Towards Better Approximation Algorithms for α-EFX Allocations: Theoretical Insights and Empirical Analysis

Arnav Bhargava

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2025

Abstract

Fair division in algorithmic and economic game theory is a very natural problem to describe. We simply wish to partition an indivisible (a set of objects) or infinitely divisible resource (like a cake) amongst n agents or participants — in a fair manner determined by how agents value each resource. However, almost each word in the definition has some nuance to it. Namely, what is *fair*? How do we define *value*? How much of the resource are we allowed to give away — some of it, all of it?

This thesis focuses on the fairness criterion known as *envy-freeness up to any good (EFX)*, a well-studied notion in the context of indivisible goods for which existence remains an open problem. We provide both theoretical and empirical contributions toward approximating EFX allocations.

On the theoretical side, we investigate existing techniques and adapt them to derive either novel results or deeper insights for specific families of instances, including the *Top-N* setting and instances involving up to *9 agents*. We also highlight the limitations of such *ad hoc* approaches. To pursue a more systematic analysis, we draw inspiration from the EFX-with-charity setting [8], employing tools from extremal graph theory—specifically, the construction of *group champion graphs*.

On the practical side, we conduct extensive experiments on the 3PA algorithm proposed in [4]. To better understand how 3PA behaves in practice, we construct *group champion graphs*, described in [8], and use them as a tool to analyze structural patterns in allocations produced by the algorithm. We also develop an efficient implementation of 3PA and other algorithms that demonstrate strong empirical performance across a variety of random input instances.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Arnav Bhargava*)

Acknowledgements

I sincerely want to thank my supervisor Dr. Aris Filos-Ratsikas, for his outstanding mentorship and unwavering support. Our discussions have been illuminating and insightful. Finally, I would also like to thank my parents and friends who supported me throughout.

Table of Contents

Chapter 1

Theoretical Preliminaries

1.1 Problem of Fair Division

Fair division in algorithmic and economic game theory is a very natural problem to describe. We simply wish to partition an indivisible (a set of objects) or infinitely divisible resource (like a cake) amongst n agents or participants - in a fair manner determined by how agents value each resource. However, almost each word in the definition has some nuance to it. Namely, what is *fair*? How do we define *value*? How much of the resource are we allowed to give away - some of it, all of it?

This thesis focuses specifically on the case where resources are indivisible, valuations are additive, and our notion of fairness is Envy-Freeness up to one good (EFX). We give a general description of the landscape of such problems while providing formal definitions where necessary.

1.1.1 Formal Definition

An instance of fair division of indivisible goods is $I = (N, M, \{v_i\}_{i \in N})$ where the finite sets N and M correspond to agents and goods, respectively. We are also given a family of *valuation functions* $v_i : M \to \mathbb{R}, \forall i \in N$ corresponding to the valuation of an agent on each good. For the purposes of this thesis, we require that each valuation function obeys the following properties:

1. *Normalised:* $v_i(\emptyset) = 0$
2. *Monotone:* $v_i(S) = v_i(T), \forall S \subseteq T \subseteq M$
3. *Additive:* $v_i(S) = \sum_{g \in S} v_i(g), \forall S \subseteq M$

Note that the additive property of the valuation functions allows us to define the valuation of any arbitrary nonempty subset of goods $B \subseteq M$, referred to as a *bundle* here onward, as the sum of individual goods in it. Researchers have also studied the problem for a general class of valuation functions with relaxed properties such as ones that are *subaddive* or *submodular* [2].

In the classical context, the task is to compute a *fair allocation* of these goods among agents.

Definition 1.1.1 (Allocation)**.** An allocation is a $|N|$-tuple of bundles of M, $X = (X_1, \cdots, X_{|N|})$ where $X_i \in M$ such that:

1. *Pairwise Disjoint:* $X_i \cap X_j = \emptyset, \forall i \neq j$
2. *Finite Cover:* $\cup_{i \in N} X_i = M$

Clearly, an allocation partitions the set of goods M, however, we also allow for *partial* allocations, which may leave some goods unallocated such that *partial* if $\cup_{i \in N} X_i \subsetneq M$, that is, there exist unallocated goods for a given allocation.

Definition 1.1.2 (Pool [4])**.** Given an instance $I = (N, M, \{v_i\}_{i \in N}$ and a partial allocation X, we define the *pool* denoted as $Pool(X)$ to the be the set of unallocated goods $M \setminus (\cup_i X_i)$.

The problem of fair division of indivisible goods has retained continued interest owing to stronger or weaker notions of fairness one can impose that can change the flavour of the problem altogether.

1.2 Notions of Fairness

There are two major notions of fairness that are often studied, namely *envy-freeness (EF)* and *proportionality (PROP)*.

Definition 1.2.1 (Envy-Freeness (EF))**.** Given an allocation $X = (X_1, \cdots, X_n)$, X is said to be *envy-free (EF)* if $v_i(X_i) \geq v_i(X_j)$ for all $i, j \in N$.

Definition 1.2.2 (Proposrtionality (PROP))**.** Given an allocation $X = (X_1, \cdots, X_n)$, X is said to be *proportional (PROP)* if $v_i(X_i) \geq v_i(M)/|N|$ for all $i \in N$.

It is quite easy to see that EF implies PROP, as we can simply sum over all j on both sides of the inequality in the EF inequality to get the PROP as well, the converse is not always true. These notions of fairness encapsulate a natural sense of what it means to divide something fairly, however, EF is stronger notion of fairness than PROP. In a sense, PROP lower bounds the value of the bundle for each agent, while, EF imposes relations in between other agents way. As we shall see, even requiring EF is too strong of a notion to do any meaningful analysis for general instances of the problem of fair division of indivisible goods. In fact, the problem of finding out whether an arbitrary instance allows for a EF or PROP allocation is **NP**-complete which can be shown via a reduction from PARTITION. [16]

1.2.1 Relaxations

Due to the intractability of finding the existence of EF or PROP allocations, much of the research has therefore hinged on relaxations of these notions of fairness. Predominantly, *Envy-freeness up to one good (EF1)* [13] and *Envy-freeness up to any good (EFX)* [6]. We give formal definitions of both.

Definition 1.2.3 (Envy-Freeness up to One Good (EF1))**.** Given an allocation $X = (X_1, \cdots, X_n)$, X is said to be *envy-free up to one good (EF1)* if $v_i(X_i) \geq v_i(X_j \setminus \{g\})$ for *some* $g \in X_j$ for all $i, j \in N$.

Definition 1.2.4 (Envy-Freeness up to Any Good (EFX))**.** Given an allocation $X = (X_1, \cdots, X_n)$, X is said to be *envy-free up to any good (EFX)* if $v_i(X_i) \geq v_i(X_j \setminus \{g\})$ for *any* $g \in X_j$ for all $i, j \in N$.

The intuition behind these notions is to allow some envy between agents which can be resolved by removing either one particular good, as in the case of EF1, or any one good as in EFX from an agents bundle. Clearly EF is a more strict notion of fairness than both EF1 and EFX. Moreover, it is also clear to see EFX is a stronger notion than EF1. Consider the two examples of instances below.

	g_1	g_2	g_3	g_4	g_5
n_1	15	3	2	2	6
n_2	7	5	5	5	7
n_3	20	3	3	3	3

Instance 1, $|N| = 3, |M| = 5$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
n_1	9	2	5	7	4	6	8
n_2	3	2	5	7	4	6	8
n_3	3	2	5	7	4	6	0
n_4	3	2	5	0	4	6	0
n_5	3	2	0	0	4	0	0

Instance 2, $|N| = 5, |M| = 7$

Table 1.1: Two instances of Fair Division of Indivisible Items

We are given the following allocations for Instance 1, where A is EF1 and B is EFX:

$$\begin{aligned} A_{n_1} &= \{g_3, g_4\}, & B_{n_1} &= \{g_4, g_5\}, \\ A_{n_2} &= \{g_2, g_5\}, & B_{n_2} &= \{g_2, g_3\}, \\ A_{n_3} &= \{g_1\}, & B_{n_3} &= \{g_1\}. \end{aligned}$$

We are able to obtain an EF1 allocation if we remove an especially valuable good (from the perspective of i) from js bundle, however, EFX requires that the removal of *any* item from the j's bundle, even not-so-valuable ones, should result in an envy-free allocation.

The existence of EF1 allocations for arbitrary instances is quite easy to show, in fact, two well known algorithms are able to achieve this. We introduce the *Round-Robin* and *Envy-Cycle Elimination* algorithms. As we shall see, these algorithms often appear as subroutines in algorithms studying EFX allocations as well.

1.3 Common Algorithms and Guarantees

We introduce the general form of these algorithms. Oftentimes, parametric versions of these algorithms appear in literature. We propose that these algorithms run in polynomial time, without offering careful analysis of their running time for brevity.

1.3.1 Round-Robin

Algorithm 1 Round Robin

Input: $I = (N, M, \{v_i\}_{i \in N})$ where $|N| = n$ goods and $|M| = m$ agents
Output: Allocation $X = (X_1, \cdots, X_n)$

```
for each agent i ∈ N do
    X_i ← ∅
end for
for l = 1, ··· , m do
    i ← l mod n
    Let g* ∈ arg max_{g∈M} v_i(g)
    X_i ← X_i ∪ {g*}
    M ← M \ {g*}
end for
```

The intuition of the Round-Robin algorithm is that given a fixed ordering of the agents we simply to deal the goods out to each agent such that we each agent gets their most valuable good from the ones available. In each round, we give one good to each agent. The crucial bit here is the fixed ordering of the agents in a round. Although it is has been shown that Round-Robin still gives an EF1 allocation given different orderings in each round, we will need a fixed ordering later when analysing EFX allocations. [15]

Proposition 1.3.1. Given an instance $I = (N, M, \{v_i\})$, Round-Robin gives an EF1 allocation.

Proof. Consider two agents i and j, such that i chooses before j according to the fixed ordering. As i has picked their favorite good before j, it cannot envy j in that round. On the other hand, j might envy i. Let the first good chosen by i be g. From that point on, we can consider the execution of the algorithm on the remaining goods as a fresh run where j picks before i. So, j does not envy i bundle after the removal of good g from it. That is j can only envy i in the first round of choices, wherein, only one good is awarded to each agent whose removal can resolve the envy. □

1.3.2 Envy-Cycle Elimination

This algorithm is due to [13], however, the version presented here is from [2]. The crux of this algorithm relies on modeling envy via a directed graph, called the envy graph.

Definition 1.3.1 (Envy Graph)**.** Given an instance $I = (N, M, \{v_i\})$ and an allocation $X = (X_1, \cdots, X_{|N|})$, we create a directed graph with nodes from N and edges

$$(i, j) \iff v_i(X_j) > v_i(X_i)$$

. This is called the *envy graph* with respect to an allocation X (whether complete or partial) which we denote as $G(X) = (N, E(X))$.

Given this envy graph we can perform Envy-Cycle Elimination as follows.

Algorithm 2 Envy-Cycle Elimination

Input: $I = (N, M, v)$ where $|N| = n$ goods and $|M| = m$ agents
Output: Allocation $X = (X_1, \cdots, X_n)$

for each agent $i \in N$ **do**
    $X_i \leftarrow \emptyset$
end for
for $l = 1, \cdots, m$ **do**
    while there does not exist and unenvied agent **do**
        Find an envy-cycle $C = (i_1, \cdots, i_d)$ and resolve cycle as follows:
        $X_{i_j}^C \leftarrow$ Swap-Cycle(C)
        $X_i \leftarrow X_i^C, \forall i \in C$
    end while
    Let i be an unenvied agent
    Let $g^* \in \arg\max_{g \in M} v_i(g)$
    $X_i \leftarrow X_i \cup \{g^*\}$
    $M \leftarrow M \setminus \{g^*\}$
end for

Clearly, an agent is *unenvied* if it's in-degree in the directed envy graph is 0, we call such agents *sources*. The intuition behind this algorithm is that if a cycle $C = (i_1, i_2, \cdots i_k)$ exists in the graph, we must have a chain of inequalities $v_{i_1}(X_{i_2}) > v_{i_1}(X_{i_1}), v_{i_2}(X_{i_3}) > v_{i_2}(X_{i_2}) \cdots, v_{i_k}(X_{i_1}) > v_{i_k}(X_{i_k})$. Since each agent values the bundle of the agent it envies strictly more, swapping the bundles backwards along the cycle resolves the envy-cycle, which reduces the in-degree of the source i_1 by 1. We can then proceed to give each unenvied agent their favorite good from the set of unallocated goods resolving cycles along the way.

A crucial property of the *Swap-Cycle* procedure is that is that each agent involved in the cycle gets *at least* a better bundle than the one it had while all other agents retain their previous bundle. This implies that the final allocation after resolving cycles *Pareto dominates* the previous allocation. That is, $\forall i \in N, v_i(\tilde{X}_i) \geq v_i(X_i)$. This property, is an invariant for the Envy-Cycle algorithm and, as we shall see, can be exploited to relaxed versions of the algorithm.

It was shown in [13] that this algorithm produces an EF1 allocation. The version of the algorithm presented here gives the most valuable good to an unenvied agent currently available, this is not necessary for producing EF1 allocations but it is needed for the analysis of EFX allocations.

1.4 EFX Allocations and Further Relaxations

The problem of even finding EFX allocations is significantly harder for arbitrary instances compared to EF1. It is even more difficult to find *polynomial-time* algorithms that construct EFX allocation on arbitrary instances. However, for certain restrictions of instances we can show EFX allocations can be found in polynomial-time. For instance, consider an instance $I = (N, M, \{v_i\}_{i \in N})$ wherein each agent has the same order for the valuations of each good, that is, there exists an ordering of the goods $g_1, \cdots g_{|M|}$ such

that $v_i(g_1) \geq v_i(g_2) \geq \cdots > v_i(g_{|M|})$ for all $i \in N$. In this case Envy-cycle elimination produces an EFX allocation for every such instance. [15]

Similarly, if we are given an instance in which valuation functions for each good are the same or globally lower bounded $v_i(g) \geq V$ for some $V \in \mathbb{R}$ for each $i \in N$. We are similarly, able to find EFX allocations [15].
On the other hand, the general case eludes polynomial time algorithms. Hence, we aim to find allocations that are *approximately* EFX.

Definition 1.4.1 (α-EFX)**.** Given an allocation $X = (X_1, \cdots, X_n)$, X is said to be α-(EFX) if $v_i(X_i) \geq \alpha \cdot v_i(X_j \setminus \{g\})$ for *any* $g \in X_j$ for all $i, j \in N$ and $\alpha \in (0, 1]$.

More generally, it has been shown that $\frac{1}{2}$-EFX allocations always exist for arbitrary instances. Crucially, Envy-Cycle-Elimination gives a $\frac{1}{2}$-EFX allocation for arbitrary instances [15]. Secondly we have the following:

Theorem 1.4.1. *Envy-Cycle Elimination gives a $\frac{1}{2}$-EFX allocation for any arbitrary instance $I = (N, M, \{v_i\}_{i \in N})$, where in the first $n = |N|$ rounds we award the sources their favorite item from the pool. [14; 15]*

Proof. In the first $n = |N|$ rounds, each agent gets their favourite good from the pool. Any agent with one good is trivially EFX and thereby, $\frac{1}{2}$-EFX, towards any other agent with one good. Moreover, the envy graph is cycle free. Let X be the allocation after n rounds; if a source s is awarded a good h in any round $m > n$ we have $v_i(X_i) > v_i(X_s)$ (s is unenvied) and $v_i(X_i) > v_i(h)$ (this good was available to i in the first n rounds) which implies $2 \cdot v_i(X_i) > v_i(X_s \cup \{h\})$. The result follows. □

More recently, an algorithm proposed in [5] gives a 0.618-EFX allocation where $0.618 \approx (\phi - 1)$ with ϕ the golden ratio. Finally, the most recent algorithm [4] finds a $\frac{2}{3}$-EFX allocation for certain restrictions on valuations and the number of agents.

The table below summarizes the progress in finding α-EFX allocations in polynomial time.

Progress on Finding Polynomial-time α-EFX Allocations				
Paper	Bound	Arbitrary Instances	Restrictions on Valuations	Restriction on Number of Agents
Amanatidis (2021a) [3]	1	–	✓ (bi-valued)	–
Plaut & Roughgarden (2020) [15]	1	–	–	✓ (2 agents)
Chaudhary et al. (2020) [7]	1 (Pseudo-poly.)	–	–	✓ (3 agents)
Plaut & Roughgarden (2020) [15]	$\frac{1}{2}$	✓	–	–
Amanatidis et al. (2020) [5]	$0.618 \approx (\phi - 1)$	✓	–	–
Amanatidis et al. (2024) [4]	$\frac{2}{3}$	–	✓ (tri-valued or multi-graph)	✓ (≤ 7 agents)

Table 1.2: Progress on finding polynomial-time α-EFX allocations

1.5 Advanced Definitions

We briefly explain the definitions in this section for a general instance $I = (N, M, \{v_i\}_{i \in N})$. In the following definitions, $\varepsilon \leq \frac{1}{2}$.

Definition 1.5.1 (Heavily Envies [7])**.** Agent i *heavily envies* a set $S \subseteq M$ if $v_i(X_i) \leq (1-\varepsilon)v_i(S)$

Definition 1.5.2 (Strongly Envies [7])**.** Agent i *strongly envies* a set S if it heavily envies S and S is a *proper* subset of M.

Clearly, if $\varepsilon = \frac{1}{3}$ we arrive at our familiar definitions, namely, i is $\frac{2}{3}$-EFX towards any agent j if it doesn't strongly envy it's bundle, as $X_j \setminus \{g\}$ is a proper subset of X_j.

Definition 1.5.3 (Championship [7])**.** Given allocation X and a good $g \in Pool(X)$, we say that an agent i is a *champion* for agent $i \neq j$ w.r.t g if there is a set $S \subset X_j \cup \{g\}$ such that:

1. $v_i(X_i) < (1-\varepsilon)v_i(S)$
2. For all $k \in N$ (which includes i and j) $(1-\varepsilon)v_k(S \setminus \{h\}) \leq v_k(X_k)$ for all $h \in S$.

Definition 1.5.4 (Valuable Good [7])**.** Given a partial allocation X, consider an agent i and a good $g \in Pool(X)$. g is *valuable* to an agent i if $v_i(g) > \varepsilon v_i(X_i)$.

Example 1.5.1. Given a partial allocation X and $\varepsilon \leq \frac{1}{2}$. If $i \in N$ strongly envies the bundle $X_s \cup \{g\}$ in $G(X)$ then it must find g valuable.

1.6 Scope of the Thesis

The scope of this thesis is to work both theoretically and experimentally towards Chapter 2, analyses techniques in better approximations for α-EFX. Particularly, we look at the task of obtaining $\frac{2}{3}$-EFX by applying a general approximation framework. We describe an algorithm called 3PA that gives us a partial allocation with particular guarantees.

These techniques, as we shall see, are ad-hoc and don't seem to generalize well for arbitrary instances. We precisely define the pitfalls of such techniques, in particular fror the task of giving away goods that are *critical*. With this aim, we make novel contributions in producing an almost complete proof of $\frac{2}{3}$-EFX for at most 9 agents, improving on the previous result for at most 7 agents. We also examine Top-N instances and give insights as to the structure of the problem along with an example displaying the complexity of the task.

Chapter 3, works towards examining the landscape of *systemically* giving away goods that are critical for a few agents after applying Algorithm 3. We also examine the relationship between critical and valuable goods theoretically. To do this we introduce techniques from [8] discussing the problem of *EFX with Bounded Charity*, which is the task of finding EFX allocations that leave at most a (sublinear) number of goods unallocated. In the paper, a surprising connection linking a problem in extremal graph theory (*Mininmum Rainbow Cycle Number*) was found. This was then used to illuminate the path towards *complete* α-EFX allocations through the construction of a *Group Champion Graph*. We give ample motivations behind the theory, some insight into adapting the described techniques to achieve complete $\frac{2}{3}$-EFX allocations using Algorithm 3 by giving away particular critical goods.

Finally, Chapter 4 describes the implementation and experimentation of all algorithms described in the thesis up until that point. We discuss generating distribution drawn random valuation functions for various types of instances as testing data. We also describe the practicalities of implementing the described algorithms through an efficient interface, we explore the execution time implications of these through experiments. We are also able to construct the group champion graphs described in Chapter 3 and examine the general properties of such a graph given the various distributions.

1.6.1 Target Audience

The target audience for this thesis is undergraduate students. Accordingly, we aim to include numerous examples to illustrate theoretical constructs and to provide deeper insights and connections to previously developed theory when presenting proofs.

Chapter 2

Approaching $\frac{2}{3}$-EFX: Techniques and Challenges

In this chapter we explore the various techniques aimed at achieving $\frac{2}{3}$-EFX. We explore some easy cases in which imposing more structure around valuations or restricting the number of agents can yield good results. However, as we shall discover, general instances are quite intricate to reason about and allude to applying more sophisticated mathematical machinery.

2.1 Applications of a General Approximation Framework

A common approach in analyzing algorithms is to define a *potential function* which improves throughout it's execution. Informally, based on our previous discussion regarding Envy-Cycle Elimination (ECE), the algorithm always finds a Pareto-dominating allocation at each step. An even careful analysis reveals that an even stronger claim holds; we can start off with a partial α-EFX allocation X (for some α) that leaves some goods unallocated. We can then complete this allocation via ECE to get an allocation $\tilde{X}$ such that not only $\forall i \in N, v_i(X_i) \geq v_i(\tilde{X}_i)$, but this complete allocation is also α-EFX. Thereby, this algorithm allows for Pareto domination of allocations to serve as a good candidate for a potential function. This idea was refined in [14], we present it here.

Theorem 2.1.1. *The algorithm below computes a* $\min(\alpha, \frac{\beta}{\beta+1})$*-EFX allocation.*

1. *For* $\alpha, \beta > 0$*, compute a partial* α*-EFX allocation* $\mathcal{X} = (X_1, \cdots, X_n)$ *with the property that*
$$v_i(X_i) \geq \beta \cdot v_i(h), \forall i \in \mathbb{N}, \forall h \in M \setminus \bigcup_{j \in N} X_j$$
2. *Run ECE starting from* X*, until there are no unallocated items. Call this final allocation* $\tilde{X}$
3. *Output* $\tilde{X}$*.*

Proof. Fix an agent i and suppose we also have an agent j that receives X_j as her final bundle. Then $v_i(X_i) \geq \alpha \cdot v_i(X_j \setminus \{g\}), \forall g \in X_j$, due to the partial allocation X being $\alpha-$EFX. Now assume that every agent gets as a final bundle a strict super-set of the partial allocation. Let h be the final good allocated to agent j. Then we have that $v_i(\tilde{X}_i) \geq v_i(S_i) \geq \beta \cdot v_i(h)$ (1). Before this, we had $v_i(\tilde{X}) \geq v_i(X_j \setminus \{h\}) \implies \beta \cdot v_i(B) \geq \beta \cdot v_i(A_j \setminus \{h\})$ (2). Where B is the bundle that i eventually gets. Using (1) and (2) we get:

$$(\beta + 1) \cdot v_i(B) \geq \beta \cdot v_i(A_j)$$

We then have that $v_i(A_i) \geq v_i(B) \geq \frac{\beta}{\beta+1} \cdot v_i(A_j)$, as ECE doesn't decrease the value of anyone's bundle. Which means that i is actually $\frac{\beta}{\beta+1}-$EF and therefore $\frac{\beta}{\beta+1}-$EFX towards j as well. □

This gives us a general approximation framework, which reduces the problem of finding complete $\min(\alpha, \frac{\beta}{\beta+1})$-EFX allocations to computing a *approximation amenable partial* allocation X satisfying the following:

1. Each agent is α-EFX towards every other agent.
2. For every agent i we have that $\forall g \in Pool(X), v_i(X_i) < \beta v_i(g)$.

We then complete such an allocation using ECE.

2.1.1 Top-N Algorithm

As a first application of the approximation framework, we analyze instances $I = (N, M, \{v_i\}_{i \in N})$ where we can partition M in to two sets T and $M - T = B$ such that $|T| = |N| = n$ and for any goods $g \in T$ and $g' \in B$ we have that $\forall i, v_i(g) > v_i(g')$. We call the instance a *Top-N instance* and T the *Top-N set*. Note that, although all agents believe in *which* the n most valuable goods are, they may not necessarily agree on their value. That is, for any two agents, i, j, it may hold that $v_i(g) \neq v_j(g)$ for some goods $g \in T$. [14] presents a simple iterative algorithm that applies the approximation framework with $\alpha = \frac{2}{3}$ and $\beta = 2$ on instances with a common Top-N set and achieves $\frac{2}{3}$-EFX, the pseudo-code and proof of correctness are included in Appendix B. Given an arbitrary ordering for the agents, the algorithm first assigns to an agent its favorite good from T (content) (at that time) or it's favorite from B (non-content). Those with a good from B get assigned another good from T in a second. An agent keeps it's favorite good $h \in T$ if $\frac{2}{3}$-EF towards the worst possible size-2 bundle it could get - $\{h', b\}$ where b is the best good in b and h' is the *worst* good in T at the time of choosing. This allocation is approximation amenable and therefore can be completed using ECE.

2.1.2 Draft-and-Eliminate

We can apply the approximation framework with $\alpha = \phi - 1$ and $\beta = \phi$ as well, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The algorithm proposed in [5] finds a partial allocation on *general valuation* functions such that it is $0.618 \approx (\phi - 1)$-EFX for all agents and one such that $\forall i \in N, v_i(X_i) \geq \phi v_i(g), g \in Pool(X)$. The algorithm awards at most two

goods to each agent. It first simulates one round of Round-Robin with lexicographically ordered agents and reorders the agents that would have high envy (up to a factor of ϕ) for the best good chosen by other agents that were considered before them. These *non-content* agents with high envy get to pick their good first, while the others are awarded a second good in Round-Robin fashion in the reversed order. We give the pseudo-code of the algorithm in Appendix A, and avoid presenting the proof of it's correctness. The proof heavily exploits the fact that $\phi - 1 = \frac{1}{\phi}$.

2.1.3 3PA Algorithm

The remainder of this section focuses on achieving $\frac{2}{3}-$EFX for arbitrary instances. While exploring these easy cases, there is a hope that we can apply the approximation framework for larger $\alpha > \phi - 1$ and find an iterative polynomial time algorithm that gives a framework-amenable partial allocation. Moreover, we hope that we are able to use this algorithm for *arbitrary* instances. With this aim in mind, we introduce some new terminology and more nuanced graph structures, adapted from [4]

Definition 2.1.1 (β-Critical Goods)**.** Given an allocation X, we call any good $g \in Pool(X)$, β-critical for i if $v_i(X_i) < \beta v_i(g)$ for $\beta \geq 1$. An approximation amenable partial allocation has no β-critical goods for any agent.

Definition 2.1.2 (Reduced Envy Graph)**.** Given an allocation X, the *reduced envy graph* $G_r(X) = (N, E_r(X))$ is a subgraph of the envy graph $G(X) = (N, E(X))$ where we remove all edges (i, j), such that $|X_i| > 1, |X_j| = 1$ and i is $\frac{2}{3}$-EF towards j. That is

$$E_r(X) = E(X) \setminus \{(i, j) : |X_i| = 2, |X_j| = 1, v_i(X_i) \geq \frac{2}{3} v_i(X_j)\}$$

Definition 2.1.3 (Enhanced Envy Graph)**.** Given an allocation X, the *enhanced envy graph* $G_e(X) = (N, E_e(X))$ is a supergraph of the reduced envy graph $G_r(X) = (N, E_r(X))$ where we add edges (i, s), such that s in $G_r(X)$, $|X_i| = 1, |X_s| > 1$ and $v_i(X_s) \geq \frac{2}{3} v_i(X_i)$. That is

$$E_e(X) = E_r(X) \cup \{(i, s) : |X_i| = 1, |X_s| = 2, v_i(X_s) \geq \frac{2}{3} v_i(X_i), s \text{ is a source in } G_r(X)\}$$

When the allocation X is clear from context, we just write, G, G_r and G_e instead of $G(X), G_r(X), G_e(X)$. We give a brief example of these graphs with respect to an allocation.

Example 2.1.1. Consider Instance 2 in Table 1.1 the allocation be:

$$X = 1 : [8], 2 : [5], 3 : [7], 4 : [6, 1], 5 : [2]$$

Then Figure 2.1 displays G and G_r, we label the bundle j that agent i envies.

Clearly as G_r is a subgraph of G and G_e is a super-graph of G_r. Thus, if s is a source in G it is also a source in G_r, and if s is a source in G_e it is also source in G_r.

We have an algorithm (3PA) [4] for applying the approximation framework with $\alpha = \frac{2}{3}, \beta = 2$. This algorithm assigns at most two goods to each agent and achieves a partial

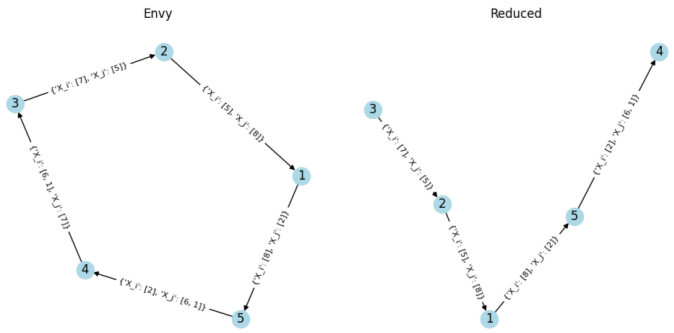


Figure 2.1: Envy and Reduced Graphs

allocation that is $\frac{2}{3}$-EFX leaving at most one good that is $\beta = 2$-critical for each agent. Moreover, it provides certain guarantees for the respective agents with respect to the pool and also towards each other. From hereon, we refer to a good g as critical for i if it is 2-critical for i, that is, $v_i(X_i) < 2v_i(g)$.

Proposition 2.1.2. The following hold for the (partial or complete) allocation X^1 at termination of Algorithm 3 for an arbitrary instance $I = (N, M, \{v_i\}_{i \in N})$ starting off with a seed allocation X:

1. Every agent i with $|X_i^1| = 1$ is EFX (and by extension also $\frac{2}{3}$-EFX) towards any other agent.
2. Every agent i with $|X_i^1| = 2$ is $\frac{2}{3}$-EFX. towards any other agent j with $|X_j| = 2$.
3. There is at most one good $g_i \in Pool(X^1)$ such that for all agents $i \in N$ with $|X_i^1| = 1$, $v_i(X_i^1) < 2v_i(g_i)$ (critical for i). Moreover this good is $v_i(X_i^1) \geq \frac{3}{2} v_i(g_i)$. Combining the two inequalities we have, $\frac{3}{2} v_i(g_i) < v_i(X_i^1) < 2v_i(g_i)$.
4. Agents with 2 goods in their bundle do not have any critical goods. That is for all agents i with $|X_i^1| = 2$, we have that $\forall g \in Pool(X^1), 2v_i(g) \leq v_i(X_i^1)$.
5. If X^1 is not a complete allocation then there is at least one source s in the *enhanced* envy graph $G_e(X^1)$. Every such source has $|X_s| = 2$.

We call the allocation returned by Algorithm 3 a *3PA allocation*. A seed allocation is one that satisfies properties 1 and 2 for every agent. Clearly, any allocation that assigns one good to each agent can serve as a valid seed. This is because, every agent with one good is trivially EFX, and by extension, $\frac{2}{3}$-EFX towards any other agent and vice versa. The proofs of the algorithm's correctness and its termination in polynomial time are omitted. However, we highlight the following claims as they will be important in understanding the running time of the algorithm.

- Once the bundle for an agent i changes from S to S', it will never be the case that $X_i = S$ for any future iteration.
- At each iteration one of the bundles is never considered again by ≥ 1 agent.

Algorithm 3 3PA Algorithm adapted from [4]

Input: A general allocation $I = (N, M, \{v_i\}_{i \in N})$ and a partial allocation X satisfying properties 1 and 2.

Output: An allocation X^1 preserving all above properties.

while $Pool(X) \neq \emptyset$ **do**
  if there is $i \in N$ with $|X_i| = 1$ and a good $g \in Pool(X)$ s.t. $v_i(g) > v_i(X)$ **then** ▷ *(Step 1)*
    $Pool(X) \leftarrow (Pool(X) \cup \{X_i\}) \setminus \{g\}$ and $X_i \leftarrow \{g\}$
  else if there is $i \in N$ with $|X_i| = 2$ and a good $g \in Pool(X)$ s.t. $v_i(g) > \frac{3}{2} v_i(X_i)$ **then** ▷ *(Step 2)*
    $Pool(X) \leftarrow (Pool(X) \cup \{X_i\}) \setminus \{g\}$ and $X_i \leftarrow \{g\}$
  else if there is $i \in N$ with $|X_i| = 1$ and goods $g_1, g_2 \in Pool(X)$ s.t. $v_i(\{g_1, g_2\}) > \frac{2}{3} v_i(X_i)$ **then** ▷ *(Step 3)*
    $Pool(X) \leftarrow (Pool(X) \cup \{X_i\}) \setminus \{g_1, g_2\}$ and $X_i \leftarrow \{g_1, g_2\}$
  else if there is $i \in N$ with $|X_i| = 2$ and goods $g \in Pool(X)$ and $g' \in X_i$ such that $v_i(g) > v_i(g')$ **then** ▷ *(Step 4)*
    $Pool(X) \leftarrow (Pool(X) \cup \{g'\}) \setminus \{g\}$ and $X_i \leftarrow (X_i \cup \{g\}) \setminus \{g'\}$
  else if there are cycles in the reduced envy graph $G_r(X)$ **then** ▷ *(Step 5)*
    $X \leftarrow \text{AllCyclesResolution}(X, G_r)$
  else if there is a source s with $|X_s| = 1$ in G_r **then** ▷ *(Step 6)*
    $Pool(X) \leftarrow Pool(X) \setminus \{g^*\}$
    $X_s \leftarrow X_s \cup \{g^*\}$ where $g^* \in \arg\max_{g \in Pool(X)} v_i(g)$
  else if $|\{g \in Pool(X) : \exists i \text{ such that } |X_i| > 1 \text{ and } v_i(g) > \frac{2}{3} v_i(X_i)\}| = 1$ **then** ▷ *(Step 7)*
    $X \leftarrow \text{SingletonPool}(X)$
  else if there are cycles in the enhanced envy graph G_e **then** ▷ *(Step 8)*
    $X \leftarrow \text{AllCyclesResolution}(X, G_e)$
  else
    break
  end if
end while
return X

2.1.3.1 Brief Note on Running Time

Given the above observations it is easy to see that the algorithm clearly terminates in polynomial time. For an instance with n agents and m goods, there are $\binom{m}{2} + \binom{m}{1} \leq m^2$ bundles for every agent since the maximum size of the bundles for any agent is 2. If each bundle is never considered again for at most one agent at each step, then all bundles will be exhausted in at most nm^2 steps. Thus the number of iterations of the main `while` loop are upper bounded by $nm^2 + 1$. However, in practice there is a heavy overhead incurred by expensive computations such as constructing the various envy graphs at each step as well as operations like finding and resolving cycles within them. Moreover, operations that run linear in the number of unallocated goods are a concern too. We consider our operations to be non-trivial on the seed, that is, instances where $m > n$ as others we can obtain a complete allocation with just one round of Round-Robin.

Finally, our notion of running time depends on our definition of the *size* of the instance $I = (N, M, \{v_i\}_{i \in N})$. The generally accepted notion is that $|I| = |N| \times |M|$. Both the Top-N algorithm and Draft-And-Eliminate run in time polynomial in the number of agents. However, our current running time bound for Algorithm 3 is pseudo-polynomial in the number of agents, as m might be much larger than n, which is usually the case. However, as we shall see, completing the allocation via ECE is the clear bottleneck for all above algorithms for larger instances. We defer the discussions on practical and experimental considerations of Algorithm 3 to Chapter 4.

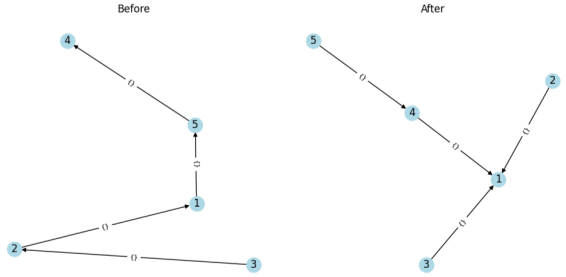
2.1.3.2 Approximation amenable Allocations via 3PA

We note, crucially, that the Algorithm 3 does not produce an approximation amenable partial allocation due to the existence of critical goods. Therefore we must find a clever way of giving out these goods. We particularly note that in a 3PA allocation goods that are critical to at most one agent can be given to a source in the enhanced envy graph that have a path to the given agent, while resolving the envy along such a path. We describe this procedure from [4] here:

We give a brief example illustrating the action of the *PathResolution* or *PathResolution** procedure. The latter assigns the favorite goods from the bundle of the source and the pool respectively.

Example 2.1.2. We show the action on the enhanced envy graph (see Figure 2.2) on an allocation X for Instance 2 in Table 1.1 after doing path resolution star on agent 1 given the source agent 3.

$$X = 1 : [8], 2 : [5], 3 : [7], 4 : [6,1], 5 : [2] \rightarrow X' = 1 : [7,4], 2 : [8], 3 : [5], 4 : [6,1], 5 : [2]$$



Algorithm 4 UncontestedCritical(X, G_e)

Input: A general allocation $I = (N, M, \{v_i\}_{i \in N})$ and a partial allocation X satisfying properties 1 to 5 .

Output: An allocation X^1 preserving all above properties.

$X \leftarrow$ AllCyclesResolution$(X, G_e(X))$
while there exists $i \in N$ and $g_i \in Pool(X)$ s.t. $v_i(g_i) > \frac{1}{2} v_i(X_i)$ **do**
  Let s be a source of G_e s.t. there exists a s-to-i path Π in $G_e(X)$
  if $v_i(X_s \cup \{g_i\}) > v_i(X_i)$ **then**
    $(X_j)_{j \in N} : \exists (j.l) \in \Pi \leftarrow$ PathResolution(X, G_e, Π)
    $X_i \leftarrow X_s \cup \{g_i\}$
  else
    $X_s \leftarrow X_s \cup \{g_i\}$
  end if
  $X \leftarrow$ AllCyclesResolution(X, G_e)
end while
return X

Figure 2.2: Enhanced Envy Graph Before and After *PathResolution**

2.1.3.3 Multi-Graph Instances

We describe an easy application, as presented in [4], where we are able to achieve a approximation amenable partial allocation using Algorithm 3 and using Algorithm as a subroutine.

Definition 2.1.4. We say that an instance $I = (N, M, \{v_i\}_{i \in N})$ is a *multi-graph instance*, if for any agent $i \in N$, there is a undirected labeled multi-graph $H = (V, E)$ with it's vertex set V such that $V \sim N$, and edges E labeled via a bijection to M, such that for any $i \in N$ and for any good $g \in M$, it holds that

$$v_i(g) \neq 0 \iff g \text{ is the label of an edge incident to the vertex labeled as } i$$

Given a 3PA allocation X, consider the set of unallocated goods that are valued by *strictly* more than 1 distinct agents, that is,

$$\mathcal{C} := \{g \in Pool(X) : \exists! i, j \in N, g \text{ is critical for } i, j\}$$

Note that a good can only be critical for an agent if it has a positive valuation for it. In the multi-graph instance case, $g \in \mathcal{C}$ is critical for *exactly* two agents i, j, thus $v_k(g) = 0, \forall k \notin N \setminus \{i, j\}$. Thus, if an agent doesn't have a critical good, it must be the case that $v_i(C) = 0$. On the other hand, if agent i has a critical good in $\mathcal{C}$, we know that X satisfies condition 3 in 2.1.3 so $|X_i| = 1$ and $0 < v_i(g_i) \leq \frac{2}{3} v_i(X_i)$ and has $v_i(g) = 0, \forall g_i \neq g \in \mathcal{C}$. Therefore, $v_i(\mathcal{C}) = v_i(g_i) \leq \frac{2}{3} v_i(X_i)$.
The algorithm in [4] gives all of $\mathcal{C}$ to a source s in $G_e(X)$. From condition 5 in 2.1.3, we know such a source exists, and has $|X_s| = 2$. All other agents retain their bundles, and the continues giving away the rest using Algorithm 2.1.3.3.

2.2 Theoretical Challenges in Adapting 3PA to Easy Cases

The contributions in this section are novel, incomplete however, they provide an insight into the challenges in ad-hoc approaches in giving away certain critical goods.

2.2.1 At Most 9 Agents

[4] presents an algorithm achieving $\frac{2}{3}$-EFX for at most 7 agents. It modifies the Algorithm 3 by adding an extra `Else-If` after Step 8, which gives us the following guarantee.

Proposition 2.2.1. If for some $i \in N$ there is a path from a source s to i in $G_r(X)$ such that $v_i(X_i) < v_i(g) + v_i(g_s)$ for some $g \in Pool(X)$ and $g_s \in X_s$. Perform a path resolution to give i the bundle $\{g, g_s\}$ and give every other agent along the path the bundle they envy. The resulting allocation satisfies Prop 2.1.3 along with $v_i(X_i) \geq v_i(\{g, g_s\})$. Call this version of the algorithm **3PA+**.

In our analysis we adapt the techniques to achieve this for at most 9 agents. Our approach follows from [4]. Given that we have at most 9 agents, we, as before, consider a 3PA allocation X. As in our previous analysis of multi-graph instances, we examine:

$$\mathcal{C} := \{g \in Pool(X) : \exists! i, j \in N, g \text{ is critical for } i, j\} \subseteq Pool(X)$$

Here we reason about the cardinality of $\mathcal{C}$ and aim to give each good in this set to a source in $G_e(X)$. Let S be the set of sources in G_e. Note that by Prop 2.1.3, all sources have $|X_s| = 2, s \in S$, therefore, $S \cap \mathcal{C} = \emptyset$. Given that we have at most 9 agents, and at least one good in $\mathcal{C}$, we can have the following cases:

$\lvert\mathcal{C}\rvert = 1$	$\lvert\mathcal{C}\rvert = 2$	$\lvert\mathcal{C}\rvert = 3$	$\lvert\mathcal{C}\rvert = 4$
$1 \leq \lvert S\rvert \leq 7$	$1 \leq \lvert S\rvert \leq 5$	$1 \leq \lvert S\rvert \leq 3$	$\lvert S\rvert = 1$

We are able to prove **all but one** case: $|\mathcal{C}| = 3, |S| = 1$

We now describe a modified algorithm that can give away most goods in C and give a proof of it's correctness barring one case.

Proof. Let $\tilde{X}$ be the allocation right before line 16 in the above algorithm and X a 3PA+ allocation. Note that for all agents i, $v_i(\tilde{X}_i) > v_i(X_i)$, moreover, the sources we give the goods gets a strictly better bundle and satisfied $\frac{2}{3}-$EFX towards all agents given X it cannot violate $\frac{2}{3}-$EFX given $\tilde{X}$.

Case 1 ($|\mathcal{C}| \leq 2, |S| \geq 1$): If this case holds then we give $\mathcal{C}$ to a source $s \in S$. Let i with $|X_i| = 1$ be an agent that has a critical good in $\mathcal{C}$. Now consider the conditions guaranteed by the termination of 3PA+, namely, by `Step 4` no longer holding, $\forall g_1, g_2 \in Pool(X), v_i(\{g_1, g_2\}) \leq \frac{2}{3} v_i(X_i)$. Thus, $v_i(C) \leq \frac{2}{3} v_i(X_i)$. We also have $v_i(X_s) \leq \frac{2}{3} v_i(X_i)$ from bounds on the envy between agents with size 1 bundles and a source s in the enhanced envy graph respectively from condition 5 in Prop 2.1.3.

$$v_i(\tilde{X}_s) = v_i(X_s \cup C) \leq \frac{2}{3} v_i(X_i) + \frac{2}{3} v_i(X_i) \leq \frac{3}{2} v_i(X_i)$$

Algorithm 5 FEWAGENTSALLOCATE'$(I = (N, M, (v_i)_{i \in N}))$

Require: An additive instance with at most 9 agents.
Ensure: A $2/3$-EFX allocation X.

$X^1 \leftarrow 3\text{PA}^+(I)$
$C \leftarrow \{g \in Pool(X^1) :$ there are distinct $i, j \in N$ so that g is critical for both $i, j\}$
if $|C| = 2$ (say, $C = \{g_1, g_2\}$) and $|S| \geq 2$, $s_1, s_2 \in S$ in $G_e(X^1)$ **then**
    $\tilde{X}_{s_1} \leftarrow X^1_{s_1} \cup \{g_1\}$
    $\tilde{X}_{s_2} \leftarrow X^1_{s_2} \cup \{g_2\}$
    $\tilde{X}_i \leftarrow X^1_i$ for any other $i \in N \setminus \{s_1, s_2\}$
else if $|C| = 3$ (say, $C = \{g_1, g_2, g_3\}$) and $|S| \geq 2$, $s_1, s_2 \in S$ in $G_e(X^1)$ **then**
    $\tilde{X}_{s_1} \leftarrow X^1_{s_1} \cup \{g_1\}$
    $\tilde{X}_{s_2} \leftarrow X^1_{s_2} \cup \{g_2, g_3\}$
    $\tilde{X}_i \leftarrow X^1_i$ for any other $i \in N \setminus \{s_1, s_2\}$
else
    Let s be a source of $G_e(X^1)$
    $\tilde{X}_s \leftarrow X^1_s \cup C$
    $\tilde{X}_i \leftarrow X^1_i$ for any other $i \in N \setminus \{s\}$
end if
$X^2 \leftarrow$ UNCONTESTEDCRITICAL$(\tilde{X}, G(\tilde{X}))$
$X \leftarrow$ ENVYCYCLEELIMINATION$(X^2, G(X^2))$
return X

Thus, i is $\frac{2}{3}$-EF towards s.

Now consider the case where i doesn't have any critical goods in $\mathcal{C}$, it may be the case $|X_i| = 2$ or $|X_i| = 1$. The bundle $\tilde{X}_s \setminus \{g\} = (X_s \cup \{\mathcal{C}\}) \setminus \{g\}$ for some $g \in \tilde{X}_s$ is of size at most 3, it may contain 1 goods from X_s and 2 goods from $\mathcal{C}$ and vice-versa. By, Prop 2.2.1, we have that $v_i(X_i) \geq v_i(\{g, g_s\})$ $g \in Pool(X)$ and a good $g_s \in X_s$. Fix $g \in \mathcal{C}$. Moreover, any other $g' \in \mathcal{C}$ is not critical for i, thus $\frac{1}{2} v_i(X_i) \geq v_i(g')$. Thus,

$$v_i(\tilde{X}_s \setminus \{g\}) \leq v_i(\{g_s, g, g'\}) = v_i(\{g_s, g\}) + v_i(\{g'\}) \leq v_i(X_i) + \frac{1}{2} v_i(X_i) = \frac{3}{2} v_i(X_i) \leq \frac{3}{2} v_i(\tilde{X}_i)$$

On the other hand, $\tilde{X}_s$ may contain 2 goods from X_s and 1 good from C. However, in this case we have at most two goods in $g, g' \in \mathcal{C}$ such that $v_i(\mathcal{C}) = v_i(g') + v_i(g) \leq \frac{1}{2} v_i(X_i) + v_i$ as no good is critical for i in $\mathcal{C}$. Moreover, s is unenvied, hence, we have that $v_i(X_i) > v_i(X_s)$ regardless of the size of i's bundle. Thus, if $g_s, g'_s \in X_s$ and $g \in \mathcal{C}$

$$v_i(\tilde{X}_s \setminus \{g\}) \leq v_i(\{g_s, g'_s, g\}) = v_i(X_s) + v_i(\{g'\}) \leq v_i(X_i) + \frac{1}{2} v_i(X_i) = \frac{3}{2} v_i(X_i) \leq \frac{3}{2} v_i(\tilde{X}_i)$$

Hence we conclude that *every* agent i is $\frac{2}{3}-$EFX towards s in this case.

Case 2 ($|\mathcal{C}| = 2, 2 \leq |S| \leq 5$): In this case the goods in $\mathcal{C}$ are divided among two sources. We fix an agent i with a critical good in $\mathcal{C}$. We have that $|\mathcal{C}| = 2$ and there are $s_1, s_2 \in G_e(\tilde{X})$, with $|X_{s_j}|, j \in \{1, 2\}$. Now suppose we give i's critical good $g_i \in \mathcal{C}$ to any s_j.

$$v_i(\tilde{X}_{s_j}) = v_i(X_{s_j} \cup \{g_i\}) \leq \frac{2}{3} v_i(X_i) + \frac{2}{3} v_i(X_i) \leq \frac{3}{2} v_i(X_i)$$

The first inequality follows from condition 5 in Prop 2.1.3 as above. The second follows from the upper bound on critical goods from condition 3 in Prop 2.1.3. Thus, i is $\frac{2}{3}$-EF towards any source s_j with $j \in \{1,2\}$.
If agent i no critical goods in $\mathcal{C}$. This in turn means that no $g \in \mathcal{C}$ is critical for i and thus $v_i(g) \leq \frac{1}{2}v_i(X_i) \leq \frac{1}{2}v_i(\tilde{X}_i)$.

$$v_i(\tilde{X}_{s_j}) = v_i(X_{s_j} \cup \{g\}) \leq \frac{2}{3}v_i(X_i) + \frac{2}{3}v_i(X_i) \leq \frac{3}{2}v_i(X_i)$$

Case 3 ($|\mathcal{C}| = 3, |\mathbf{S}| \geq 2$) In this case we split the goods amongst two sources $s_1, s_2 \in G_e(X)$.

Every agent is i $\frac{2}{3}$-EFX towards s_1 as it gets 1 good by the analysis in Case 2. Every agent i is $\frac{2}{3}$-EFX towards s_2 since it gets 2 goods by the analysis in Case 1.
Case 4 ($|\mathcal{C}| = 3, |\mathbf{S}| = 1$) *Deferred to 2.2.1.1*

Case 5 ($|\mathcal{C}| = 4, |\mathbf{S}| = 1$) The goods in $\mathcal{C}$ are given to a source $s \in G_e(X)$. In this case all agents, except s, have a good in $\mathcal{C}$. Fix an agent i, with $|X_i| = 1$ and it's critical good $g_i \in \mathcal{C}$. By the termination of 3PA+, `Step 4` is not executed for i, thus for an arbitrary good $g_i \neq g \in \mathcal{C} \subset Pool(X)$, we have $v_i(\{g_i, g\}) \leq \frac{2}{3}v_i(X_i)$. Finally, as g_i is critical for i, $v_i(g_i) > \frac{1}{2}v_i(X_i)$. This implies $v_i(g) \leq \frac{2}{3}v_i(X_i) - v_i(g_i) \leq \frac{2}{3}v_i(X_i) - \frac{1}{2}v_i(X_i) = \frac{1}{6}v_i(X_i)$.

This implies that

$$v_i(\mathcal{C}) \leq \frac{2}{3}v_i(X_i) + \frac{2}{6}v_i(X_i) = v_i(X_i)$$

Now consider any subset of $\tilde{X}_s \setminus \{g\} \subset X_s$. Clearly, i's envy towards it is maximized if g is not valued highly. We know that there is a good in $\mathcal{C}$ such that $v_i(g) \leq \frac{1}{6}v_i(X_i)$. So at worst we have that:

$$v_i(\tilde{X}_s \setminus \{g\}) = v_i((X_s \cup \mathcal{C}) \setminus \{g\}) \leq \frac{2}{3}v_i(X_i) + \frac{2}{3}v_i(X_i) + \frac{1}{6}v_i(X_i) = \frac{3}{2}v_i(\tilde{X}_i)$$

Thus, every agent i satisfies $\frac{2}{3}-$EFX towards s.

□

2.2.1.1 The One Difficult Case

Consider the case where $|\mathcal{C}| = 3$ and $|S| = 1$. Now $v_i(\mathcal{C}) \leq \frac{3}{2}v_i(X_i)$ for agents i that do not have a critical good in C since $\frac{1}{2}v_i(X_i) \geq v_i(g)$. Moreover, these agents (there are at most 2, since $|C| = 3$) could have arbitrary bundle sizes (1 or 2), and thereby, *i could* have a bundle of size 2 and only be EF towards s in G_e - unlike agents with bundle size 1. Thus agents with no content goods of bundle size 2 are not $\frac{2}{3}$-EFX. Finally, the guarantees from the extra step in 3PA+ do not give us anything either, since giving 3 goods to s leads to a bundle size of 5 does not give us any guarantees regarding size 4 bundles after removing any arbitrary good.

2.2.2 Top-N Instances

We now attempt to apply our techniques for Top-N instances $I = (N, M = T \sqcup B, \{v_i\}_{i \in N})$ to see whether we can achieve a similar approximation for Top-(N-1) instance, that is instances where $|T| = |N| - 1$. Moreover, instances that admit a common Top-N set constitute a more relaxed setting than multi-graph instances, as they impose fewer constraints on agents' valuations over goods. Consequently, studying this case can provide additional insights into the structural conditions under which 3PA successfully produces complete allocations.

Note that we already have an easy algorithm for achieving a $\frac{2}{3}$-EFX approximation amenable partial allocation for Top-N instances. Moreover, this algorithm runs in time linear in $|N|$. However, we make the following novel observations.

Lemma 2.2.2. Consider the case where X satisfies Prop 2.1.3 after termination of Algorithm 3. We have the following:

- For every agent i with $|X_i| = 1$ $X_i = \{t_i\}$ where $t_i \in T$.

  Proof. If not then, $X_i = \{b\}, b \in B$. But $v_i(g) > v_i(b), \forall g \in T \cap Pool(X)$, therefore `Step 1` should run and this contradicts the termination of 3PA. □

- For every agent i with $|X_i| = 2$, $\exists t_i \in X_i$ where $t_i \in T, b_i \in B$.

  Proof. If not then, $X_i \subset B$. But there exists $v_i(t) > v_i(g), \forall t \in T \cap Pool(X), g \in X_i$, therefore `Step 4` should run and this contradicts the termination of Algorithm 3. □

- *All* goods in T are assigned.

  Proof. From above observations we get that $|T| = |N|$ and each for each good $g \in T$, $\exists i \in N$, such that $g \in X_i$. □

- As Condition 1 in Prop 2.1.3 holds for an agent i with $|X_i| = 1$ we have that $v_i(t_i) > v_i(t_j)$ for any agent j with $|X_j| = 2$.
- As Condition 3 in Prop 2.1.3 holds for an agent i with $|X_i| = 2$, it has no critical goods, namely:
$$\forall g \in Pool(X), \frac{1}{2} v_i(X_i) \geq v_i(g)$$
- As Condition 4 in in Prop 2.1.3 holds for an agent i with $|X_i| = 1$, it can have *at most* one critical good.
- $Pool(X) \subset B$, since all goods in T are allocated.

Since all goods in T are allocated, our seed allocation X^* can just be one round of Round-Robin for goods in T.

Note that although the 3PA allocation starting from X^* is quite similar to the one described in Section 2.2.2, it may not be approximation amenable; due to the existence

of at most one critical good for agents with bundle size 1. This implies that $\frac{1}{2}v_i(t_i) < v_i(b), b \in Pool(X), |X_i| = 1$. We can easily construct an example where a particular good in the pool is critical for at least 2 distinct agents and is not already allocated up until Step 5 of Algorithm 3.

Example 2.2.1. Consider the Top-N instance where, for all $i \in N$,

$$v_i(g) = \begin{cases} 1, & \text{if } g \in T \\ \frac{1}{2} + \varepsilon, & \text{if } g = b_m, b_n \text{ for some specific } b_m, b_n \in B \\ 0, & \text{if } g \in B \setminus \{b_m, b_n\} \end{cases}$$

With small $\varepsilon > 0$. Starting from X^*, let $X_i^* = \{t_i\}, \forall i \in N, t_i \in T$.

1. `Step 1` and `Step 2` will not happen starting from X^*. There are no cycles in G_r as performing one round of RoundRobin, can only lead to envy from an only from an agent assigned a good before it.
2. Now $v_i(b_m, b_n) \geq \frac{2}{3}v_i(t_i) = \frac{2}{3}$ will be satisfied for an agent `Step 3` once. It's new bundle will be $\{b_i, b_j\}$ and t_i is added back into the pool. `Step 1-3` will never occur for this agent.
3. If t_i is not envied by any other agents, it (or any other good from T) will eventually be traded back into i's bundle as $v_i(t) > \max(v_i(b_m), v_i(b_n))$ via `Step 4` putting, without loss of generality, b_n back into the pool. i has bundle size 2.
4. From this point onwards `Step 3` will not be called again for agent i leaving a critical good b_n in the pool. This good cannot be put into any other agent's $k \notin i, j$ bundle via `Step 3` as all agents only find b_n, b_i critical and b_i is already in i's bundle. Thereby, `Steps 1-4` won't be called for this instance anymore.
5. `Step 5` only resolves cycles in G_r, so it doesn't add any new goods to the allocation.

The above example illustrates the fact that we can have a good that is critical to almost all agents at least up until Step 5 of Algorithm 3. Our counterexample (as was unfortunately discovered) fails to describe a Top-N counterexample wherein Step 6 doesn't execute at least once. We divulge the shortcomings of our attempt in Appendix C. Nevertheless, we readily have an algorithm for computing Top-N, pushing towards a Top-(N-1) using 3PA seems largely obfuscated.

Chapter 3

An Interesting Connection to Extremal Graph Theory

The goal of this chapter is to give new insights into systematically producing a approximation amenable partial allocation via 3PA that is $\frac{2}{3}$-EFX. Our previous techniques involved ad-hoc modifications to either the algorithm or other subroutines that took advantage specific properties of instances themselves. Moreover, from our previous analysis we have seen that giving away goods that are critical for more than one agent is a highly non-trivial task. With this in mind we try to leverage the theoretical techniques from a similar problem, namely, α-EFX *with Charity*. This is a further relaxation of α-EFX in that we are allowed to actually leave some goods *unallocated*, which are (charitably) given away.

In fact, achieving very high approximations for EFX leaving a sublinear (with respect to the number of agents) number of goods unallocated is feasible in polynomial time.

The techniques here are described in [8]. These techniques are quite robust but complex. We provide detailed exposition where necessary.

As a brief note, a critical good ($\frac{1}{2}$-critical as defined before) g for an agent i is also valuable to i as $\varepsilon \leq \frac{1}{2}$, that is, $v_i(g) > \frac{1}{2}v_i(X_i) > \varepsilon v_i(X_i)$

We now describe some update rules based on analysis of the envy graph $G(X)$ with respect to a partial allocation X.

Lemma 3.0.1 (Update Rule 1 [8])**.** Consider a $(1-\varepsilon)$-EFX allocation X. If there is a source s in $G(X)$ - the envy-graph with respect to X, and a good in the pool $g \in Pool(X)$ such that no agent strongly envies $X_s \cup g$, then if $X' = (X_1, \cdots, X_s \cup \{g\}, \cdots, X_n)$ is a $(1-\varepsilon)$-EFX allocation that *pareto dominates* X.

Lemma 3.0.2 (Update Rule 2 [8])**.** Consider a $(1-\varepsilon)$-EFX allocation X. If there is an agent $i \in N$ such that it heavily envies $Pool(X)$ i.e. $v_i(X_i) < (1-\varepsilon)v_i(Pool(X))$, then we can determine a $(1-\varepsilon)$-EFX allocation X' that *pareto dominates* X and for some agent $j \in N$ we have $(1-\varepsilon)v_j(X'_j) \geq v_j(X_j)$.

Lemma 3.0.3 (Update Rule 3 [8])**.** Consider a $(1-\varepsilon)$-EFX allocation X. If there exist

a set of sources $s_1, \cdots, s_l$ in $G(x)$, a set of unallocated goods $g_1, \cdots, g_l$ and a set of agents $t_1, t_2, \cdots t_l$ such that each t_i is reachable from s_i in $G(X)$ and t_i is the *champion* of $X_{s_j} \cup \{g\}$, then we can determine a $(1-\varepsilon)$-EFX allocation X' that *pareto dominates* X and for some agent $j \in N$ we have $(1-\varepsilon)v_j(X'_j) \geq v_j(X_j)$.

In words, this update rule gives away goods g_i to sources s_i which have a path from t_i and performs a procedure similar to *PathResolution**.

3.1 The Minimum Rainbow Cycle Number Problem

We switch gears and describe a problem in extremal graph theory called *Minimium Rainbow Cycle Number (MRCN)*. MRCN is a problem on a class of directed-graphs called *multi-partite* graphs, $\mathcal{M}_k^d = \{G = (V_1 \sqcup \cdots \sqcup V_k, E), |V_i| \leq d\}$, indexed by a positive integers k and d. Here, each V_i is an independent set, that is no two vertices in V_i are connected.

Example 3.1.1. Note that, for a fixed d, if $G \in \mathcal{M}_k^d$ then $G \in \mathcal{M}_{k'}^d$ for $k' > k$.

In simple terms, MRCN is the task of finding the largest k for $G = (V_1 \sqcup \cdots \sqcup V_k, E) \in \mathcal{D}_k^d$ in which, no simple cycle exists with respect to the multi-graph $H' = (V' = \{V_1, \cdots, V_k\}, E')$, where we consider the parts themselves as vertices, for a specific subset $\mathcal{D}_k^d \subset \mathcal{M}_k^d$ of multi-partite directed graphs. We give a formal description of the MRCN problem and $\mathcal{D}_k^d$ as follows:

Definition 3.1.1 (MRCN)**.** For integers k and d, let $\mathcal{D}_k^d \subset \mathcal{M}_k^d$. $G = (V_1 \sqcup \cdots \sqcup V_k, E)$ is in $\mathcal{D}_k^d$ if an only if:

- G has exactly k parts $V_1, \cdots V_k$.
- Each part has at most d vertices, that is $1 \leq |V_i| \leq d$ for all $i \in [k]$
- For each $\forall v \in V_i, \exists$ unique $u \in V_j, j \neq i,$ s.t. $uv \in E$.
- $\forall i \in [k]$ each vertex $v \in V_i$ has in-degree $k-1$ (implied by above)
- There exists no cycles C in G that visits each part at most once (a simple cycle over the parts) also called a *Rainbow Cycle*.

The MRCN problem is, given a fixed integer d, find the largest k such that $\exists G \in \mathcal{M}_k^d$ such that $G \in \mathcal{D}_k^d$. k is known as the *Rainbow Cycle Number* with respect to d denoted as $R(d)$.

This definition for the MRCN problem can vary depending on whether we are consulting literature in the mathematical field of extremal graph theory or computer science.

3.1.1 Brief Note on Orientability

To form some familiarity with the structure of such graphs, a bi-partite graph is one where we can partition the set of nodes into 2 independent sets.

A multi-partite is graph is one in which we can partition the set of nodes into $k > 2$ independent sets. It is quite easy to reason about *undirected* bi-partite graphs in $\mathcal{M}_2^d$ as properties like 'no odd-cycles' transfer over to any orientation of said bi-partite graph. However, the same is not true over the much larger class of graphs in $\mathcal{M}_k^d$, as we often need additional restrictions on minimum degree, connectivity and reachability to make claims about *any* orientations for general graphs in $\mathcal{M}_k^d$, for all k and d. This motivates why this problem is difficult to solve.

3.1.2 Bounds on R(d)

[8] determined that $R(1) = 1, R(2) = 2$. Moreover, $R(d)$ is shown to be finite for all d. It is first shown that any graph $G \in \mathcal{M}_k^d$ with $4d$ parts in which all parts have the same configuration of edges between them and satisfy the *first 4* rules of membership in $\mathcal{D}_k^d$ then we must have a rainbow cycle.

We then reduce this problem to the problem of finding a monochromatic clique of size $4d$ over a complete graph K_k with $|\mathcal{J}| \in 2^{O(d^2)}$ colors. Here, $\mathcal{J}$ is the set of all possible configurations of edges between any two parts V_i, V_j of size at most d. This is precisely the task of finding the Ramsey Number over $|\mathcal{J}|$ unique colors. In this way,

$$R(d) = k \iff \mathcal{R}(n_1, n_2, \cdots, n_{|\mathcal{J}|}) = k$$

[8] also find a polynomial upper bound $R(d) \leq d^4 + d$, and the most recent upper bound in [1] and [12] found $R(d) \in O(d \log d)$. It is suspected that $R(d)$ scales linearly with respect d, the upper bound on the number of vertices in each part. The proofs are not constructive, in that they do not give a method to determine $R(d)$ given d. In fact, the best we can do is, to perform an exhaustive search over $\mathcal{M}_k^d$ and produce a counter-example to $R(d) > k$ by finding a graph $G \in \mathcal{D}_k^d$. Such experiments have already been done in [12] for small d.

3.2 Connection to EFX Allocations

We now connect MRCN to the problem of finding $\frac{2}{3}$-EFX allocations (fixing $\varepsilon = \frac{1}{3}$).

For the remainder of the section we work with a partial allocation X in which Update Rule 1 (Lem. 3.0.1) and Update Rule 2 (Lem. 3.0.2) *do not* apply and the graph $G(X)$ is acyclic, lets call these set of preconditions **(*)**. This gives us the following facts:

- Some agent strongly envies $X_s \cup \{g\}$ for each $g \in Pool(X)$
- Any agent does not heavily envy $Pool(X)$. That is $\forall i \in N, v_i(X_i) \geq \frac{2}{3} v_i(S)$
- Let S be the set of sources in $G(X)$ then $|S| \geq 1$. Moreover, for every agent i that is not a source there is a path from a source $s \in S$ to i in $G(X)$.

Given an integer d and a partial allocation we first partition $Pool(X)$ into two subsets L_X and H_X such that a good $g \in L_X$ if at most d agents find g valuable, otherwise $g \in H_X$. We provide a simple upper-bound on $|H_X|$.

Proposition 3.2.1. Given **(*)**, $|H_X| < 2n/(\varepsilon \cdot d)$

Proof. For each $g \in H_X$, let n_g be the number of agents that find g valuable. By definition, of membership in H_X, $n_g > d$, which implies $\sum_{g \in H_X} n_g > |H_X|d$.

Now $\sum_{g \in H_X} n_g < n \cdot (2/\varepsilon)$. Since, Update Rule 2, doesn't apply we have that $v_i(X_i) \geq (1-\varepsilon)v_i(Pool(X)) \implies \frac{1}{(1-\varepsilon)} v_i(X_i) \geq v_i(Pool(X))$. Moreover, since $\varepsilon \leq \frac{1}{2}$ we have $2v_i(X_i) \geq v_i(Pool(X))$.

Finally, any valuable good $g \in Pool(X)$ has $v_i(g) > \varepsilon v_i(X_i)$ for agent i. Now, at most all goods in the pool are valuable to i, and thus, $|Pool(X)|\varepsilon v_i(X_i) < 2v_i(X_i)$, thus at most $n_g < \frac{2}{\varepsilon} \implies \sum_{g \in H_X} n_g < \frac{2n}{\varepsilon}$.

Substituting in the above, we have that,

$$|H_X| < \frac{\sum_{g \in H_X} n_g}{d} < \frac{2n}{d\varepsilon}$$

□

The hard part here is bounding $|L_X|$, these are goods that are not valuable to too many agents. For this we consider the *group champion* graph.

This was first introduced as an iteration to *champion graphs* from [7] and adapted for the case of EFX with sublinear charity from [8].

We are given an arbitrary partial allocation X and an integer d. Consider the set of good $g \in L_X$, then it must be the case that at most d agents find this good valuable. Now, let $Q_g \subset N$ be the set of all agents that find g valuable. More concretely,

$$Q_g = \{i \in N : v_i(g) > \varepsilon \cdot v_i(X_i)\}$$

First note that $Q_g \cap Q_{g'}$ may be non-empty, that is, an agent may find 2 goods valuable. This condition does not hold for critical goods (which are also valuable) given the 3PA allocation by property 3 in Proposition 2.1.3. Thus, if a good is critical to an agent, it will uniquely belong to one such set.

Now to each such agent in $a \in Q_g$ we assign a source $s(a)$ in $G(X)$ such that there is path from $s(a)$ to a. Such a source is guaranteed to exist as $G(X)$ is acyclic. If there are multiple paths from unique sources to a given agent, then assign $s(a)$ arbitrarily. It may be the case that the i is a source itself. Thus, we construct

$$V_g = \{(g, s(a)) : a \in Q_g\}$$

First note that $Q_g \cap Q_{g'}$ may be non-empty, that is, an agent may find 2 *unique* goods valuable. This condition does not hold for critical goods (which are also valuable) given the 3PA allocation by property 3 in Proposition 2.1.3. Thus, if a good is critical to an agent, it will uniquely belong to one such set. On the other hand, $V_g \cap V_{g'}$ are disjoint, because $(g, s(a)) \neq (g', s(a))$ even though the assigned sources for a are the same.

This is the vertex set of our group champion graph $V_{champ} = \bigsqcup_{g \in L_X} V_g$. By definition, $|V_g| \leq d$ as $g \in L_X$.

We now construct the edge set as follows.

For any goods $g, h \in L_X$ and agents $a \in Q_g$ and $b \in Q_h$,

$$\text{Edge between } (g, s(a)) \in V_g \text{ and } (h, s(b)) \in V_h \iff a \text{ is a champion of } X_{s(b)} \cup \{g\}$$

From our intuition, this corresponds to whether a will not be $(1-\varepsilon)$-EF and all other agents are $(1-\varepsilon)$-EFX towards $s(b)$ after we give a good a finds valuable to $X_{s(b)}$.

Proposition 3.2.2. Given **(*)**, for any $g, h \in L_X$, each $(h, s(b)) \in V_h$ has an incoming edge from a vertex $(g, s(a)) \in V_g$.

Proof. Since Update Rule 1 doesn't apply, there is an agent that strongly envies $X_{s(b)} \cup \{g\}$. Fix an agent a as the champion of $X_{s(b)} \cup \{g\}$, this must also exist otherwise, we could apply Update Rule 1. Moreover, all agents a that strongly envy $X_{s(b)} \cup \{g\}$ consider g valuable and hence $a \in Q_g$. Thus there is an edge from $(g, s(a)) \in V_g$ to $(h, s(b)) \in V_h$. □

Proposition 3.2.3. Given **(*)**, and given a cycle C in $G_{champ}(X)$ that visits each part at most once where $g \in L_X$, then we can apply Update Rule 3 (Lem. 3.0.3) (See [8] for proof)

Here we have already shown that $G_{champ} \in \mathcal{M}_k^d$. We now connect the problem to MRCN.

Theorem 3.2.4. *Consider a $(1-\varepsilon)$-EFX allocation X. If $|L_X| > R(d)$, there is a $(1-\varepsilon)$-EFX allocation X' that pareto dominates X and for some agent $j \in N$ we have $(1-\varepsilon)v_j(X'_j) \geq v_j(X_j)$. Moreover, $|L_X| \leq R(d) \implies G_{champ}(X) \in D_k^d$.*

3.2.0.1 Implications for 3PA Allocations for 9 Agents

How can we systematically give away goods in $\mathcal{C}$ for a 3PA allocation (see Section **??**)?

For the case of at most 9 agents, we had that $|\mathcal{C}| \leq 4$ and we also have that at least two distinct agents find this good critical, and thereby valuable. Moreover, an agent $a \in N$ with $|X_a| = 1$ that finds a good $g \in \mathcal{C}$ critical is also in V_g, thus $|V_g| > 2$. Note that a the set of sources S_e in $G_e(X)$ is a subset of the sources S in $G(X)$. Now any agent a that has a critical good cannot strongly envy $X_{s(b)} \cup g$ if $s(b)$ is a source in G_e. As $s(b)$ is a source in G_e, we not only have $v_i(X_i) > v_i(X_s)$ but also $v_i(X_i) > \frac{3}{2}v_i(X_s)$. This means, that it can never be a champion for this good.

As a proposed solution to the difficult case in Section 2.2.1, if we are able to satisfy $\frac{2}{3}$-EFX by giving the goods to the single source $s \in S_e$ (including envy from agents with no critical goods in $\mathcal{C}$ with, possibly, size 2 bundles) then do so. Otherwise, we could give 2 critical goods in $\mathcal{C}$ to the single source and resolve the cycles. Now must have an agent t that champions $X_{s'} \cup \{g\}$ for source s' in G'_e, the graph after resolving cycles and $g \in \mathcal{C}$. Thus we can apply Update Rule 3 to give away this final good.

Chapter 4

Experimental Analysis of 3PA and Related Algorithms

In this chapter we explore practical considerations regarding Algorithm 3, we compare it's running time for different distributions and bottlenecks, compare number of critical goods for various distributions of and connection to rainbow cycle numbers in relation to group champion graphs. Moreover, we provide complete implementations of all algorithms discussed so far:

MultigraphAllocate (See Section 2.1.3), *AtMost7Agents* ([4] and Section 2.1.3), *Draft-And-Eliminate* (Section 2.1.2), *Round-Robin* (Algorithm 1) and *Envy-Cycle-Elimination* (Algorithm 2).

In our implementation, we prioritize performance and correctness over API elegance, within reasonable limits. Nevertheless, we ensure that our results are reproducible.

The implementation of algorithms, plotting and the testing suite was done in Python 3.12.2 on a Apple Macbook Air with 8-core Apple M1 @ 3.2GHz, 16GB Memory.

4.1 Practical considerations

4.1.1 Representation of General Fair Division Instance

The representation of the general fair division problem instance was chosen to be as simple as possible. Each instance $I = (N, M, \{v_i\}_{i \in N})$ is represented by a Python class that maintains a set of goods, agents, the $|N| \times |M|$ valuation table, an allocation as well as the pool of unallocated goods represented as a set.

We chose to represent an allocation as a dictionary (`dict`) mapping an agent to a list of goods and the valuation table as a `numpy.array`. Secondly, given n agents and m goods, the set of agents and goods were labeled as $[n] = \{1, \cdots, n\}$ and $[m] = \{1, \cdots, m\}$. This was chosen to both ease indexing the valuation table and to keep the representation of agents and goods lightweight. While we could have used classes to represent agents and goods—allowing for increased polymorphism and the ability to associate more data

with each object—this would have introduced significant overhead. Since these objects are accessed frequently, and Python classes are relatively slow, we opted for simpler representations to maintain performance.

We also introduced some general purpose instance methods associated with the class with the following specifications:

- Given a valuation function v_i for agent i, `get_val(i, g)` returns $v_i(g)$.
- Given a partial allocation X, `bundle_value(i,j)` computes $v_j(X_i)$, the value of agent i's bundle with respect to j's valuation function.
- Functions to assign or remove a good from the pool, which performs updates of the form $X_i \leftarrow X_i * \{g\}$ and $Pool(X) \leftarrow Pool(X) *' \{g\}$ where $*, *'$ is set addition or removal, where appropriate. This function is *only* called for a good in the pool.
- A function `assign_bundle_in_allocation(i, S)` which performs the following update $X_i \leftarrow S$ where $S \subset M$ is an arbitrary bundle consisting of any collection of goods. Note that this may include goods from the pool, hence pool updates must be done separately.
- Given a partial allocation X, `get_max_val_good(pool, i)` computes $\arg\max v_i(g)$ over all goods $g \in Pool(X)$.
- Given a directed graph G, `resolve_all_cycles(G, type)` resolves a cycle and regenerates G until cyclical, returns a graph G of the specified type. `contains_cycles(G)` checks whether G contains any simple cycles.
- Adapted from [4], given a directed graph G, `path_resolution(path)` swaps the bundles backwards along the path from a source to an agent i, and returns the bundle of the source.
- Functions like `get_critical_goods` and `get_valuable_goods` that return a dictionary of goods that are critical/valuable paired with the list of agents that value them.

Alongside the class template for general indivisible-good fair division instances, we decided to keep track of the bundle value of every agent i with respect to it's valuation function, that is $v_i(X_i)$, as well. This allowed us to cache this particular value as `bundle_value(i,i)` was called frequently in the execution of the algorithms and optimized calls to the function, which could now return the stored value. To accommodate this, we had to perform updates to the stored value every time i's bundle was changed. However, in most cases where we are adding or removing one good to/from X_i which results in a constant time update.

We now introduce two important subroutines that, starting from any partial allocation X, output a complete allocation X'. These are required for more complex algorithms.

The first, is the general purpose `round_robin(order, max_iter)`. This subroutine is required as part many complex algorithms, for instance, as part of the Draft-And-Eliminate algorithm and as a subroutine in generating the seed allocation for 3PA. This function allows for completing any partial allocation via Round-Robin for the

remaining goods in the pool. The first parameter `order` specifies the set of agents as well as the order in which we consider them. The second parameter `max_iter` specifies the maximum number of rounds of Round-Robin allowed. The default order in which agents are considered is *lexicographic*, which in our implementation corresponds to the natural increasing order $[1, \cdots, n]$. If `max_iter` is unspecified, then we award goods until the pool is exhausted. Note that, if `max_iter` $= k$, we award at most k goods among the agents given the order. This general implementation of Round-Robin allows us to easily generate a seed allocation for 3PA by setting `max_iter` $= 1$ and also doing a round of Round-Robin for a specific number of agents in reverse ordering, by setting `max_iter = 1,order` $= [n, \cdots k], k > 1$.

Keeping the approximation framework described as a result of Thm 2.1.1 in mind, we require a function to complete any partial allocation X using ECE. Our previous discussion regarding the ECE algorithm elucidated that we may award an arbitrary good to an unenvied agent at each step, however, in our implementation we instead give the most valuable good to the agent. In effect, this decision is inconsequential. The rest of the implementation is quite standard wherein we use the `resolve_all_cycles` function to resolve all cycles in the envy graph and update the envy graph, while awarding a good to a unenvied agent at each step. The discuss how this is done given the way we handle graph data structures.

4.1.1.1 Considerations Due to Floating Point Errors

Inequality comparisons are common in our algorithms, particularly when verifying fairness conditions such as EFX or EF between agents. However, since valuations are generated using floating-point arithmetic (via the `numpy.random` package), direct comparisons such as $v_i(X_i) \geq v_i(X_j)$ can be unreliable due to precision limitations inherent to floating-point representations.

To address this, we use `numpy.isclose()` for equality checks, which allows us to account for small numerical inaccuracies by considering values as equal if they are sufficiently close within a specified tolerance. This method helps mitigate issues caused by floating-point rounding errors.

4.1.2 Graph Constructions

We use the NetworkX [11] package for representing graph data structures used within the algorithms for testing purposes. NetworkX is an ideal choice for the task at hand. It has a lightweight representation of graphs using adjacency lists, out-of-the-box visualization using third-party libraries like MatplotLib, cycle detection algorithms over directed graphs and efficient node/edge queries. Moreover, it has the ability to store arbitrary data alongside nodes and edges.

The functionality described above were integral to producing the various envy graphs (regular, reduced or enhanced) for use in both the 3PA and ECE algorithm, and while testing correctness of the algorithms. Crucially, we leveraged the efficient cycle finding algorithms that NetworkX offers to create a high-performance version of ECE as it is used as a subroutine in almost all advanced algorithms.

Oftentimes, we require a subroutine that resolves all cycles in a given envy-graph. Here we make an observation: since our vertex set is finite, the length of any cycle in an envy-graph must be bounded. There are two cycle-finding algorithms offered by NetworkX for finding bounded-length cycles in a directed graph, `find_cycle` and `simple_cycles`, the former finds a cycle using depth-first-search to find one cycle, while the latter uses an algorithm of Gupta and Suzumura [10] to find *all* simple cycles. We prefer the former. The envy-graphs (regular, reduced or enhanced) are dynamic objects and the valuation functions of agents are uncorrelated. Therefore, resolving any one cycle, that is, changing the allocation, can result in a completely different envy-graph in the next iteration. As an example, not that any agent i involved in a given cycle receives a better bundle, however, any other agent in said cycle that also did not envy i, may do so now. This is why we must regenerate the graph after resolving a cycle; this is a necessary and unavoidable step. It makes sense then, to use the `find-cycle` procedure to resolve any *one* cycle, regenerate the graph. We note that regenerating the envy-graph is the bottleneck procedure here, as locating a cycles and then swapping bundles is local to agents present in the cycle barring the cost of copying the bundle. On the other hand, the regeneration procedure requires use to consider the envy-between pairs of agents not present within the cycle as well while modifying the graph structure.

Finally, the visualization capabilities of NetworkX bindings to MatplotLib allowed for careful debugging as individual graphs at each step could be generated on-the-fly and stored for visualization later. Secondly, multi-partite champion graphs.

4.1.3 Static Dataset from *Spliddit*

We also use real life data collected from an archived version of the *Spliddit.org* website[9] - an online implementation of fair division algorithms, now defunct. The dataset contains *2309* real-life instances of fair-division of indivisible items of varying instance sizes. Similar to our setting participants were asked to choose their valuation on goods from a range 1 to 1000, however, their valuations were 'budgeted', that is $\forall i \in N, \sum_{g \in M} v_i(g) = 1000$.

4.1.4 Distribution Drawn Valuation Functions over Fixed Agents

We now discuss the various ways in which random valuation functions were generated for testing purposes. The general strategy was to draw values from distributions. We distinguish between *general* instances with random valuation functions and rando valuation functions with extra invariants like *Multi-graph* or *Top-N*.

We would like to run our algorithms to run on all types of random instances, where possible.

As a semantic note, we use the notation $v_i(g) \iff (V_{\mathcal{D}})_{ig}$ to interchangeably refer to the valuation functions and as entries in the valuation matrix $V_{\mathcal{D}}$ sampled from distribution $\mathcal{D}$.

We use the `numpy.random` package to generate the valuation tables, supporting the following distributions:

- Uniform
- U-Quadratic (Concave - Face-down)
- U-Quadratic (Convex - Face-up)
- Uniform Binary
- Uniform Homogeneous
- Normal distribution with specified parameters $\mu = 500$ and $\sigma = 250$

Except for the binary distribution, we fix the value range to $r = [1, 1000]$. This ensures that, regardless of the chosen distribution, all valuations fall within the same bounded interval. We used the the `numpy.random` package draw random matrices of arbitrary sizes directly and allows us to set a threshold for the minimum and maximum values drawn from a distribution. This method works for all but the U-Quadratic distributions.

The U-Quadratic distributions (concave/convex) are specified given the minimum and maximum bounds, α, β. Given a support of $[a,b]$ we define $\forall x \in [a,b]$ the probability distribution function (pdf) f for the *convex (face-up)* U-Quadratic as:

$$f(x) = \alpha(x-\beta)^2, \alpha = \frac{12}{(b-a)^3}, \beta = \frac{b+a}{2}$$

We can then sample values with replacement to get the required distribution for the valuation function.

For the *concave* variant, we take the additive inverse of the pdf then shift and normalize ensuring that, values are positive and satisfy $\sum_x f(x) = 1$, thus describing a valid pdf.

4.1.5 Multi-graph Instances

We now consider the task of generating random multi-graph instances $I = (N, M, \{v_i\}_{i \in N})$, This is done via a undirected labeled graph $H = (V, E)$ wherein $V \sim N$ and edges E are labeled via a bijection to M such that $\forall g \in M, \forall i \in N, v_i(g) \neq 0 \iff g$ is the label of an edge incident to the vertex labeled as i.

Sine we require that the edges in our graph are labeled in bijection to M - the set of goods; we can have at most $|M|$ edges in the graph. Secondly, we note that *at most* two agents can have positive value for a particular good, since each undirected edge can have at most two unique vertices incident to it. Therefore, our graph can have two types of edges (i, j). If $i \neq j$ then this is a regular edge between two unique vertices, and $i = j$ is a self-loop and only i may have a value for this good. Finally, we allow for parallel edges, namely, we may have two edges (i, j) labeled by two different goods $g, h \in M$ respectively. Thus, the description of random multi-graph instances can be fully captured by generating an undirected random graph with $|M|$ edges over $|N|$ nodes with parallel edges and self-loops allowed, such a graph is called a multi-graph.

The method we use to generate the random multi-graphs corresponds roughly to the $G(n,m)$ framework for generating random simple graphs according to the *Erdös-Renyi*

model [**?**] given a fixed vertex set of size $n = |N|$. There are two standard ways of generating random graphs as prescribed by the model. The first is the $G(n,p)$ model which independently and identically tosses a biased coin with probability p for edge included in the graph between any pair of two unique vertices. On the other hand, the $G(n,m)$ model samples a simple graph uniformly at random from the collection of all graphs with m edges over n vertices. The key difference in between these two methods is that in the former ($G(n,p)$), the number of edges m behaves as a binomial random variable sampled from $\text{Bin}(\binom{n}{2}, p)$. Obtaining a graph with exactly m edges requires rejection sampling, and the number of trials until we get a graph with exactly m edges behaves as a geometric random variable with vanishing success probability as n grows. This is undesired behavior as we need precise control over the number of edges in order to study properties of the resulting graph conditioned on fixed edge cardinality.

Here there is a caveat, the two models described above are suited to generate random *simple* graphs, that is $m \leq \binom{n}{2}$ and edge endpoints are unique. There are several implementations of the $G(n,m)$ model, particularly, those offered by the `NetworkX` library, however, we cannot use these to generate multi-graphs. Instead, we adopt a similar approach to the `NetworkX` implementation for simple graphs; we simply sample $|M|$ pairs at once from the set of all pairs of vertices, however here, endpoints are chosen *with* replacement. We add these edges to a `NetworkX.multi-graph`. This allows us to model multi-graphs, where multiple edges between the same pair of vertices are permitted.

We then generate, given a distribution $\mathcal{D}$, a fully populated valuation table $K_{\mathcal{D}}$ and copy over the values where allowed as stipulated by the multi-graph H in the final valuation table $V_{\mathcal{D}}$, that is $\forall g \in M, i \in N, V_{\mathcal{D}}(i,g) = K_{\mathcal{D}}(i,g) > 0 \iff g$ is the label of an edge incident to the vertex labeled as i in H and $V(i,g) = 0$ otherwise.

4.1.6 Top N Instance

To generate *Top-N* instances $I = (N, M = T \sqcup B, \{v_i\}_{i \in N})$, we proceed as follows. We begin by selecting a subset $T \subseteq M$ of $|N|$ goods uniformly at random — this will be the *Top-N* set. For each agent $i \in N$, we then generate valuations over the goods in T using a distribution $\mathcal{D}$, resulting in a $|N| \times |N|$ valuation matrix $T_{\mathcal{D}}$.

We define a threshold

$$k := \min_{i,j} (T_{\mathcal{D}})_{ij},$$

which captures the smallest value assigned to any top good by any agent.

Next, we generate valuations for the remaining $|M| - |N|$ goods (denoted $B = M \setminus T$) using a distribution $\mathcal{D}'$, identical in type to $\mathcal{D}$ but adjusted to produce values in the interval $[1, k)$. This yields a $|N| \times (|M| - |N|)$ matrix $B_{\mathcal{D}'}$.

Finally, we concatenate $T_{\mathcal{D}}$ and $B_{\mathcal{D}'}$ horizontally to form the complete valuation matrix, ordering columns lexicographically by the natural ordering of good labels.

This ensures that agents find some goods critical at some point. This is because we have chosen k such that values in B are drawn as close to valuations of T as possible, thereby

creating competition for goods in B. This, in turn, leads to higher likelihood of goods in B being *critical*.

4.1.7 Construction of Group Champion Graph Given a 3PA Allocation

Given a 3PA allocation we can generate the Group Champion graph as described in [8] for analysis. This requires subroutines that find all sources in the envy-graph, finding the list and number of agents that find a particular good valuable. For all intents and purposes we fix $\varepsilon = \frac{1}{3}$ since 3PA aims to construct a complete $1-\varepsilon = \frac{2}{3}$-EFX allocation.

Finally, we require a function for checking for championship between agents. We follow the definition exactly and consider the case where X is a 3PA allocation.

To construct the group champion graph we must check championship between an agent a a source s and a good g such that $(g, s(a)) \in V_g$. Recalling the definition, we say that an agent a is a *champion* for agent $a \neq s$ w.r.t g if there is a set $S \subset X_s \cup \{g\}$ such that:

1. To verify whether agent a envies another agent s after a new good g is added to s's bundle, we only need to check size-2 subsets of $X_s \cup \{g\}$ that include g. Since a was already $\frac{2}{3}$-EFX towards s, and X_s had size 2, a did not envy any single good in X_s, and doesn't envy X_s as s is a source. Thus, the only potential for new envy comes from subsets involving the newly added good. Larger subsets do not need to be checked if smaller ones already satisfy the fairness guarantee.
2. For all $k \in N$ (which includes i and j) $(1-\varepsilon)v_k(S \setminus \{h\}) \leq v_k(X_k)$ for all $h \in S$. This step is done iteratively removing a single good and checking for envy.

4.2 Description of Experiments and Results

We fix the following parameters for Experiments 1-3, with modifications stated explicitly if applicable.

Random Valuations	**Number of Instances**	1,000 per distribution.
	Agents per Instance	Number of agents drawn uniformly at random from $N \in \{3, \cdots, 25\}$.
	Goods per Instance	Number of goods drawn uniformly at random from $\{N+1, \cdots, 100\}$.
	Valuation Range	$r = [1, 1000]$
	Distributions Tested	Uniform, Uniform Homogeneous, Quadratic Face-Up, Quadratic Face-Down, Normal, Binary.
	ε	Fix $\varepsilon = \frac{1}{3}$ when discussing valuable goods.

Table 4.1: Common Experimental Setup for Experiments 1–3

The reason for choosing the number of goods is $|N| = |M|$ is a trivial allocation and we are interested in the behavior when there are at least some goods left in the pool.

A more detailed exposition regarding the experimental setup is found in Appendix B.

4.2.1 Experiment 1: Runtime Comparison for Complete Allocations

In this experiment, we compare the average execution times of algorithms that produce complete allocations either via Algorithm 3 or the Top-N algorithm. We focus on three families of instances: multi-graph valuations, Top-N and instances with at most seven agents. For Top-N instances, we compare the simple algorithm (described in Section 2.2.2). We compare these against a set of benchmark algorithms.

The benchmark algorithms considered are Draft-and-Eliminate, Envy-Cycle Elimination (ECE), and Round-Robin. These algorithms are selected for their general applicability to arbitrary instances, despite offering weaker fairness guarantees across distributions.

We hypothesize that the average execution times for all algorithms—excluding Round-Robin—will be *largely comparable*, as the dominant computational bottleneck arises from the ECE subroutine. This is due to the repeated reconstruction of the envy graph at each step of the ECE process, which incurs significant computational overhead.

Moreover, in instances where the ratio of goods to agents is high, we observe an increase in ECE runtime. This is expected, as approximation amenable partial allocations, constitute a small fraction of the total goods; the remaining unassigned goods must be processed via ECE.

4.2.1.1 Observations from Experiment 1

Overall the average runtime for Draft-And-Eliminate, Top-N, Multi-graph-Allocate, and AtMost7Agents was largely the same as predicted.

Consequently, we analyze whether the structure of the valuation distribution—specifically, whether it is of the multi-graph or Top-N type—induces specific patterns in average execution time, and whether algorithmic performance diverges in comparison to the benchmarks. Since the execution time trends were qualitatively similar across all valuation distributions and instance types, we report results for the normal distribution in the main text and defer results for other distributions to Appendix E.

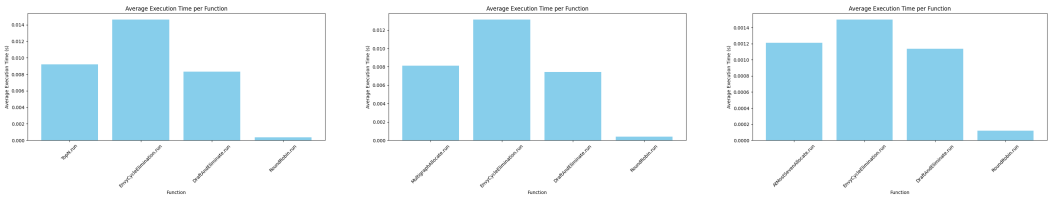


Figure 4.1: Average Runtime for Normal Distribution for all random instance types (left to right: Top-N, Multi-graph, General)

For the static *spliddit* dataset we filter for instances with at most 7 agents and analyze average execution time across those.

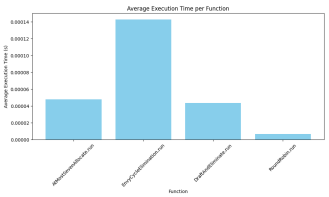


Figure 4.2: Average Runtime Across *Spliddit* dataset

4.2.2 Experiment 2: Runtime and Frequency Breakdown for 3PA

In this experiment, we investigate the computational behavior of the Algorithm 3.

We perform the following analyses on the various distributions along with the *Spliddit* dataset:

1. *Runtime Analysis*: We measure the distribution of average time spent in each subroutine that checks the conditions for the 8 steps.
2. *Frequency Breakdown*: We also track *how often* these condition checks are triggered during the execution of the algorithms across the instances.

The goal of this experiment is to identify bottlenecks or stages of 3PA that contribute most significantly to the total runtime, as well as to examine the relationship between the frequency of checks and the various random types and distributions. We highlight particular distributions with interesting properties across the three instance types in Figure 4.3.

To interpret the figure, note that if a given step i is checked often, this means that all steps $i-1$ were also checked before it. The execution time may or may not, be correlated as checks may return quickly if a specific precondition is readily not satisfied, this is the case for Step 7 which involves a complex checking of preconditions while examining the envy graph.

If a step involves having to iterate through pairs of goods in the pool at every step (Step 4) it will take longer to execute than a step checking for sources or cycles (Step 5,6,8), which are (almost) linear time operations.

4.2.2.1 Observations from Experiment 2

As we can see in Figure 4.3, across all distributions and instance types Step 3 and Step 4 are checked quite frequently.

However, more curiously, across Top-N instances we see that most of the execution time as well as the frequency is spent in either Step 1, Step 3 or Step 4. Given our discussion regarding Top-N instances, this alludes to the fact that each agent who wants to exchange its bundle $X_i = \{t_i\}$ to $X_i' = \{t_j, b_i\}, t_i, t_j \in T, b_i \in B$ for a top-N set T, needs to first execute Step 3, to get a bundle of two goods in B. Then, any agent j that envied t_i exchanges it's top good t_j through Step 1 for t_i and then agent i swaps t_j for one of the goods from B in it's bundle. This trend is highlighted clearly across all distribution types.

The overall execution time is also consistently lower for Top-N instances compared to the other instances (see Figure 4.3). Thereby, Top-N instances possibly provides more structure to the problem resulting in generally fewer swaps and checks.

Step 7 is executed exceedingly rarely across all instance types (see Figure 4.3). Moreover, strictly more steps were checked for the normal and quadratic face up distributions.

We can also give an estimate of the ”efficiency” by considering how much time is spent on each execution of a step, relative to how often that step is called. Lower values mean more efficient (excluding Step 7 which was not executed at all).

Step	step1	step2	step3	step4	step5	step6	step7	step8
Time Per Frequency	4.69e-06	0.0073	0.00024	3.65e-06	0.00399	1.52e-06	0.00	7.80e-05

Table 4.2: Efficiency of each step (average frequency divided by average time per call)

4.2.3 Experiment 3: Ratio of Critical/Valuable Goods and Number of Sources in Enhanced Envy Graph

We collect data on the ratio of critical or valuable goods assigned to each source agent in the enhanced envy graph after obtaining a 3PA allocation. We produce a heatmap scaling with respect to the size of instances. The goal is to provide insights into the structural complexity of the allocation problem after Algorithm 3 across various instance types, enabling us to understand how different value distributions influence the allocation process and its associated fairness properties.

What we expect to see is that the ratio of valuable/critical good to sources is skewed towards instances with a high number of goods and few agents - skewed towards top right of the heatmap. This is an obvious by product of having a large pool and few agents to give these goods to. 3PA allocation satisfies Proposition 2.1.3 - each agent may have at most one critical good, and thereby, we may have at most $|N|$ critical goods if any are present. However, we have no such restrictions on valuable goods.

More notably, keeping in mind the results from the previous experiment, agents were frequently incentivize to switch to bundles of size 2 (via step 3 and step 4) - thereby having no critical goods.

4.2.3.1 Observations from Experiment 3

The occurrence of even a single critical good was exceedingly rare across all distributions (see Figure 4.4).

For random Top-N instances, we get no critical goods across the board and even valuable goods are generally rarer.

For all but few distributions we get no critical goods (Orange implies no critical goods) in Figure 4.4. However, for certain distributions uniformly homogeneous, normal we do get critical goods for multi-graph instances. For uniform homogeneous, and somewhat

for quadratically distributed, normally distributed instances, the distribution of valuable goods strays from the top-right as we would expect.

4.2.4 Experiment 4 : Generating Group Champion Graphs

In this experiment we generate the group champion graphs as described in Section 3.2, and analyze the number of edges and vertices. To do this we examine the case where $d \leq 4$, that is each good is valuable to at most 4 agents. Usually, it is useful to examine these graphs when Update Rule 1 and Update Rule 2 do not apply, since we get guarantees about connectivity. However, it is equally useful to see what guarantees Algorithm 3 will provide given the pool. Thus, we consider the case where there are at least 10 goods left in the pool for each instance. For more details regarding the experimental setup see Appendix D.

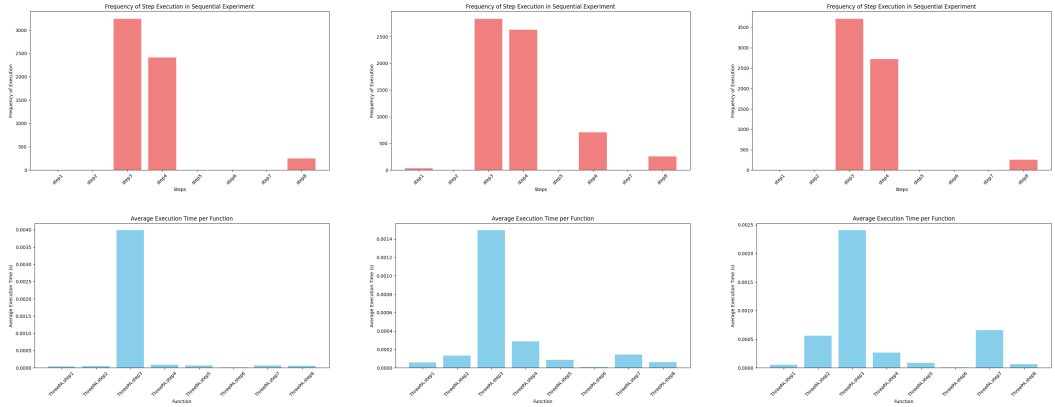
4.2.4.1 Observations from Experiment 4

Group champion graphs produced by random valuation tables were extremely sparse (almost no edges). In this case, we utilize the real-life *Spliddit* data. In Figure 4.5, vertices are denoted as $(g, s(a))$ for good g and the assigned source of agent a. Vertices with the same g belong to the same *part*.

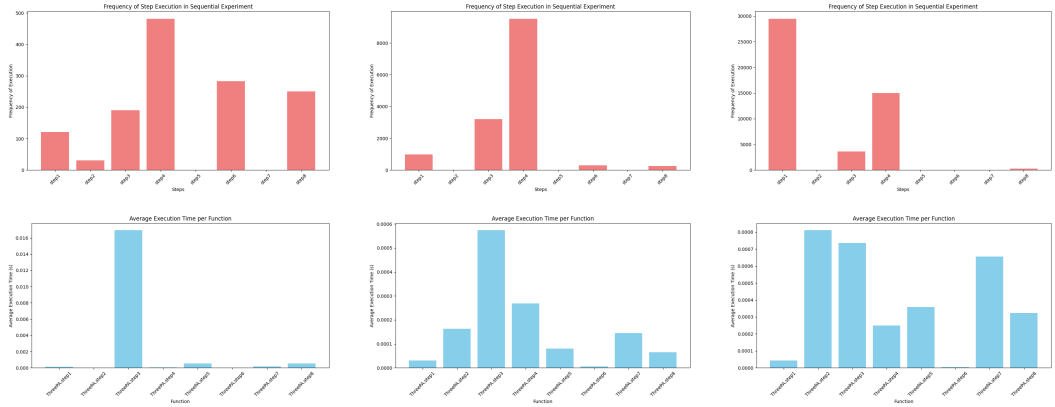
We observed that out of the dense group champion any two vertices with an edge between them had the same source.

We were able to get group champion graphs with a cycle of length 2 between them (row 2 of Figure 4.5), thus we can apply Theorem 3.2.4 to resolve the cycle and produce a $\frac{2}{3}$-EFX allocation.

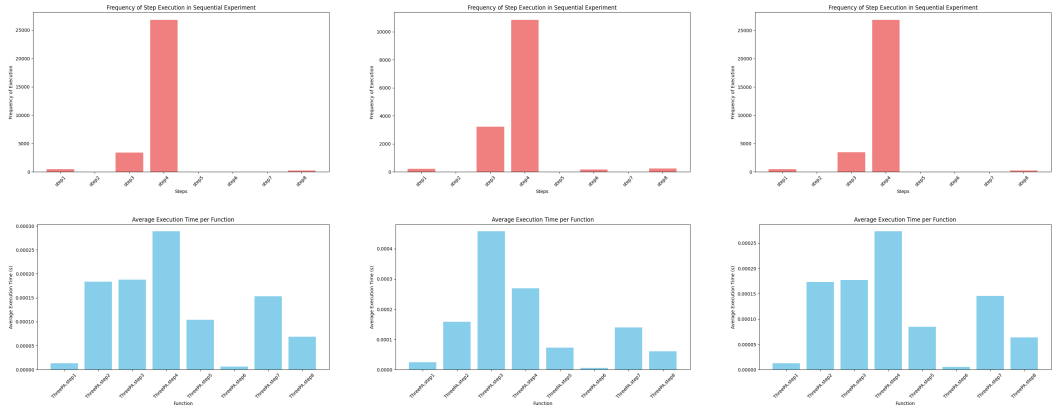
As gleaned from our discussion regarding critical/valuable goods in the previous experiment, Top-N instances saw the fewest valuable/critical goods. For these particular instances all graphs were sparse and disconnected; that is, not very meaningful.



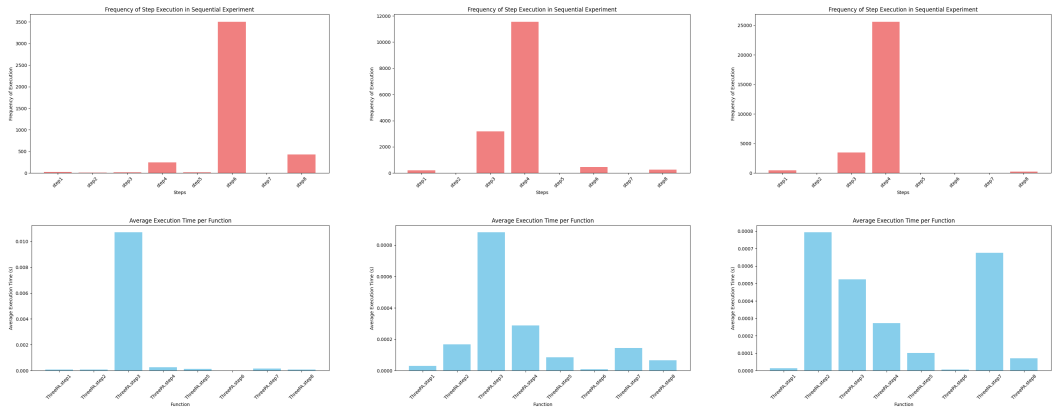
(a) Binary Distribution



(b) Uniform Homogenous Distribution

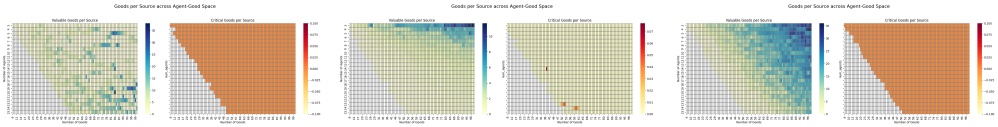


(c) Normal Distribution

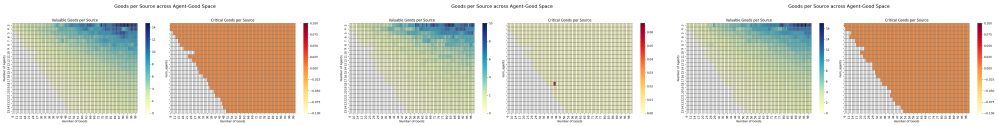


(d) Quadratic Face-Up Distribution

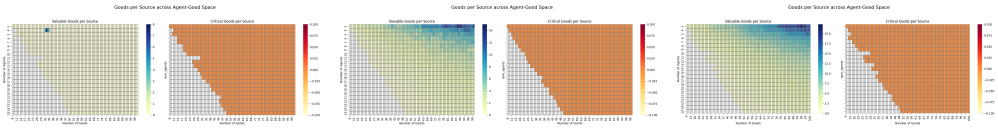
Figure 4.3: Frequencies (Top row) and Runtimes (Bottom row) of the Algorithm 3 Across Different Distributions and Types (Top-N, Multigraph, General)



(a) Uniform Homogenous Distribution



(b) Normal Distribution



(c) Quadratic Face-Down Distribution

Figure 4.4: Average Number of Valuable/Critical Goods per Source Across Different Distributions and Graph Classes

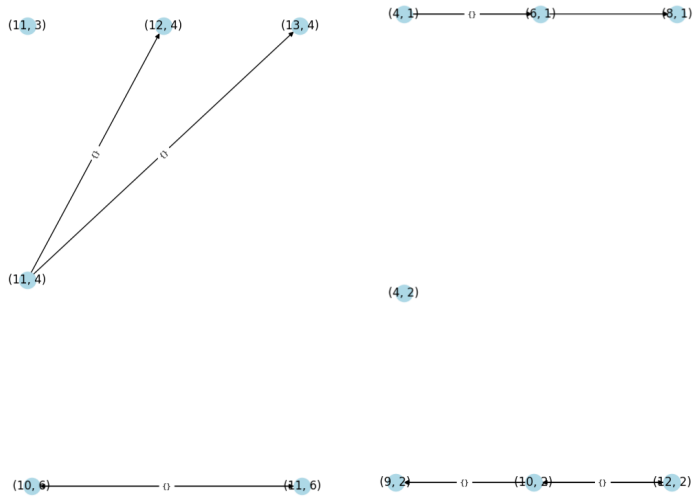


Figure 4.5: Group Champion Graphs Acquired from *Spliddit* Dataset in Experiment 4

Chapter 5

Conclusion

In this work, we investigated the structure and limitations of achieving $\frac{2}{3}$-EFX allocations, with a particular focus on the applicability and implications of the 3PA algorithm. From a theoretical perspective, our exploration revealed that while 3PA is a powerful tool in restricted settings, its adaptability to arbitrary instances appears limited. In particular, increasing the number of agents from 7 to 9—while seemingly minor—leads to intricate edge cases that are difficult to resolve without significant combinatorial machinery. This reflects the fragility of ad hoc extensions in the absence of deeper structural guarantees.

To complement our theoretical analysis, we introduced and adapted *Group Champion Graphs* to the context of 3PA allocations. These structures, along with the accompanying visualization tools developed throughout this work, offer a powerful and intuitive framework for analyzing envy and identifying structural patterns in allocations. We believe this toolkit can serve as a valuable resource for researchers looking to study fairness through more complex graph structures.

We also turned our attention to Top-N instances, which represent a more relaxed class of valuation profiles compared to multi-graph instances. Although allocating critical goods in $\mathcal{C}$ remains challenging even in this setting, we found that certain observations and structural simplifications could still be made. Notably, in all Top-N instances tested, critical goods were entirely absent, and valuable goods were also exceptionally rare. This suggests the possibility of an inherent structural property in Top-N instances that suppresses the emergence of such goods.

Further, we described the relationship between the set of goods $\mathcal{C}$ — those critical to multiple agents—and the set L_X — goods valued by a bounded number of agents (see Section 3.2). Given the known bounds on $R(d)$ and its tight connection to goods that are not valuable to too many agents, an open question arises: can we leverage this connection to systematically reallocate goods in $\mathcal{C}$? While we do not yet have a conclusive theoretical answer, the tools and intuition developed here provide a starting point for future work in this direction.

From a practical standpoint, we observed that 3PA can be computationally expensive, and the cost of its checks varies with the distribution of valuation functions. Moreover,

random valuation functions, while convenient for testing, rarely yield interesting cases involving critical goods. This leads us to advocate for an *adversarial* testing approach, where carefully constructed instances are designed to stress-test the algorithm's behavior. To the best of our knowledge, this direction remains largely unexplored and presents a promising area for further empirical work.

Given the consistent absence of critical goods in our Top-N experiments and the difficulty in producing valid counterexamples where 3PA fails to allocate such goods, we are led to a natural conjecture: for arbitrary Top-N instances, the size of the set $\mathcal{C}$ may be bounded—potentially even logarithmic in the number of agents for arbitrary number of goods. Proving such a result could have significant consequences for our understanding of partial envy-freeness under structured valuations.

Finally, EFX with charity, a problem tightly intertwined with the Minimum Rainbow Cycle Number (MRCN) improving the best known bound of $R(d) \in O(d \log d)$ could unlock new pathways for analyzing and achieving EFX under relaxed allocation models.

Bibliography

[1] Hannaneh Akrami, Noga Alon, Bhaskar Ray Chaudhury, Jugal Garg, Kurt Mehlhorn, and Ruta Mehta. EFX: A Simpler Approach and an (Almost) Optimal Guarantee via Rainbow Cycle Number. In *Proceedings of the 24th ACM Conference on Economics and Computation*, EC '23, page 61, New York, NY, USA, July 2023. Association for Computing Machinery.

[2] Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A. Voudouris, and Xiaowei Wu. Fair Division of Indivisible Goods: Recent Progress and Open Questions. *Artificial Intelligence*, 322:103965, September 2023. arXiv:2208.08782 [cs].

[3] Georgios Amanatidis, Georgios Birmpas, Aris Filos-Ratsikas, Alexandros Hollender, and Alexandros A. Voudouris. Maximum Nash Welfare and Other Stories About EFX. *Theoretical Computer Science*, 863:69–85, April 2021. arXiv:2001.09838 [cs].

[4] Georgios Amanatidis, Aris Filos-Ratsikas, and Alkmini Sgouritsa. Pushing the Frontier on Approximate EFX Allocations, June 2024. arXiv:2406.12413 [cs].

[5] Georgios Amanatidis, Apostolos Ntokos, and Evangelos Markakis. Multiple Birds with One Stone: Beating $1/2$ for EFX and GMMS via Envy Cycle Elimination, March 2021. arXiv:1909.07650.

[6] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The Unreasonable Fairness of Maximum Nash Welfare. *ACM Trans. Econ. Comput.*, 7(3):12:1–12:32, September 2019.

[7] 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, pages 1–19, New York, NY, USA, July 2020. Association for Computing Machinery.

[8] Bhaskar Ray Chaudhury, Jugal Garg, Kurt Mehlhorn, Ruta Mehta, and Pranabendu Misra. Improving EFX Guarantees through Rainbow Cycle Number, March 2021. arXiv:2103.01628 [cs].

[9] Jonathan Goldman and Ariel D. Procaccia. Spliddit: unleashing fair division algorithms. *SIGecom Exch.*, 13(2):41–46, January 2015.

[10] Anshul Gupta and Toyotaro Suzumura. Finding All Bounded-Length Simple Cycles in a Directed Graph, May 2021. arXiv:2105.10094 [cs].

[11] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th python in science conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[12] Shayan Chashm Jahan, Masoud Seddighin, Seyed-Mohammad Seyed-Javadi, and Mohammad Sharifi. Rainbow Cycle Number and EFX Allocations: (Almost) Closing the Gap, July 2023. arXiv:2212.09482 [cs].

[13] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM conference on Electronic commerce*, EC '04, pages 125–131, New York, NY, USA, May 2004. Association for Computing Machinery.

[14] Evangelos Markakis and Christodoulos Santorinaios. Improved EFX Approximation Guarantees under Ordinal-based Assumptions, August 2023. arXiv:2308.04860.

[15] Benjamin Plaut and Tim Roughgarden. Almost Envy-Freeness with General Valuations, November 2017. arXiv:1707.04769 [cs].

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

Appendix A

Pseudo-code for Draft-And-Eliminate and Related Subroutines

Algorithm 6 Preprocessing(I)

Input: A general instance $I = (N, M, \{v_i\}_{i \in N})$
Output: An ordering of agents for 1 round of Round-Robin and size of *non-content* agent set.

$L = \emptyset; A = N; k = 1; P = Pool(\cdot); \mathcal{L} = []$
A is ordered lexicographically.
while $A \neq \emptyset$ **do**
    Let i be the lexicographically first agent of A
    $h_i = \arg\max_{g \in M} v_i(g)$
    $t_i = |M| - |P| + 1$
    Let $R = (N \setminus (A \cup L)) \cup \{i\}$
    $j = \arg\max_{t \in R} v_i(h_t)$
    if $\phi \cdot v_i(h_i) < v_i(h_j)$ **then**
        $h_i = h_j$
        $L = L \cup \{i\}; \mathcal{L}[k] = i$
        $k = k + 1$
        $A = (A \setminus \{i\}) \cup \{j\}$
    else
        $A_i \setminus \{i\}$ (content)
        $P = P \setminus \{h_i\}$
    end if
end while
for $i \in N \setminus L$ in order of increasing timestamp t_i **do**
    $\mathcal{L}[k] = i$
    $k = k + 1$
end for
return $\mathcal{L}, |L|$

Algorithm 7 Draft-And-Eliminate(I)

Input: A general allocation $I = (N, M, \{v_i\}_{i \in N})$
Output: A $\phi - 1$-EFX allocation.

$\mathcal{L}, n' =$ Preprocessing(I)
A is ordered lexicographically.
Let $X_i = \emptyset$ for each agent i.
$(X, \mathcal{M}') =$ Round-Robin$(N, M, \mathcal{L}, |N|)$
$\mathcal{L}^R = (\mathcal{L}[n], \mathcal{L}[n], \cdots, \mathcal{L}[1])$
$(X, \mathcal{M}') =$ Round-Robin$(N, M', \mathcal{L}, |N| - n')$
$X' =$ Envy-Cycle-Elimination$((N, M', \{v_i\}_{i \in N}, X)$
return X'.

Appendix B

Pseudo-code and Proof of Correctness for Simple Top-N Algorithm in [14]

Algorithm 8 2/3 EFX for identical Top-N set

Input: $I = (N, M, \{v_i\}_{i \in N})$ with $M = T \sqcup B$, where T is the set of n most valuable items and B consists of remaining $|M| - |N|$ items.

Output: 2/3 EFX partial $\mathcal{S}$

for each agent $i \in N$ **do**
  $h_i = \arg\max_{m \in T} v_i(m)$
  $g_{1,i} = \arg\min_{m \in T} v_i(m)$
  $g_{2,i} = \arg\max_{m \in B} v_i(m)$
  if $v_i(g_{1,i}) + v_i(g_{2,i}) \geq \frac{2}{3} \cdot v_i(h_i)$ **then**
    $A_i = \{g_{2,i}\}$ (non-content)
    $B = B \setminus g_{2,i}$
  else
    $A_i = \{h_i\}$ (content)
    $T = T \setminus \{h_i\}$
  end if
end for
For every non-content agent, allocate to her one item arbitrarily from T
Continue with running ECE algorithm

We cover the proof of this algorithm's correctness.

Proposition B.0.1. For all instances $I = (N, M, \{v_i\}_{i \in N})$ with a *common top-N* set, the above algorithm computes a $\frac{2}{3}$-EFX allocation.

Proof. We check the conditions for applying the approximation framework, namely, We prove that the partial allocation X created before running ECE is $\frac{2}{3} = \alpha$-EFX and that all remaining goods in the pool are such that $\forall i \in N, v_i(X_i) > 2v_i(g), g \in Pool(X)$. Let $M = T \sqcup B$. Recall, that all goods in T are assigned by the algorithm, and thus $Pool(X) \subseteq B$. Fix an agent i.

If agent i is content, it receives it's most valuable good available h from T such that $\frac{2}{3}v_i(h) > v_i(g_{1,i}) + v_i(g_{2,i})$ holds. Now, agent i is trivially $\frac{2}{3}$-EFX towards any other (content) agent with one good and any agent is also trivially $\frac{2}{3}$-EFX towards agent i. Thereby, suppose there is an agent j with two goods. Let $X_j = \{t_j, b_j\}$ where $t_j \in T$ and $b_j \in B$. Now $v_i(h) > v_i(b_j)$ as $b_j \in B$ and $v_i(h) > v_i(t_j)$ as t_j was available to i when it chose h. Thus i is EFX towards j's bundle and therfore it is $\frac{2}{3}$-EFX as well. Moreover, for a good g in the pool,

$$\frac{2}{3}v_i(h) > v_i(g_{1,i}) + v_i(g_{2,i}) \implies \frac{2}{3}v_i(h) > 2v_i(g_{2,i}) \implies v_i(h) > 3v_i(g_{2,i}) > 3v_i(g)$$

If agent i is not content, it receives two goods. We focus on the case where an agent j receives two goods. Now, we know that $v_i(g_{1,i}) + v_i(g_{2,i}) \geq \frac{2}{3} \cdot v_i(h_i)$. The bundle $\{g_{1,i}, g_{2,i}\}$ - which consists of the worst and best available good from T and B respectively - is the worst possible bundle i can get given a fixed ordering of agents.
Now it must be the case that the $v_i(h) > v_i(g)$ for any $g \in X_j$. Therefore,

$$v_i(X_i) \geq v_i(g_{1,i}) + v_i(g_{2,i}) > \frac{2}{3}v_i(h) > \frac{2}{3}v_i(X_j \setminus g)$$

Where the second inequality follows from i not being content. Thus i is $\frac{2}{3}$-EFX. Finally, any good left in the pool must at least be worse then $g_{2,i}$ - the *best* good available. Therefore,

$$\forall g \in \mathit{Pool}(X), v_i(X_i) > 2v_i(g_{2,i}) > 2v_i(g)$$

□

Appendix C

Top-N Counterexample Attempt

Example C.0.1. Consider the Top-N instance where, for all $i \in N$,

$$v_i(g) = \begin{cases} 1-\varepsilon, & \text{if } g \in T \text{ and } i = j \\ 1, & \text{if } g \in T \text{ and } i \neq j \\ \frac{1}{2}+\varepsilon, & \text{if } g = b_m, b_n \text{ for some specific } b_m, b_n \in B \text{ and } i \neq j \\ 0, & \text{if } g \in B \setminus \{b_m, b_n\} \text{ and } i \neq j \\ 0, & \text{if } g \in \{b_m, b_n\} \text{ and } i = j \\ \varepsilon/2 & \text{if } g \in B \setminus \{b_m, b_n\} \text{ and } i = j \end{cases}$$

Our analysis is largely the same up until Step 5 in Section 2.2.2, it was assumed that agent j (a source in G_r) receives a good through Step 6. However, before this, after any agent $i \neq j$ relinquishes it's (to switch to a bundle of size 2) good t_i from T that j envies, it will trade t_j with t_i in Step 1.

Even if Step 6 is executed and j envies all other agents, j will still be $\frac{2}{3}$-EFX towards i and it's edges towards i will be removed from G_r in the next iteration.

Many iterations were tried, but almost all attempts at coming up with a valid counterexample lead to multiple executions of Step 6 or Step 8 for many agents.

Appendix D

Experimental Setup

Aspect	Description
Objective	Compare execution times of algorithms capable of producing complete allocations.
Algorithms Tested	3PA and Top-N.
Instance Classes	• Multi-graph instances • Instances ≤ 7 agents (Agent per instance drawn from $\{1, \cdots, 7\}$ instead) • Top-N instances
Baseline Algorithms	Draft-And-Eliminate, Envy-Cycle Elimination,Round-Robin
Evaluation Metric	Wall-clock runtime.

Table D.1: Summary of Experiment 1 — Runtime Comparison for Complete Allocations (See Table 4.1 for shared parameters)

Aspect	**Description**
Objective	Compare execution times and frequency of checks of the 8 Steps in 3PA on various Distributions and Instance types
Algorithms Tested	3PA
Instance Classes	• Multi-graph random instances • General random instances • Top-N random instances • *Spliddit* dataset
Evaluation Metric and Plot	Bar-graph of Average Wall-clock runtime and Average Raw Frequency Count

Table D.2: Summary of Experiment 2 — Runtime Comparison for Complete Allocations (See Table 4.1 for shared parameters)

Aspect	**Description**
Objective	Comparing Critical goods per Source and Valuable per Source
Algorithms Tested	3PA
Instance Classes	See shared parameters for full description of distributions: • Multi-graph random instances • General random instances • Top-N random instances • *Spliddit* dataset
Baseline Algorithms	Draft-And-Eliminate, Envy-Cycle Elimination,Round-Robin
Evaluation Metric and Plot Type	Heatmap of all 1000 instances plotting Critical Good per Source (if any) and Valuable Good per Source in envy-graph G respectively.

Table D.3: Summary of Experiment 3 — Runtime Comparison for Complete Allocations (See Table 4.1 for shared parameters)

Aspect	**Description**
Objective	Generating group champion graphs
Algorithms Tested	3PA
Size of Random Instance	Draw $\|N\| \in \{1, 25\}$ and $\|M\| \in \{N+1, 2*N+10\}$ to ensure at least 10 goods are left in the pool.
Instance Classes	• Multi-graph instances with all distributions • General instances with all distributions • *Spliddit dataset* • Top-N instances with all distributions
Evaluation Metric	Sparsity of Group Champion Graphs

Table D.4: Summary of Experiment 4 — Runtime Comparison for Complete Allocations (See Table 4.1 for shared parameters)

Appendix E

Extra Results

Figure E.1: (Experiment 1) Average Runtime Across All Distributions and Instance Types (Columns: Top-N, Multi-graph, General; Rows: Normal, Uniform Homogeneous, Uniform, Quadratic Face-Up, Quadratic Face-Down, Binary)

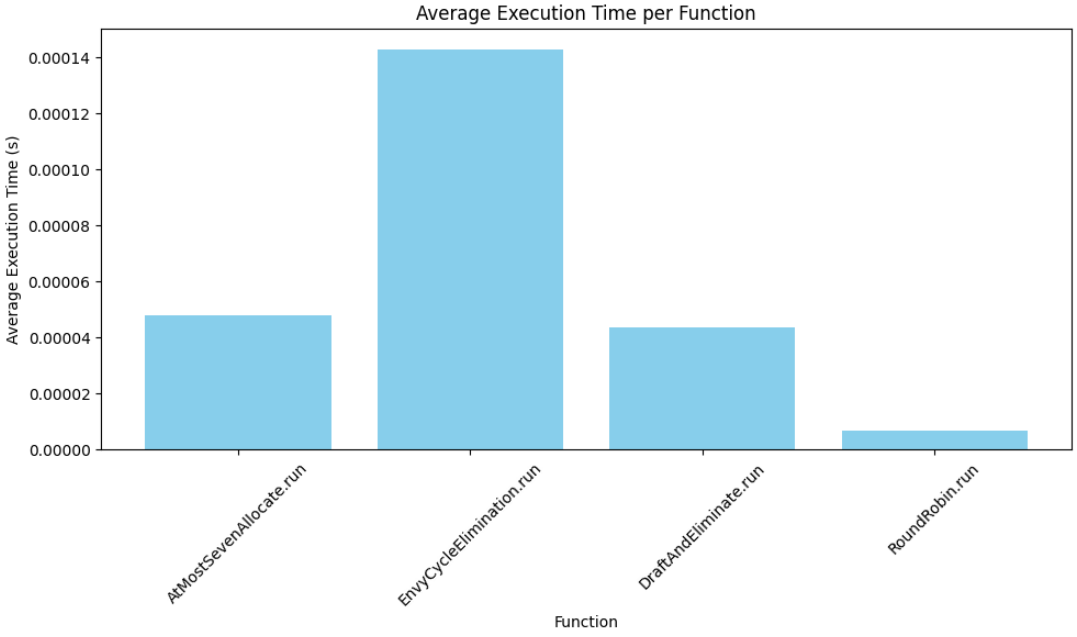


Figure E.2: (Experiment 1) Average Runtime Across *Spliddit* data

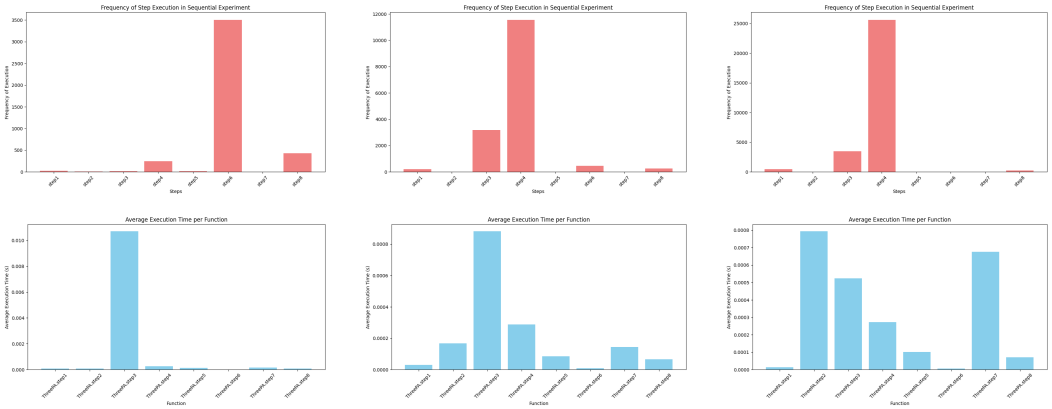


Figure E.3: (Experiment 2) Frequency and Runtime Breakdown for Quadratic Face-Up Distribution

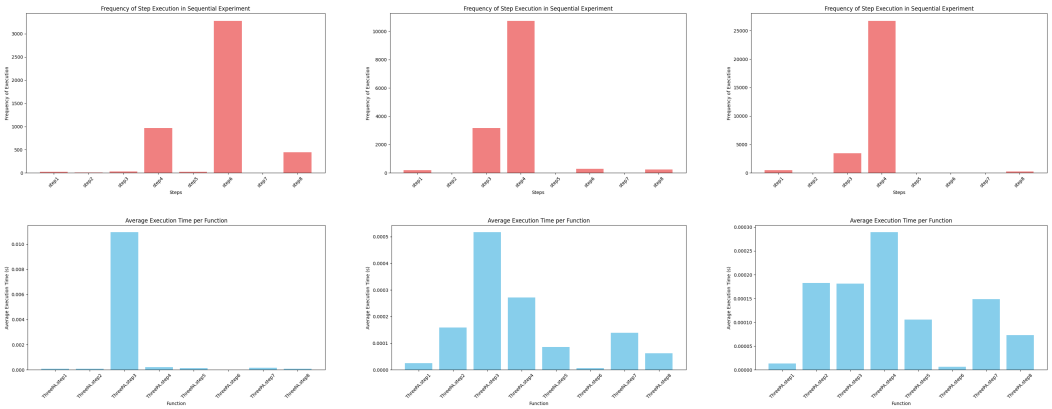


Figure E.4: (Experiment 2) Frequency and Runtime Breakdown for Uniform Distribution

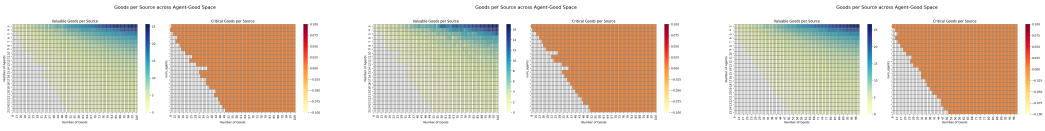


Figure E.5: (Experiment 3) Valuable/Critical Goods per Source for Binary Distribution

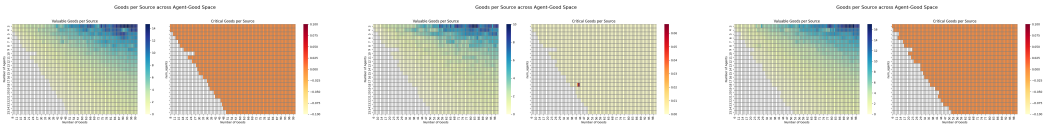


Figure E.6: (Experiment 3) Valuable/Critical Goods per Source for Normal Distribution

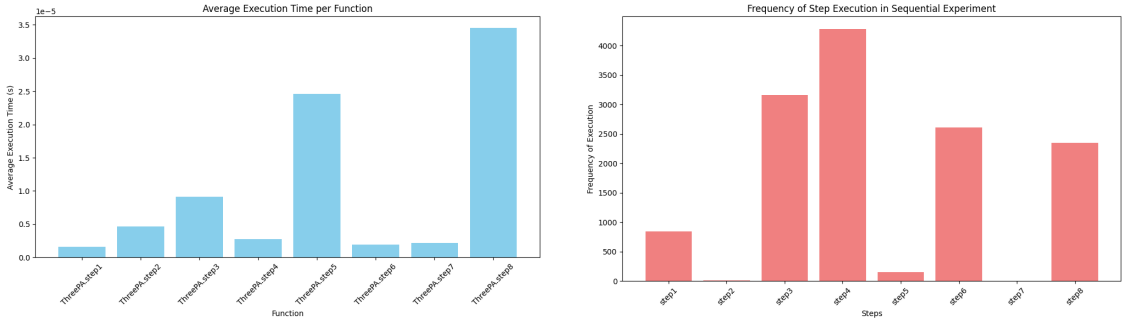


Figure E.7: (Experiment 2) Frequency and Runtime Breakdown for *Spliddit* data

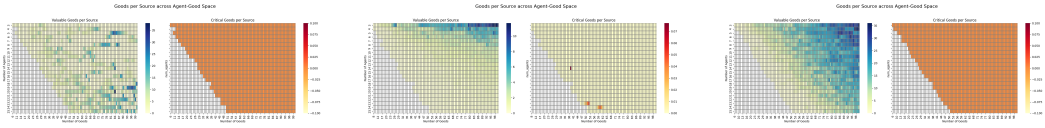


Figure E.8: (Experiment 3) Valuable/Critical Goods per Source for Uniform Homogeneous Distribution

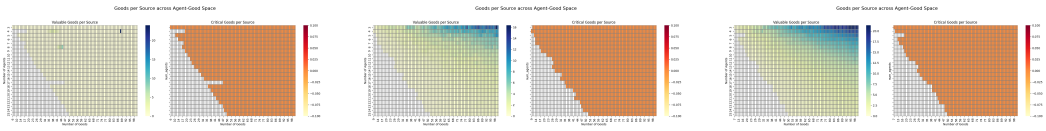


Figure E.9: (Experiment 3) Valuable/Critical Goods per Source for Uniform Distribution

Figure E.10: (Experiment 3) Valuable/Critical Goods per Source for Quadratic Face-Up Distribution

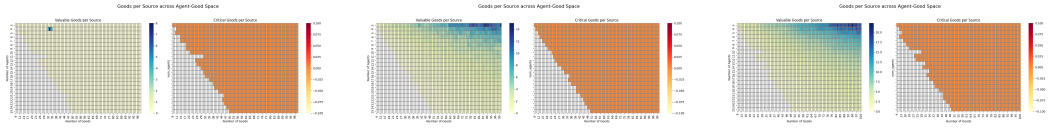


Figure E.11: (Experiment 3) Valuable/Critical Goods per Source for Quadratic Face-Down Distribution

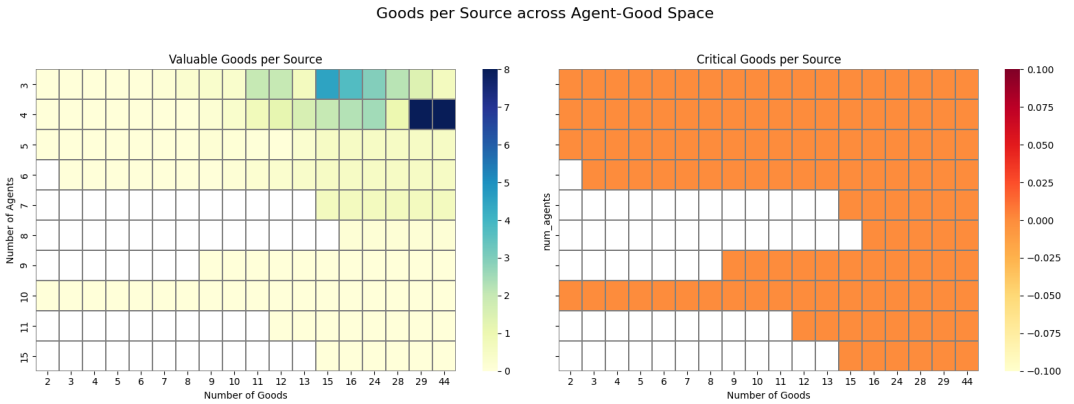


Figure E.12: (Experiment 3) Valuable/Critical Goods per Source for *Spliddit* data