with agent k would maintain the proportionality of the $(\mathcal{A}_w, N_w)$ sub-problem.

The sub-problems corresponding to the vertices of G involve bundles $S_1, \ldots, S_{t-1}$ and some set of $t-1$ agents so the number of vertices in G is at most $t-1$. We add to G two more vertices. The first vertex, w_α, corresponds to the agent $k \in N_R$ mentioned above; this vertex has outgoing edges to all the sub-problems $(\mathcal{A}, N')$ of D for which $\frac{v_k(\mathcal{A})}{|N'|} \geq \frac{1}{n}$. The second vertex, w_β, corresponds to the bundle S_t that we wish to introduce to this decomposition. This vertex has incoming edges from any vertex whose sub-problem includes an agent i' with value $v_{i'}(S_t) \geq 1/n$. In this graph, let R be the set of vertices that are reachable from w_α via directed paths.

Case 1. If this set R includes the vertex w_β, corresponding to the bundle S_t, i.e., if there is a path from w_α to w_β, then the subroutine reallocates agents along the sub-problems of this path. Specifically, for each edge (u, w) on this path, we remove from the sub-problem of u the agent that is responsible for the existence of this edge (we choose one arbitrarily if there are multiple) and we place that agent in the sub-problem of w. As a result, D would then include bundle S_t as well as agent k, thus increasing $|D.agents|$ and, as we argue in Section 6, this modification maintains the proportionality of the decomposition.

Case 2. If the set R does not include the vertex w_β, but it includes some agent i' with $\frac{v_{i'}(S_1 \cup \cdots \cup S_t)}{t} \leq \frac{1}{n}$, then we perform an analogous shift of the agents across the sub-problems along the path from k to i', but remove agent i' from the decomposition and add her to the set N_R. This, again, does not compromise the proportionality of the decomposition, but it ensures that the updated value of c will drop by 1 since agent k was removed from N_R and replaced with agent i' who does not contribute toward an increase of the value of c.

Case 3. Finally, if neither of the cases above holds, the subroutine takes all the agents and all the bundles corresponding to vertices in R and merges them into a single sub-problem, together with agent k and bundle S_t. This, again, increases $|D.agents|$ and as we show using a separate argument in Section 6, it maintains the proportionality of the decomposition.

6 Correctness of the Algorithm

To verify the correctness of the algorithm, we first show that it always terminates and returns an allocation (in fact, we demonstrate in Section 7 that the algorithm completes in polynomial time). As we verify in subsection 6.1, a single call to the UPDATEDECOMPOSITION subroutine returns an updated decomposition that has either one additional agent (and bundle) or has reduced the value of c by 1. Since the value of c at the beginning of each iteration can never be more than $n-1$, this ensures that the while loop will always terminate within a finite number of iterations. If for some iteration $t \leq n-1$ the value of $|D.agents|$ drops below t, then the algorithm recurses on smaller problems and returns the induced allocation. If, on the other hand, $|D.agents|$ does not drop below t for any iteration $t \leq n-1$, then when $t = n$ we have an empty set N_R, necessarily leading to D including every agent except for the divider (meaning $|D.agents| = n - 1 < t$). Also, note that the size of the sub-problems solved recursively always strictly decreases. We demonstrate in subsection 6.2 that this process yields a proportional allocation for all agents.

6.1 Correctness of UPDATEDECOMPOSITION

We now formally prove that after every execution of the UPDATEDECOMPOSITION subroutine, either $|D.agents|$ increases by 1, or the value of c drops by 1. In both cases, the resulting decomposition remains proportional, throughout the execution of the algorithm.

First, note that the set R of vertices that are reachable from agent k is bound to be non-empty. This is due to the fact that $v_k(S_1 \cup \cdots \cup S_t)/t > 1/n$, i.e., the agent's average bundle value is more than $1/n$ and, by pigeonhole principle, there must exist some sub-problem $(\mathcal{A}, N')$ such that $v_k(\mathcal{A})/|\mathcal{A}| \geq 1/n$. Since the set R is not empty, the description of the subroutine in the previous section clearly shows that it always achieves either an increase of $|D.agents|$ or a drop of the value of c. Therefore, the rest of this subsection focuses on proving that all of these updates on the decomposition maintain its proportionality.

Lemma 1. *Given a proportional decomposition, the described re-allocation of agents across the directed edges of a path in the decomposition's sub-problem graph leads to a new decomposition that remains proportional.*

Proof. Note that, by definition of the sub-problem graph, any agent that caused the existence of an edge (u, w) must have a value of at least $|N_w|/n$ for the bundles of the sub-problem corresponding to vertex w. As a result that agent will still satisfy proportionality if moved from u to w. This is true for

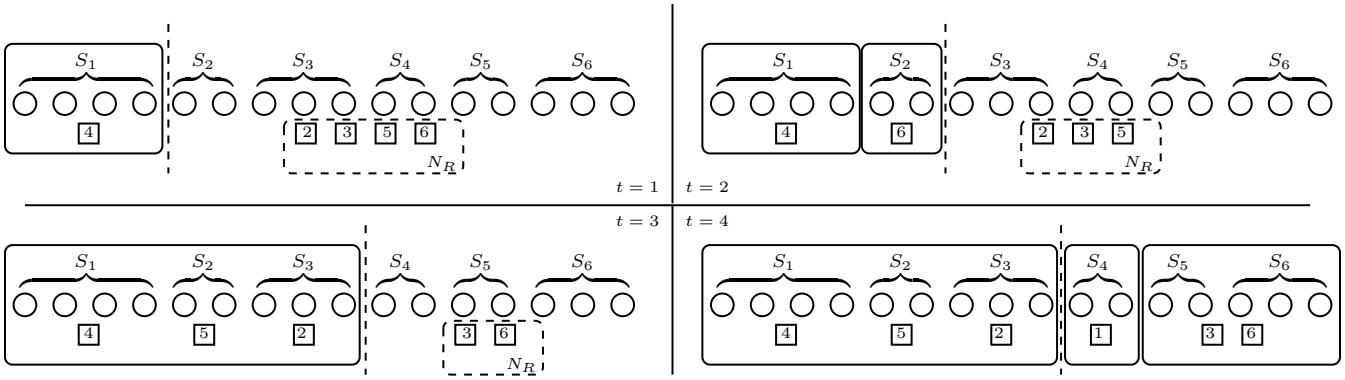

Figure 1: The state of our algorithm after the completion of iterations $t \in \{1, 2, 3, 4\}$ on a sample instance with six agents and agent 1 as the divider. The circles correspond to items, which are grouped together into bundles $\{S_1, \ldots, S_6\}$ by the divider. The numbered boxes beneath the items correspond to agents 2 through 6, and each of the larger rounded rectangles, containing bundles and agents, correspond to sub-problems. In each iteration of the algorithm, captured by this figure, the dashed vertical line separates the bundles of the decomposition D, on the left, from the remaining bundles. After each iteration t, either the decomposition is updated to include bundle S_t (as we see for $t \in \{1, 2, 3\}$), or bundle S_t is allocated to the divider agent (as wee see for $t = 4$), finalizing the set of sub-problems to be solved recursively.

all the edges on this graph (including the ones connecting the two added vertices w_α and w_β), ensuring the proportionality is maintained. □

The more demanding case is to verify that proportionality is also maintained by the third type of update that this subroutine performs, i.e., the creation of a sub-problem involving the agents and goods in R as well as agent k and bundle S_t.

Lemma 2. *If $\mathcal{A}'$ is the collection of all the bundles corresponding to sub-problems not reachable from w_α, excluding S_t, then for every agent q from a sub-problem in R we have $v_q(\mathcal{A}')/|\mathcal{A}'| < 1/n$. The same is true for the agent k that corresponds to vertex w_α, i.e., $v_k(\mathcal{A}')/|\mathcal{A}'| < 1/n$.*

Proof. Assume that this is not the case, i.e., that either some agent q corresponding to a sub-problem in R, or agent k, has an average bundle value at least $1/n$ for the bundles in $\mathcal{A}'$. By the pigeonhole principle, this implies that there must exist some sub-problem not reachable from R such that this agent's average value for the bundles in that sub-problem is at least $1/n$. But, based on the definition of the sub-problem graph, this would imply the existence of an edge from that agent's vertex to this sub-problem's vertex, contradicting the fact that the latter is not reachable from the former. □

Whenever the subroutine resorts to the third type of decomposition update (Case 3), this means that there is no agent i' in a sub-problem in R such that $\frac{v_{i'}(S_1 \cup \cdots \cup S_t)}{t} \leq \frac{1}{n}$ (Case 2). Thus, agent k and all agents q from sub-problems in R have value for the items in $S_1 \cup \cdots \cup S_t$ greater than t/n. Lemma 2 implies that their total value for the bundles in $\mathcal{A}'$ is less than $|\mathcal{A}'|/n$. Therefore, these agents' value for the items contained in sub-problems in R is at least $(t - |\mathcal{A}'|)/n$.

Also, note that the overall number of agents in the sub-problems of G (excluding k) is $t-1$, and the number of agents in the sub-problems not reachable from R are $|\mathcal{A}'|$, so the total number of agents in the sub-problems of R is $t - |\mathcal{A}'| - 1$. Therefore, if we define a new sub-problem using the agents from R, combined with agent k corresponding to vertex w_α, and the bundles from R, combined with S_t, then this sub-problem will be proportional, because every agent's value for the bundles in it will be at least $(t - |\mathcal{A}'|)/n$ and the number of agents in it is $t - |\mathcal{A}'|$.

6.2 All Agents Are PROPm-satisfied

To verify that the induced allocation is always PROPm, we first restate a useful observation from [Baklanov *et al.*, 2020]. This observation provides us with a sufficient condition under which "locally" satisfying PROPm in each sub-problem yields a "globally" PROPm allocation. Given an allocation of a subset of items to a subset of agents, we say that this partial allocation is PROPm if the agents involved would be PROPm-satisfied if no other agents or items were present.

Observation 3. *Let N_1, N_2 be two disjoint sets of agents, let M_1 and $M_2 = M \setminus M_1$ be a partition of the items into two sets, and let X be an allocation of the items in M_1 to agents in N_1 and items in M_2 to agents in N_2. Then, if some agent $i \in N_1$ is PROPm-satisfied with respect to the partial allocation of the items in M_1 to the agents in N_1, and $\frac{v_i(M_1)}{|N_1|} \geq \frac{1}{|N_1+N_2|}$, then i is PROPm-satisfied by X regardless of how the items in M_2 are allocated to agents in N_2.*

We now prove a result regarding the partition implied by the divider agent's preferences, which is analogous to a theorem that is shown by [Baklanov *et al.*, 2020] for a different, much more complicated, partition of the items.

Theorem 4. *If the divider agent receives any bundle S_ℓ and no item from $S_1 \cup S_2 \cup \cdots \cup S_{\ell-1}$ is allocated to the same agent as an item from $S_{\ell+1} \cup S_{\ell+2} \cup \cdots \cup S_n$, then agent i will be PROPm-satisfied.*

Proof. For all $k \in [n]$, we have $v_i(S_k) \leq \frac{1}{(n+1-k)} v_i(M \setminus (S_1 \cup S_2 \cup \cdots \cup S_{k-1}))$ by definition of S_k. Applying this upper bound on $v_i(S_k)$ for $k = 1$, because $v_i(M) = 1$ we have that $v_i(M \setminus S_1) \geq 1 - \frac{1}{n} = \frac{n-1}{n}$. By applying the upper bound on $v_i(S_k)$ for $k = 2$ and our lower bound on $v_i(M \setminus S_1)$ we get $v_i(M \setminus (S_1 \cup S_2)) \geq \frac{n-1}{n} - \frac{1}{n-1} \cdot \frac{n-1}{n} \geq \frac{n-2}{n}$. Iteratively

repeating this process, we obtain that for all $k \in [n]$ we know that $v_i(M \setminus (S_1 \cup S_2 \cup \cdots \cup S_k)) \geq \frac{n-k}{n}$. Also by definition, we have that $v_i(S_\ell) + \min_{j \in M \setminus (S_1 \cup S_2 \cup \cdots \cup S_\ell)}\{v_{ij}\} \geq \frac{1}{(n+1-\ell)} \cdot v_i(M \setminus (S_1 \cup S_2 \cup \cdots \cup S_{\ell-1})) \geq \frac{1}{n+1-\ell} \cdot \frac{n-(\ell-1)}{n} = \frac{1}{n}$. But finally, as long as the items from $S_1 \cup S_2 \cup \cdots \cup S_{\ell-1}$ are not included in any of the bundles containing the items in $M \setminus (S_1 \cup S_2 \cup \cdots \cup S_\ell)$ in the complete allocation X, we have that $d_i(X) \geq \min_{j \in M \setminus (S_1 \cup S_2 \cup \cdots \cup S_\ell)}\{v_{ij}\}$ so i is PROPm-satisfied when allocated set S_ℓ. □

Lemma 5. *The divider agent is always PROPm-satisfied. All non-divider agents are always PROPm-satisfied as well.*

Proof. Note that the divider agent always receives a bundle S_t in some iteration t. All the items from bundles $S_1, \ldots, S_{t-1}$ are allocated to the agents that were in the decomposition D at that time, while all the items from bundles $S_{t+1}, \ldots, S_n$ are allocated to the agents that were in N_R at the time (and hence not in D). Then, given Theorem 4, we conclude that the divider agent is always PROPm-satisfied.

Now observe that no agent other than the divider agent is directly allocated a bundle by our algorithm. Instead, the allocation to the other agents is decided recursively in some recursive call of a smaller sub-problem, when they are assigned the role of the divider. The important thing to verify is that PROPm-satisfying these agents in a recursive call, based on a subset of the agents and a subset of the goods, does, in fact, imply that they are PROPm-satisfied with respect to the original problem instances as well.

In order to ensure this fact, we combine the statement of Observation 3 with the definition of proportional sub-problems and decompositions. In particular, our definition of proportionality for a sub-problem guarantees that the conditions of Observation 3 are met. Since we ensure that proportionality is maintained after every execution of the UPDATEDECOMPOSITION subroutine, we guarantee that the combination of PROPm allocations for the generated sub-problems yields a PROPm allocation for the original problem. □

7 Running Time

We now move to demonstrate that Algorithm 1 completes in time polynomial in the number of agents and items. Lines 1 through 9 correspond to the "divide phase" and lines 10 through 14 correspond to the "conquer phase".

The running time of Algorithm 1 can be expressed as

$$T(m, n) = f(m, n) + \sum_{j=1}^{k} T(m_j, n_j),$$

where $f(m, n)$ denotes the cost of the main call with m items and n agents, and the sum captures the cost of the recursive calls. The number of recursive calls is k (equal to the number of sub-problems from lines 12 and 13), while m_j and n_j are the number of items and agents, respectively, of the j-th sub-problem. Since all the sub-problems consist of distinct bundles and agents, we must have $\sum_{j=1}^{k} m_j \leq m - 1$ and $\sum_{j=1}^{k} n_j \leq n - 1$. Thus the width of any level of the recursion tree is at most $\max\{m, n\}$. Furthermore, since the size of each sub-problem strictly decreases through the recursion, the recursion tree has at most depth $\min\{m, n\}$. This means the total number of vertices in the recursion tree is polynomial in n and m. All that remains is to show $f(m, n)$ is polynomially bounded in m and n.

Producing the divider's bundles takes polynomial time since it requires only sorting the items in non-decreasing order of value and a linear pass over the sorted items. In the body of the for loop of Algorithm 1, computing the initial value of c takes time linear in the number of agents and items by asking each agent their value for each item in $S_1 \cup \cdots \cup S_t$. This initial value of c is at most $n - 1$. As demonstrated in subsection 6.1, at each iteration of the while loop (beginning at line 6) in Algorithm 1, the UPDATEDECOMPOSITION subroutine either increases the value of $|D.agents|$ from $t - 1$ to t or decreases the value of c. Thus, the number of iterations of the while loop is at most n at any iteration of the for loop.

We now demonstrate that the body of the while loop takes polynomial time. Note that computing the new value of c after then UPDATEDECOMPOSITION subroutine takes linear time (in the number of agents and items). To verify that UPDATEDECOMPOSITION also takes polynomial time, observe that the sub-problem graph induced by D in iteration t of the for loop contains at most $t - 1$ vertices (which would occur when all $t-1$ agents in D and bundles are assigned to distinct sub-problems). We then add two additional vertices w_α and w_β so there are at most $t + 1 = O(n)$ vertices overall at any iteration of the for loop. Note that checking if an edge exists between two vertices in the graph takes time linear in the number of agents and goods in the two sub-problems and each sub-problem has at most n agents and m items. Thus, we can construct the graph in time $O(n \cdot (n+m))$. Finally, computing the set R of reachable vertices from w_α can be accomplished by a simple breadth-first-search which is known to take time linear in the number of the vertices and edges in the graph. Since updating the decomposition just requires propagating changes along a path of length at most n, the entire UPDATEDECOMPOSITION process takes polynomial time.

8 Conclusion

In this paper, we solve the problem of computing PROPm allocations among agents with additive valuations, but we leave open another interesting problem proposed in [Baklanov *et al.*, 2020]: the question of the existence and computation of an *average-EFx* (a-EFx) allocation. To determine if an agent is a-EFx satisfied by some allocation, we remove i's least favorite item from each other agent's allocated bundle, and then ask that i's value for her own bundle is at least as high as her average value for all the other agents' bundles. This notion is stronger than PROPm and may provide an interesting stepping stone toward the, much harder, EFx problem.

Acknowledgements

The first author gratefully acknowledges support from the HSE University Basic Research Program. The last two authors were partially supported by NSF grants CCF-2008280 and CCF-2047907. We would also like to thank Maxim Timokhin for helpful feedback toward improving our algorithm.

References

[Amanatidis *et al.*, 2020] Georgios Amanatidis, Evangelos Markakis, and Apostolos Ntokos. Multiple birds with one stone: Beating 1/2 for EFX and GMMS via envy cycle elimination. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1790–1797, Apr. 2020.

[Aziz *et al.*, 2020] Haris Aziz, Hervé Moulin, and Fedor Sandomirskiy. A polynomial-time algorithm for computing a Pareto optimal and almost proportional allocation. *Operations Research Letters*, 2020.

[Baklanov *et al.*, 2020] Artem Baklanov, Pranav Garimidi, Vasilis Gkatzelis, and Daniel Schoepflin. Achieving proportionality up to the maximin item with indivisible goods. *CoRR*, abs/2009.09508, 2020. To appear at the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021).

[Barman *et al.*, 2018] Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding fair and efficient allocations. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 557–574, 2018.

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

[Caragiannis *et al.*, 2019] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Trans. Economics and Comput.*, 7(3):12:1–12:32, 2019.

[Chaudhury *et al.*, 2020a] Bhaskar Ray Chaudhury, Jugal Garg, and Kurt Mehlhorn. EFX exists for three agents. In *Proceedings of the 21st ACM Conference on Economics and Computation*, EC '20, page 1–19, New York, NY, USA, 2020. Association for Computing Machinery.

[Chaudhury *et al.*, 2020b] Bhaskar Ray Chaudhury, Telikepalli Kavitha, Kurt Mehlhorn, and Alkmini Sgouritsa. A little charity guarantees almost envy-freeness. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2658–2672. SIAM, 2020.

[Conitzer *et al.*, 2017] Vincent Conitzer, Rupert Freeman, and Nisarg Shah. Fair public decision making. In Constantinos Daskalakis, Moshe Babaioff, and Hervé Moulin, editors, *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 629–646. ACM, 2017.

[Lee, 2017] Euiwoong Lee. APX-hardness of maximizing Nash social welfare with indivisible items. *Inf. Process. Lett.*, 122:17–20, 2017.

[Lipton *et al.*, 2004] Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings 5th ACM Conference on Electronic Commerce (EC-2004), New York, NY, USA, May 17-20, 2004*, pages 125–131, 2004.

[Moulin, 2019] Hervé Moulin. Fair division in the internet age. *Annual Review of Economics*, 11:407–441, 2019.

[Oh *et al.*, 2019] Hoon Oh, Ariel D Procaccia, and Warut Suksompong. Fairly allocating many goods with few queries. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2141–2148, 2019.

[Plaut and Roughgarden, 2018] Benjamin Plaut and Tim Roughgarden. Almost envy-freeness with general valuations. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2584–2603, 2018.