Fairly Allocating Many Goods with Few Queries

Hoon Oh
Computer Science Department
Carnegie Mellon University

Ariel D. Procaccia
Computer Science Department
Carnegie Mellon University

Warut Suksompong
Department of Computer Science
Stanford University

Abstract

We investigate the query complexity of the fair allocation of indivisible goods. For two agents with arbitrary monotonic valuations, we design an algorithm that computes an allocation satisfying envy-freeness up to one good (EF1), a relaxation of envy-freeness, using a logarithmic number of queries. We show that the logarithmic query complexity bound also holds for three agents with additive valuations. These results suggest that it is possible to fairly allocate goods in practice even when the number of goods is extremely large. By contrast, we prove that computing an allocation satisfying envy-freeness and another of its relaxations, envy-freeness up to any good (EFX), requires a linear number of queries even when there are only two agents with identical additive valuations.

1 Introduction

Fair division is the study of how to allocate resources among interested agents in such a way that all agents find the resulting allocation to be fair (Brams and Taylor, 1996; Moulin, 2003). One of the field's paradigmatic applications is the allocation of *indivisible* goods; this task typically arises in inheritance cases, when, say, an art or jewelry collection is divided between several heirs. Indeed, dividing goods is one of five applications offered by *Spliddit* (Goldman and Procaccia, 2014), a not-for-profit fair division website; since its launch in November 2014, the website has served more than 130,000 users, and, in particular, has solved thousands of goods-division instances submitted by users.

While Steinhaus (1948) was the first to study fairness from a mathematical point of view, the history of fair division actually goes back much further: A simple fair division mechanism called the *cut-and-choose protocol* is mentioned in the Book of Genesis. After a dispute between Abraham and Lot, Abraham suggests that the two go their separate ways. He divides the land into two parts that—here we are perhaps using artistic license—he likes equally, and lets Lot choose the part that he prefers. The cut-and-choose protocol ensures that the resulting allocation satisfy an important fairness property called *envy-freeness*—each of Abraham and Lot finds his part to be worth at least as much as the other person's part. Even though envy-freeness can always be satisfied when the allocated resources are divisible (Stromquist, 1980), this is not the case when we deal with indivisible resources. With two agents and a single indivisible good, we already see that one of the agents will not receive the good and envy the other agent.

Consequently, various relaxations of envy-freeness have been considered, the most prominent one being *envy-freeness up to one good (EF1)*. This means that some agent may envy another agent under the given allocation, but that envy can be eliminated by removing a single good from the latter agent's bundle. Lipton et al. (2004) showed that EF1 can be guaranteed even when the agents have arbitrary monotonic valuations. They achieved this by using an algorithm that we will refer to as the *envy cycle elimination algorithm*, which runs in time $O(n^3m)$, where n is the number of agents and m the number of goods.

Given that an EF1 allocation always exists and can be found efficiently at this level of generality, a natural question to ask is how much we need to know about the agents' valuations to compute such an allocation. This issue is crucial for combinatorial valuations, since merely writing down a complete valuation might already take exponential time. But the question is equally important for additive valuations; while expressing such a valuation only takes linear time, this may already be prohibitive if the number of goods is very large. In fact, the goods application on Spliddit elicits additive valuations and computes an EF1 allocation (Caragiannis et al., 2016); the largest instance that was encountered involved ten siblings and roughly 1400 goods. In this case, the siblings actually prepared a spreadsheet with their value for each of the goods!

1.1 Our Results

We allow algorithms to elicit the valuations of agents via a standard interface, *value queries*, which ask an agent for her value for a given subset of goods. The complexity of algorithms is measured in terms of the worst-case number of queries they require.

In Section 3 we consider the case of two agents. We show that it is possible to compute an EF1 allocation for agents with arbitrary monotonic valuations using a logarithmic number of queries (Theorem 3.1). This is asymptotically tight, even for two agents with identical and very simple binary valuations (Proposition 3.2). We then turn to envy-freeness and establish that determining whether an envy-free allocation exists takes an exponential number of queries for

Fairness notion	Monotonic valuations	Additive valuations
EF	$\geq \binom{m}{m/2}$ (Prop. 3.3)	$\Theta(m)$ (Thm. 3.4)
EFX	$\Omega\left(\frac{1}{m}\binom{m}{(m-1)/2}\right)$ (Plaut and Roughgarden, 2018)	$\Theta(m)$ (Thm. 3.5)
EF1	$\Theta(\log m)$ (Thm. 3.1, Prop. 3.2)	$\Theta(\log m)$ (Thm. 3.1, Prop. 3.2)

Table 1: Query complexity in the setting with two agents. All lower bounds hold even when the two agents have identical valuations.

agents with identical monotonic valuations (Proposition 3.3) and a linear number of queries for agents with identical additive valuations (Theorem 3.4); our latter bound is also exactly tight. We end our investigation of the two-agent case by considering another relaxation of envy-freeness called *envy-freeness up to any good (EFX)*, a stronger notion than EF1. We show that computing an EFX allocation already takes a linear number of queries for agents with identical additive valuations (Theorem 3.5). This complements a recent result of Plaut and Roughgarden (2018), who showed that while an EFX allocation always exists for two agents with arbitrary monotonic valuations, computing one such allocation already requires an exponential number of queries in the worst case, even when the valuations of the agents are identical. Taken together, these results suggest that, when the number of goods is large, EF1 is the 'right' notion of fairness, whereas EFX is too demanding. The results of Section 3 are summarized in Table 1.

In Section 4 we address the case of three agents. Our main result is an algorithm that computes an EF1 allocation for three agents with additive valuations using a logarithmic number of queries (Theorem 4.4). Our algorithm adapts the Selfridge-Conway procedure, a classical *cake-cutting* protocol for computing an envy-free allocation of a heterogeneous *divisible* good, to the setting of indivisible goods. In particular, as a building block we use an algorithm that, for three agents with *identical* additive valuations, computes an EF1 allocation satisfying the extra property that any three predetermined goods belong to three different bundles (Lemma 4.3).

Finally, in Section 5 we consider the setting where there can be any number of agents. We show that the envy cycle elimination algorithm of Lipton et al. (2004) can be implemented using a relatively modest number of queries (Theorem 5.1). To complement this positive result, we conclude by presenting a lower bound on the number of queries needed to compute an EF1 allocation (Theorem 5.2).

1.2 Related Work

The paper that is most closely related to ours is the one mentioned above, by Plaut and Roughgarden (2018). Using an interesting reduction from the local search problem on a class of graphs known as Kneser graphs, they show that the problem of finding an EFX allocation requires an exponential number of queries, even for two agents with identical valuations. They also examine when EFX can be achieved in conjunction with other properties such as Pareto optimality, and establish the existence of allocations satisfying an approximate version of EFX for agents with subadditive valuations.

A bit further afield, query complexity has long been a topic of interest in computational fair division, albeit in the context of cake cutting (Procaccia, 2013). The standard query model is due to Robertson and Webb (1998), and allows two types of operations: evaluate (which is similar to our value queries) and cut. In this model, the query complexity of achieving fair cake allocations, under various notions of fairness, is well studied (Edmonds and Pruhs, 2006; Procaccia, 2009; Deng, Qi, and Saberi, 2012; Kurokawa, Lai, and Procaccia, 2013; Aziz and Mackenzie, 2016a; Procaccia and Wang, 2017).

2 Preliminaries

There is a set $G = \{g_1, g_2, \ldots, g_m\}$ of goods and a set $A = \{a_1, a_2, \ldots, a_n\}$ of agents. A *bundle* is a subset of G. Each agent a_i has a nonnegative valuation $u_i(G')$ for each $G' \subseteq G$. We sometimes abuse notation and write $u_i(g)$ for $u_i(\{g\})$.

A valuation is said to be *monotonic* if $u_i(G_1) \leq u_i(G_2)$ for any i and any $G_1 \subseteq G_2 \subseteq G$. It is said to be *additive* if $u_i(G') = \sum_{g\in G'} u_i(g)$ for any $G' \subseteq G$, and *binary* if it is additive and $u_i(g) = 0$ or 1 for each $g \in G$. While additivity is significantly more restrictive than monotonicity, many papers in fair division assume that agents' valuations are additive (Amanatidis et al., 2015; Bouveret and Lemaître, 2016; Caragiannis et al., 2016; Kurokawa, Procaccia, and Wang, 2018). This assumption is also made by Spliddit's app for dividing goods (Caragiannis et al., 2016), as, in practice, additive valuations hit a sweet spot between expressiveness and ease of elicitation. We assume throughout the paper that agents have monotonic valuations[1] and that, without loss of generality, $u_i(\emptyset) = 0$ for all i.

An *allocation* is a partition of G into n bundles $(G_1, G_2, \ldots, G_n)$, where bundle i is allocated to agent i. If the goods lie on a line, for each good g we denote by L_g and R_g the set of goods to the left and right of g, respectively. A *contiguous* allocation is an allocation in which every bundle forms a contiguous block on the line.

[1]Without this assumption, neither of the relaxations of envy-freeness that we consider can always be satisfied even when there are two agents.

We now define the fairness notions that we consider.

Definition 2.1. An allocation $(G_1, G_2, \ldots, G_n)$ is said to be

- *envy-free* if $u_i(G_i) \geq u_i(G_j)$ for any i, j.
- *envy-free up to any good (EFX)* if for any i, j and any good $g \in G_j$, $u_i(G_i) \geq u_i(G_j \backslash \{g\})$.
- *envy-free up to one good (EF1)* if for any i, j such that $u_i(G_i) < u_i(G_j)$, there exists a good $g \in G_j$ such that $u_i(G_i) \geq u_i(G_j \backslash \{g\})$.[2]

It is easy to see that envy-freeness is stronger than EFX, which is in turn stronger than EF1. Envy-freeness is a classical and well-studied fairness notion that goes back to Foley (1967). By contrast, its two relaxations are relatively new: EF1 was introduced by Budish (2011) and a related property was studied by Lipton et al. (2004), while EFX was only proposed recently by Caragiannis et al. (2016). Interestingly, it is not known whether an EFX allocation always exists, even for three agents with additive valuations (Caragiannis et al., 2016; Plaut and Roughgarden, 2018).

We will consider algorithms that compute fair allocations according to these fairness notions. In order to discover the agents' valuations, an algorithm is allowed to issue *value queries*. In each query, the algorithm chooses an agent a_i and a subset $G' \subseteq G$, and finds out the value of $u_i(G')$. We assume that the algorithm is deterministic, and allow it to be *adaptive*, i.e., the algorithm can determine its next query based on its past queries and the corresponding answers.

3 Two Agents

In this section, we consider the setting with two agents. We organize our results based on fairness notion: EF1, envy-freeness, and EFX.

3.1 EF1

We begin by describing an algorithm that computes an EF1 allocation for two agents with arbitrary monotonic valuations. The algorithm is similar to the cut-and-choose protocol for cake cutting: the first agent partitions the goods into two bundles with the property that she would be satisfied with either bundle, and the second agent chooses the bundle that she prefers. In order to minimize the number of queries, we arrange the goods on a line and use binary search to determine the cut point of the first agent.

Algorithm 1 (for two agents with monotonic valuations)

Step 1: Arrange the goods on a line in arbitrary order. Find the rightmost good g such that $u_1(L_g) \leq u_1(R_g \cup \{g\})$.

Step 2: If $u_1(L_g) \leq u_1(R_g)$, consider the partition $(L_g \cup \{g\}, R_g)$; else, consider the partition $(L_g, R_g \cup \{g\})$. Give a_2 the bundle from the partition that she prefers, and a_1 the remaining bundle.

[2] The clause "such that $u_i(G_i) < u_i(G_j)$" is necessary for the case where $G_j = \emptyset$.

We claim that the algorithm computes an EF1 allocation using a logarithmic number of queries.

Theorem 3.1. *For two agents with arbitrary monotonic valuations, Algorithm 1 computes an EF1 allocation. Moreover, the algorithm can be implemented to use $O(\log m)$ queries in the worst case.*

Proof. We first show that the algorithm computes an EF1 allocation. Since a_2 gets the bundle that she prefers, she does not envy a_1.

To reason about a_1's envy, assume first that $u_1(L_g) \leq u_1(R_g)$. It holds that $u_1(L_g \cup \{g\}) \geq u_1(R_g)$: This is clearly true if g is the rightmost good on the line, and otherwise it follows from the definition of g that $u_1(R_g) < u_1(L_g \cup \{g\})$. Therefore, if a_1 receives $L_g \cup \{g\}$, she is not envious at all. And if she receives R_g, it holds that

$$u_1(R_g) \geq u_1(L_g) = u_1((L_g \cup \{g\}) \setminus \{g\}),$$

so EF1 is satisfied.

The second case is where $u_1(L_g) > u_1(R_g)$. If a_1 gets $R_g \cup \{g\}$ then she is not envious, since, by the definition of g, $u_1(R_g \cup \{g\}) \geq u_1(L_g)$. If she gets L_g instead, then EF1 holds, because

$$u_1(L_g) > u_1(R_g) = u_1((R_g \cup \{g\}) \setminus \{g\}).$$

Next, we show that the algorithm can be implemented to use $O(\log m)$ queries. By monotonicity, Step 1 can be done by binary search using $O(\log m)$ queries. In Step 2, we use two queries to compare $u_1(L_g)$ and $u_1(R_g)$, and two more queries to compare a_2's valuation for the two bundles in the partition. Hence the total number of queries is $O(\log m)$. □

The following proposition shows that the bound $O(\log m)$ in Theorem 3.1 is tight.

Proposition 3.2. *Any deterministic algorithm that computes an EF1 allocation for two agents with identical binary valuations uses $\Omega(\log m)$ queries in the worst case, even when each agent values only two goods.*

Since Proposition 3.2 will later be generalized by Theorem 5.2, we do not present its proof here.

3.2 Envy-freeness

Next, we turn our attention to envy-freeness. Unlike the case of EF1, allocations that satisfy envy-freeness are not guaranteed to exist. We show that for two agents with identical monotonic valuations, even an algorithm that only decides whether an envy-free allocation exists already needs to make an exponential number of queries in the worst case. A similar argument holds for algorithms that compute an envy-free allocation whenever one exists.

Proposition 3.3. *Assume that m is even. Any deterministic algorithm that determines whether an envy-free allocation exists for two agents with identical monotonic valuations uses at least $\binom{m}{m/2}$ queries in the worst case.*

The proof of Proposition 3.3, and all other omitted proofs, can be found in Appendix A.

Even though an algorithm that decides whether an envy-free allocation exists needs to make an exponential number of queries for agents with monotonic valuations, when we restrict our attention to agents with additive valuations, the exponential lower bound no longer holds since the algorithm can query the value of both agents for every good and find out the full valuations. Nevertheless, it is still conceivable that there are algorithms that do asymptotically better, e.g., use a logarithmic number of queries. We show that this is not the case: a linear number of queries is necessary, even when the two agents have identical valuations. In fact, we leverage linear-algebraic techniques to establish that at least m queries are needed in this case. This bound is tight for two identical agents since the algorithm can find out the common valuation by querying the value of each of the m goods.

Theorem 3.4. *Assume that m is even. Any deterministic algorithm that decides whether an envy-free allocation exists for two agents with identical additive valuations uses at least m queries in the worst case.*

Proof. For ease of notation, let $x_i = u(g_i)$ for $i = 1, 2, \ldots, m$, where u is the common valuation. Note that an envy-free allocation exists if and only if the goods can be partitioned into two sets of equal value. Consider an algorithm that always uses at most $m-1$ queries. Assume without loss of generality that the algorithm always uses exactly $m-1$ queries; whenever it uses fewer than $m-1$ queries, we add arbitrary queries for the algorithm. The idea is that for each query, if the queried subset has size s, we will give an answer close to s in such a way that after all queries, it is still possible that there exists an envy-free allocation, but also that there does not exist one. This will allow us to obtain the desired conclusion.

For $i = 1, 2, \ldots, m-1$, let $\mathbf{v}_i$ be a vector of length m where the jth component is 1 if good g_j is included in the ith query, and 0 otherwise. Therefore the ith query asks for the value $\mathbf{v}_i \cdot \mathbf{x} = \sum_{j=1}^{m} v_{i,j} x_j$. Furthermore, let W be the set of all vectors of length m all of whose components are ± 1, and let $W' \subset W$ be the set of vectors with an equal number of -1 and 1. Note that an envy-free allocation exists exactly when $\mathbf{w} \cdot \mathbf{x} = 0$ for some $\mathbf{w} \in W$.

When we receive the ith query, if $\mathbf{v}_i \in \text{span}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{i-1})$, our answer is already determined by previous answers. Assume therefore that $\mathbf{v}_i \notin \text{span}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{i-1})$ for all i. For each $\mathbf{w} \in W$ such that $\mathbf{w} \in \text{span}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_i) \backslash \text{span}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{i-1})$, there exists a unique answer that would force $\mathbf{w} \cdot \mathbf{x} = 0$. We avoid all such (finite number of) answers. After query number $m-1$ we have a subspace $\mathcal{V} = \text{span}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{m-1})$ of dimension $m-1$ such that we know the value $\mathbf{v} \cdot \mathbf{x}$ if and only if $\mathbf{v}$ is in the subspace.

Next, let $\mathcal{W}' = \text{span}(W')$. Clearly, all vectors in $\mathcal{W}'$ are orthogonal to the vector $(1, 1, \ldots, 1)$. We claim that $\mathcal{W}'$ in fact has dimension $m-1$, and therefore consists of all vectors orthogonal to $(1, 1, \ldots, 1)$. To see this, take two distinct vectors in W' that differ only in the first and ith component for $i = 2, 3, \ldots, m$. The difference vector, which consists of a 1 in the first position, a -1 in the ith position, and 0 elsewhere belongs to $\mathcal{W}'$. No nontrivial linear combination of these vectors can produce the all-zero vector, meaning that $\mathcal{W}'$ indeed has dimension $m-1$. Now, since any vector $\mathbf{v}_i$ is not orthogonal to $(1, 1, \ldots, 1)$, we have $\mathcal{V} \neq \mathcal{W}'$, and so there exists $\mathbf{w}' \in W'$ such that $\mathbf{w}' \notin \mathcal{V}$. (If this were not the case, we would have $\mathcal{W}' \subseteq \mathcal{V}$, and then the two subspaces would be equal because they are of the same dimension.) Since $\mathcal{V}$ is of dimension $m-1$ and $\mathbf{w}' \notin \mathcal{V}$, setting the value of $\mathbf{w}' \cdot \mathbf{x}$ will, in combination with the previous constraints, uniquely determine $\mathbf{x}$. If we set $\mathbf{w}' \cdot \mathbf{x} = 0$, an envy-free allocation exists. On the other hand, if we set $\mathbf{w}' \cdot \mathbf{x}$ so that $\mathbf{w} \cdot \mathbf{x} \neq 0$ for all $\mathbf{w} \in W$, an envy-free allocation does not exist. This choice of value for $\mathbf{w}' \cdot \mathbf{x}$ is available because for each $\mathbf{w} \in W$, only one value of $\mathbf{w}' \cdot \mathbf{x}$ forces $\mathbf{w} \cdot \mathbf{x} = 0$.

It remains to show that we can give the answers so that after setting the value of $\mathbf{w}' \cdot \mathbf{x}$, all components of the unique solution for $\mathbf{x}$ are nonnegative. Let $\delta > 0$ be such that for any vector $\mathbf{z}$ with $|z_i| < \delta$ for all $i = 1, 2, \ldots, m$, and any $m \times m$ invertible matrix M all of whose entries are -1, 0, or 1, the unique solution $\mathbf{y}$ to $M\mathbf{y} = \mathbf{z}$ has $|y_i| < 1$ for all i. The existence of δ is guaranteed by the fact that the linear transformation M takes the all-zero vector to itself, by the continuity of the transformation, and by the fact that there are a finite number of such matrices M. For each query on a subset of size k, we give an answer in the range $(k-\delta, k+\delta)$. Moreover, we choose the value of $\mathbf{w}' \cdot \mathbf{x}$ to be in the range $(-\delta, \delta)$.

Write $y_i = x_i - 1$ for all i, where $\mathbf{x}$ is the unique solution according to our choices. Our answers to the queries ensure that the values of $\mathbf{v}_i \cdot \mathbf{y}$ for $i = 1, 2, \ldots, m-1$ belong to the range $(-\delta, \delta)$, and our choice of $\mathbf{w}' \cdot \mathbf{x}$ ensures that $\mathbf{w}' \cdot \mathbf{y}$ also belong to the same range. Since all elements of $\mathbf{v}_i$ and $\mathbf{w}'$ belongs to the set $\{-1, 0, 1\}$, by taking M to be the $m \times m$ matrix with $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{m-1}$ and $\mathbf{w}'$ as its rows, and $\mathbf{z}$ to be a vector of length m with $\mathbf{v}_1 \cdot \mathbf{y}, \mathbf{v}_2 \cdot \mathbf{y}, \ldots, \mathbf{v}_{m-1} \cdot \mathbf{y}$ and $\mathbf{w}' \cdot \mathbf{y}$ as its elements, our definition of δ implies that $|y_i| < 1$ for all i. It follows that $x_i > 0$ for all i, as desired. □

For two agents with additive valuations, envy-freeness is equivalent to another well-known fairness notion called proportionality, which requires that each agent receive at least half of her value for the whole set of goods. Thus, the lower bound in Theorem 3.4 also holds for two agents with identical additive valuations with respect to proportionality.

3.3 EFX

We end this section by considering EFX. For two agents with monotonic valuations, Plaut and Roughgarden (2018) showed that an EFX allocation is guaranteed to exist, but computing it takes an exponential number of queries in the worst case. If the agents have additive valuations, however, the algorithm can already find out the full valuations using only a linear number of queries. Our next result shows that a linear number of queries is, in fact, needed for computing an EFX allocation.

Theorem 3.5. *Assume that m is odd. Any deterministic algorithm that computes an EFX allocation for two agents* *with identical additive valuations uses at least* $(m-1)/2$ *queries in the worst case.*

Note that Theorem 3.5 is incomparable with Theorem 3.4, even though EFX is a relaxation of envy-freeness, because the former result deals with a *search* problem (finding an EFX allocation knowing that one always exists), whereas the latter deals with a *decision* problem (deciding whether an EF allocation exists at all).

4 Three Agents

In this section, we study the setting with three agents who are endowed with additive valuations. Our main result is an algorithm that finds an EF1 allocation using $O(\log m)$ queries, but we first need to develop some machinery for the case where the agents have identical valuations.

4.1 Identical Additive Valuations

While the case of identical additive valuations might seem trivial at first glance, as we will see, there are already several interesting statements that we can make about the setting, so it may be of independent interest. We begin by establishing some properties of a particular partition of goods on a line into two contiguous blocks.

Lemma 4.1. *Assume that the goods lie on a line. Suppose that an agent with an additive valuation* u *chooses the partition of the goods into two contiguous blocks that minimizes the difference between the values of the two blocks. (If there are goods of value 0 next to the cut point, move the cut point until these goods belong to the block of lower value.) Let* L *be the left block and* g_l *the rightmost good of the block. Similarly, let* R *be the right block and* g_r *the leftmost good of the block. Then:*

1. *We have that* $\min\{u(G)/2, u(L)\} \geq u(R\backslash\{g_r\})$ *and* $\min\{u(G)/2, u(R)\} \geq u(L\backslash\{g_l\})$.
2. *The partition can be computed using* $O(\log m)$ *queries in the worst case.*

Next, we present an algorithm that computes a contiguous EF1 allocation for three agents with identical valuations using a logarithmic number of queries. Not only will the contiguity condition be useful later in our algorithm for three agents with arbitrary valuations, but in certain applications it may also be desirable to produce a contiguous allocation. For example, if the goods are office space, it is conceivable that each research group wishes to have a consecutive block of offices in order to facilitate collaboration within the group. While contiguous fair allocations of indivisible goods have recently been studied (Bouveret et al., 2017; Suksompong, 2017), to the best of our knowledge even the *existence* of a contiguous EF1 allocation for three agents with identical valuations has not been established before, let alone an algorithm that computes such an allocation using a small number of queries. Hence our result may be of interest even if one is not concerned with the number of queries made. In the appendix, the existence result is generalized to any number of agents with identical *monotonic* valuations (Lemma C.2).

Algorithm 2 (for three agents with *identical* additive valuations)

Step 1: Assume that the goods lie on a line, and denote by u the common valuation of the three agents. Let g_1 be the leftmost good such that $u(L_{g_1} \cup \{g_1\}) > u(G)/3$, and let g_2 be the rightmost good such that $u(R_{g_2} \cup \{g_2\}) > u(G)/3$. (Possibly $g_1 = g_2$.) Assume without loss of generality that $u(L_{g_1}) \geq u(R_{g_2})$.

Step 2: If $L_{g_1} \neq \emptyset$, let g_3 be the leftmost good such that $u(L_{g_3} \cup \{g_3\}) \geq u(R_{g_2})$. Set $A = L_{g_3} \cup \{g_3\}$. Else, set $A = \emptyset$. In both cases, set $C = R_{g_2}$ and $B = G\backslash(A \cup C)$.

Step 3: If $u(C) \geq u(B\backslash\{g_2\})$, return the allocation (A, B, C). Else, set $C' = R_{g_2} \cup \{g_2\}$. Partition the remaining goods into two contiguous blocks according to Lemma 4.1; denote by A' the left block and B' the right block. Return the allocation (A', B', C').

The following lemma establishes the claimed properties of Algorithm 2.

Lemma 4.2. *Assume that the goods lie on a line. For three agents with identical additive valuations, Algorithm 2 computes a contiguous EF1 allocation. Moreover, the algorithm can be implemented to use* $O(\log m)$ *queries in the worst case.*

A bonus of Algorithm 2 is that in the allocation produced by the algorithm, if some agent envies another agent, then the envy can be eliminated by removing not just some arbitrary good from the latter agent's bundle, but one of the goods at the end of the latter agent's block. In fact, we can also choose this good to be a good next to a cut point; this nails down a unique good for the agents getting the left or right block. The property can be deduced from the proof of Lemma 4.2.

To demonstrate that even the problem of establishing the existence of a contiguous EF1 allocation in this setting is not straightforward, we present a very natural approach that, perhaps surprisingly, does not work. We first pretend that the goods are divisible and find the two cut points that would divide the goods into three parts of exactly equal value. For each cut point, if the cut point falls between two (now indivisible) goods, we keep it; otherwise we round it either to the left or to the right. One might be tempted to claim that at least one of the resulting allocations must be EF1. Indeed, Lemma 4.1 implies that an analogous approach works for two agents. However, an example given in Appendix B shows that the approach does not work for three agents, no matter how we round the cut points.

Next, we leverage Algorithm 2 to show that for three agents with identical additive valuations, if we designate three goods in advance, it is possible to compute an EF1 allocation such that all three designated goods belong to different bundles.

Lemma 4.3. *Let g_1, g_2, g_3 be three distinct goods. For three agents with identical additive valuations, there exists a deterministic algorithm that computes an EF1 allocation such that the three goods belong to three different bundles using $O(\log m)$ queries in the worst case.*

Note that for two agents, an analogous statement holds even when the agents have arbitrary monotonic valuations, since we can place the two designated goods at different ends of a line and apply Algorithm 1.

4.2 Arbitrary Additive Valuations

With Algorithm 2 and Lemma 4.3 in hand, we are now ready to present an algorithm that computes an EF1 allocation for three agents with arbitrary additive valuations using a logarithmic number of queries. The algorithm is based on the Selfridge-Conway procedure for computing an envy-free allocation of *divisible* goods, often modeled as a cake, for three agents. At a high level, the Selfridge-Conway procedure operates by letting the first agent divide the cake into three equal pieces and letting the second agent trim her favorite piece so that it is equal to her second favorite piece. Then, the procedure allocates one "main" piece to each agent, with the third agent choosing first, and allocates the leftover cake in a carefully designed way.

Like the Selfridge-Conway procedure, our algorithm starts by letting the first agent divide the goods into three almost equal bundles using Algorithm 2, so that no matter how the bundles are allocated, the agent finds the allocation to be EF1. It then proceeds by letting the second agent trim her favorite bundle so that her value for the bundle goes just below that for her second favorite bundle. However, a difficulty in our indivisible goods setting is that at this point, the second agent might find the remaining part of her favorite bundle to be worth less than her second favorite bundle *even* if we remove any good from her second favorite bundle. This is possible, for instance, if the last good that she removes from her favorite bundle is of high value, and her second favorite bundle only consists of goods of low value. We will need to fix this problem by finding "large" goods in the leftover bundle that help us recover the EF1 guarantee; this is done in Step 3 of the algorithm. While identifying these large goods can be done easily if we can make queries for the value of every good in the leftover bundle, we would not achieve the logarithmic bound if the leftover piece contained more than a logarithmic number of goods.

Algorithm 3 (for three agents with additive valuations)

Step 1: Compute an EF1 allocation (A, B, C) for three identical agents with valuation u_1. If a_2 and a_3 have different favorite bundles among A, B, C, give them their favorite bundles, and give the remaining bundle to a_1. Else, assume without loss of generality that $u_2(A) > u_2(B) \geq u_2(C)$ and $u_3(A) > \max\{u_3(B), u_3(C)\}$.

Step 2: Let a_2 divide A into A' and T such that $u_2(A') \leq u_2(B)$ and there exists $g \in T$ with $u_2(A' \cup \{g\}) > u_2(B)$. If $u_3(A') \geq \max\{u_3(B), u_3(C)\}$, give A' to a_3, B to a_2, and C to a_1. Compute an EF1 allocation (T_1, T_2, T_3) of the goods in T for three identical agents with valuation u_2. Let a_3 choose her favorite bundle followed by a_1, and let a_2 take the remaining bundle. Else, we have $u_3(A') < \max\{u_3(B), u_3(C)\}$.

Step 3: Define $d = u_2(B) - u_2(A') \geq 0$. Call a good g *large* if $g \in T$ and $u_2(g) \geq d$, where we update A', T, and d during the course of the step. Let a_2 find up to three large goods. The first large good is the good $g \in T$ such that $u_2(A' \cup \{g\}) > u_2(B)$. To find further large goods, let $E \subseteq T$ be such that $u_2(E) \geq d$, $u_2(E \backslash \{g\}) < d$ for some $g \in E$, and E does not contain any identified large good. If such a set E (and good g) exists, remove the goods in $E \backslash \{g\}$ from T and add them to A', and decrease d by $u_2(E \backslash \{g\})$. (For the first large good, take $E = \{g\}$.) Then g is a new large good. If $u_3(A') \geq \max\{u_3(B), u_3(C)\}$ with the updated set A', allocate the goods according to Step 2. So we may still assume that $u_3(A') < \max\{u_3(B), u_3(C)\}$. On the other hand, if no such set E exists, remove all goods except the (up to two) identified large goods from T and add them to A', and decrease d by a_2's value for these goods.

Step 4: Compute an EF1 allocation (T_1, T_2, T_3) of the goods in T for three identical agents with valuation u_3 in such a way that all identified large goods belong to different bundles.

Step 5: Let S_3 be a_3's preferred bundle between B and C, and let S_1 be the other bundle. Give $S_2 := A'$ to a_2, S_3 to a_3, and S_1 to a_1.

Step 6: If there is an identified large good in each of T_1, T_2, and T_3, give a_2 her favorite bundle T_i if we were to remove the identified large good from each of these bundles. Let a_1 choose her preferred bundle from the remaining two bundles (without removing the large goods), and give a_3 the remaining bundle. Else, give the first identified large good to a_2 and the second identified large good (if exists) to a_1.

The following theorem, which we view as our main result, establishes the claimed properties of Algorithm 3 by leveraging the machinery developed above.

Theorem 4.4. *For three agents with additive valuations, Algorithm 3 computes an EF1 allocation. Moreover, the algorithm can be implemented to use $O(\log m)$ queries in the worst case.*

5 Any Number of Agents

In this section, we consider the general setting where there can be any number of agents. We state and discuss some results here, and relegate several results that require stronger assumptions to Appendix C.

Our starting point is the *envy cycle elimination algorithm* of Lipton et al. (2004), which computes an EF1 allocation for agents with arbitrary monotonic valuations. The algorithm works by allocating one good at a time in arbitrary order. It also maintains an *envy graph*, which has the agents as its vertices, and a directed edge from a_i to a_j if a_i envies a_j with respect to the current (partial) allocation. At each step, the next good is allocated to an agent with no incoming edge, and any cycle that arises as a result is eliminated by giving a_j's bundle to a_i for any edge from a_i to a_j in the cycle. This allows the algorithm to maintain the invariant that the envy graph has no cycles and therefore has an agent with no incoming edge before each allocation of a good. The envy cycle elimination algorithm runs in time $O(n^3m)$ in the worst case. We refer to the paper of Lipton et al. (2004) for the proof of correctness and detailed analysis of this algorithm.

Our main positive result for this section is the observation that the envy cycle elimination algorithm can be implemented using a relatively modest number of (value) queries.

Theorem 5.1. *For any number of agents with arbitrary monotonic valuations, the envy cycle elimination algorithm can be implemented to compute an EF1 allocation using:*

1. $O(nm)$ *queries in the worst case.*
2. $O(n^3k \log m)$ *queries in the worst case, if the valuation of each agent takes at most* k *(possibly unknown) values across all subsets of goods.*

Theorem 5.1 illustrates a sharp contrast between EF1 and the stronger fairness notions of envy-freeness and EFX. For the latter two notions, computing a fair allocation requires an exponential number of queries in the worst case, even in the most restricted setting of two agents with identical valuations. On the other hand, for EF1 we can get away with only $O(nm)$ queries even in the most general setting of any number of agents with arbitrary monotonic valuations. Moreover, if n and k are small compared to m, the bound of Item 2 of the theorem can be better than that of Item 1. In particular, if n and k are constant, the implementation only requires $O(\log m)$ queries. The case of $k = 2$ corresponds to the setting where each agent either approves or disapproves each subset of goods.[3] A small value of k may occur in settings where the mechanism designer gives a predefined set of preferences that the agents can express on each subset of goods, e.g., 'very interested', 'somewhat interested', and 'not interested'.

To complement this positive result, we conclude by giving a lower bound (which, sadly, does not match the upper bound) on the number of queries needed to compute an EF1 allocation.

Theorem 5.2. *Let* $m \geq n^\alpha$ *for some constant* $\alpha > 1$*. Any deterministic algorithm that computes an EF1 allocation for* n *agents with binary valuations uses* $\Omega(n \log m)$ *queries in the worst case.*

[3] This is not to be confused with what we call binary valuations in this paper, for which k can be as large as m.

Proof. Assume first that n is even, say $n = 2k$, and that each agent has value 1 for two goods and 0 for the remaining goods. Suppose further that for $i = 1, 2, \dots, k$, agents a_{2i-1} and a_{2i} have identical valuations; we abuse notation and denote this valuation by u_i. Note that if both of the goods valued by some agent are allocated to a single agent, the resulting allocation cannot be EF1.

Initially, for each $i = 1, 2, \dots, k$, let G_i be the whole set of goods. As long as $|G_i| > 2$, we answer the query of the algorithm on the value of $u_i(H)$ for a subset H of goods as follows. If $|G_i \cap H| \geq |G_i|/2$, answer 2 and replace G_i by $G_i \cap H$; else, answer 0 and replace G_i by $G_i \backslash H$. While $|G_i| > n$, the only information that the algorithm has is that both valued goods are contained in G_i. This information is not sufficient to return an allocation such that the two valued goods are guaranteed to be in different bundles, so the algorithm must keep making queries until $|G_i| \leq n$ for every i. Since initially $|G_i| = m$ and the size of G_i decreases by no more than half with each query, the algorithm uses at least $k \log(m/n)$ queries in the worst case. The conclusion follows from the observation that $\log(m/n) \geq \frac{\alpha-1}{\alpha} \cdot \log m$.

If n is odd, we can assume that the last agent has value 0 for all goods and deduce the same asymptotic bound using the remaining $n - 1$ agents. □

Since the assumption of Theorem 5.2 holds for any constant n if m is large enough, and when $n = 2$ the two agents considered in the proof have identical valuations and each agent values only two goods, this theorem implies Proposition 3.2.

6 Discussion

From a technical viewpoint, the main take-home message of our work is this: Envy-free cake cutting protocols, designed for *divisible* goods, can be adapted to yield EF1 allocations of *indivisible goods* using a logarithmic number of queries. On a high level, the idea is to arrange the goods on a line, and approximately implement cut operations using binary search. We do this to obtain Theorem 3.1, by adapting the cut-and-choose protocol, and Theorem 4.4, by building on the classic Selfridge-Conway protocol.

However, making sure the approximation errors do not add up in a way that violates EF1 already becomes nontrivial when there are three agents, as illustrated by Algorithm 3 and Theorem 4.4. Extending the approach even to four agents with arbitrary additive valuations, therefore, seems very challenging. A related difficulty is that the known envy-free cake cutting protocols for four or more agents are quite involved (Brams and Taylor, 1995; Aziz and Mackenzie, 2016a,b; Amanatidis et al., 2018).

Another intriguing question is whether the logarithmic upper bound on the complexity of EF1 extends to three agents with monotonic valuations. Such valuations cannot be handled by the Selfridge-Conway procedure, which is designed for the cake cutting setting where additivity is assumed. Of course, it is possible that, in fact, there is super-logarithmic lower bound on the query complexity in this case.

References

Amanatidis, G.; Markakis, E.; Nikzad, A.; and Saberi, A. 2015. Approximation algorithms for computing maximin share allocations. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, 39–51.

Amanatidis, G.; Christodoulou, G.; Fearnley, J.; Markakis, E.; Psomas, C.-A.; and Vakaliou, E. 2018. An improved envy-free cake cutting protocol for four agents. In *Proceedings of the 11th International Symposium on Algorithmic Game Theory (SAGT)*. Forthcoming.

Aziz, H., and Mackenzie, S. 2016a. A discrete and bounded envy-free cake cutting protocol for any number of agents. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, 416–427.

Aziz, H., and Mackenzie, S. 2016b. A discrete and bounded envy-free cake cutting protocol for four agents. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, 454–464.

Bouveret, S., and Lemaître, M. 2016. Characterizing conflicts in fair division of indivisible goods using a scale of criteria. *Autonomous Agents and Multi-Agent Systems* 30(2):259–290.

Bouveret, S.; Cechlárová, K.; Elkind, E.; Igarashi, A.; and Peters, D. 2017. Fair division of a graph. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 135–141.

Brams, S. J., and Taylor, A. D. 1995. An envy-free cake division protocol. *The American Mathematical Monthly* 102(1):9–18.

Brams, S. J., and Taylor, A. D. 1996. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press.

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

Caragiannis, I.; Kurokawa, D.; Moulin, H.; Procaccia, A. D.; Shah, N.; and Wang, J. 2016. The unreasonable fairness of maximum Nash welfare. In *Proceedings of the 17th ACM Conference on Economics and Computation (EC)*, 305–322.

Deng, X.; Qi, Q.; and Saberi, A. 2012. Algorithmic solutions for envy-free cake cutting. *Operations Research* 60(6):1461–1476.

Edmonds, J., and Pruhs, K. 2006. Cake cutting really is not a piece of cake. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 271–278.

Foley, D. 1967. Resource allocation and the public sector. *Yale Economics Essays* 7:45–98.

Goldman, J., and Procaccia, A. D. 2014. Spliddit: Unleashing fair division algorithms. *SIGecom Exchanges* 13(2):41–46.

Kurokawa, D.; Lai, J. K.; and Procaccia, A. D. 2013. How to cut a cake before the party ends. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, 555–561.

Kurokawa, D.; Procaccia, A. D.; and Wang, J. 2018. Fair enough: Guaranteeing approximate maximin shares. *Journal of the ACM* 64(2): article 8.

Lipton, R. J.; Markakis, E.; Mossel, E.; and Saberi, A. 2004. On approximately fair allocations of indivisible goods. In *Proceedings of the 6th ACM Conference on Economics and Computation (EC)*, 125–131.

Moulin, H. 2003. *Fair Division and Collective Welfare*. MIT Press.

Plaut, B., and Roughgarden, T. 2018. Almost envy-freeness with general valuations. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2584–2603.

Procaccia, A. D., and Wang, J. 2017. A lower bound for equitable cake cutting. In *Proceedings of the 18th ACM Conference on Economics and Computation (EC)*, 479–495.

Procaccia, A. D. 2009. Thou shalt covet thy neighbor's cake. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 239–244.

Procaccia, A. D. 2013. Cake cutting: Not just child's play. *Communications of the ACM* 56(7):78–87.

Robertson, J. M., and Webb, W. A. 1998. *Cake Cutting Algorithms: Be Fair If You Can*. A. K. Peters.

Steinhaus, H. 1948. The problem of fair division. *Econometrica* 16:101–104.

Stromquist, W. 1980. How to cut a cake fairly. *American Mathematical Monthly* 87(8):640–644.

Suksompong, W. 2017. Fairly allocating contiguous blocks of indivisible items. In *Proceedings of the 10th International Symposium on Algorithmic Game Theory (SAGT)*, 333–344.

A Omitted Proofs

Proof of Proposition 3.3

Assume that the common valuation u is such that $u(G_1) < u(G_2)$ whenever $|G_1| < |G_2|$. With this valuation, any envy-free allocation must give an equal number of goods to both agents. Since the values of subsets of size $m/2$ can be arbitrary, the algorithm must, in the worst case, query all such subsets in order to determine whether there exists a set G' such that $u(G') = u(G\backslash G')$. Hence the algorithm needs to query $\binom{m}{m/2}$ subsets in the worst case.

Proof of Theorem 3.5

Let $m = 2k+1$, and consider an algorithm that always uses at most $k-1$ queries. Whenever the algorithm queries a subset of size s, we answer that the subset has value s. Suppose that upon termination the algorithm outputs the allocation

(G_1, G_2), where we assume without loss of generality that $|G_1| \le k$. If $|G_1| < k$, then we assign to every good a value of 1; this is clearly consistent with our answers, but the allocation is not EFX.

Assume, therefore, that $|G_1| = k$. We assign to every good in G_1 a value of 1; this is again consistent with our answers to queries on subsets of G_1. Now, our answers give rise to at most $k-1$ linear constraints on the values of the goods in G_2. Add the constraint that the sum of these $k+1$ values is $k+1$. The allocation (G_1, G_2) is EFX if and only if every good in G_2 has value 1. Since all of our constraints are satisfied when every good has value 1, our set of constraints is satisfiable. There are $k+1$ variables and at most k constraints, so if we compute the reduced row-echelon form of the constraint matrix, we can find a nonempty set of free variables. The remaining variables (i.e., the leading variables) can be written as a linear combination of these free variables. If all free variables are set to 1, all leading variables must also be 1. Hence, we can perturb one of the free variables by a small amount so that all of the leading variables are still nonnegative. This yields a valuation that is consistent with our answers but according to which the allocation returned by the algorithm is not EFX.

Proof of Lemma 4.1

1. By symmetry it suffices to prove that $\min\{u(G)/2, u(L)\} \ge u(R\backslash\{g_r\})$. Suppose that $u(L) < u(R\backslash\{g_r\})$. We have $u(g_r) > 0$ since otherwise we would have moved the cut point to the right. Then $u(L \cup \{g_r\}) > u(L)$ and $u(R\backslash\{g_r\}) > u(L)$; hence $(L \cup \{g_r\}, R\backslash\{g_r\})$ is a more equal partition than (L, R), a contradiction. If $u(R\backslash\{g_r\}) > u(G)/2$ then we also have $u(R\backslash\{g_r\}) > u(L)$, and the same argument yields a contradiction.
2. We use binary search to find the leftmost good g such that $u(L_g \cup \{g\}) \ge u(G)/2$. The left block of the partition will be either L_g or $L_g \cup \{g\}$. Indeed, if the left block is smaller than L_g then having L_g as the left block yields a more equal partition, while if it is larger than $L_g \cup \{g\}$ then having $L_g \cup \{g\}$ as the left block yields a more equal partition. Moreover, with this choice of partition, any good of value 0 next to the cut point will already belong to the block of lower value. The total number of queries required for the binary search is $O(\log m)$.

Proof of Lemma 4.2

We first show that the algorithm computes an EF1 allocation. We consider two cases.

- *Case 1*: At the beginning of Step 3, $u(C) \ge u(B\backslash\{g_2\})$ (so we return the allocation (A, B, C) from Step 2). We have $u(A) \ge u(C) \ge u(B\backslash\{g_2\})$, so a_1 and a_3 do not envy a_2 up to one good. Since $u(B) \ge u(G)/3 \ge u(A) \ge u(C)$, a_2 does not envy a_1 or a_3, and a_1 does not envy a_3. Moreover, if $L_{g_1} \ne \emptyset$, then by definition of g_3 we have $u(A\backslash\{g_3\}) \le u(C)$, which means that a_3 does not envy a_1 up to one good. If $L_{g_1} = \emptyset$ then $A = C = \emptyset$, and again a_3 does not envy a_1 up to one good.
- *Case 2*: At the beginning of Step 3, $u(C) < u(B\backslash\{g_2\})$. We also have $u(C) \le u(A)$. Since we partition $A \cup (B\backslash\{g_2\})$ into two blocks according to Lemma 4.1, both blocks are also worth at least $u(C) = u(C'\backslash\{g_2\})$, meaning that a_1 and a_2 do not envy a_3 up to one good. We claim that a_2 and a_3 also do not envy a_1 up to one good; the claim for a_1 and a_3 towards a_2 can be shown similarly. Let g be the rightmost good in A'. (If $A' = \emptyset$, the claim holds trivially.) It suffices to show that $u(B') \ge u(A'\backslash\{g\})$ and $u(C') \ge u(A'\backslash\{g\})$. By Lemma 4.1, we have $u(B') \ge u(A'\backslash\{g\})$ and

$$\begin{aligned} u(A'\backslash\{g\}) &\le u(A' \cup B')/2 \\ &= \frac{u(G) - u(C')}{2} \\ &\le u(G)/3 \\ &\le u(C'), \end{aligned}$$

Hence the allocation is EF1.

Next, we show that the algorithm can be implemented to use $O(\log m)$ queries. By monotonicity, both finding g_1 and g_2 in Step 1 and finding g_3 in Step 2 can be done by binary search using $O(\log m)$ queries. By Lemma 4.1, the partition in Step 3 can be found using $O(\log m)$ queries. The remaining operations of the algorithm only require a constant number of queries. Hence the total number of queries is $O(\log m)$.

Proof of Lemma 4.3

Before we establish the claim, we show an interesting property of Algorithm 2.

Lemma A.1. *Assume that the goods lie on a line, and let g be a good such that $u(L_g) \ge u(G)/3$ and $u(R_g) \ge u(G)/3$. Then g belongs to the middle bundle in the allocation returned by Algorithm 2.*

Proof. By definition of g_1 in Algorithm 2, we have that g is either g_1 itself or to the right of g_1, so $g \notin A$. Similarly, $g \notin C$. If the allocation (A, B, C) is returned, g belongs to the middle bundle B. Otherwise, we must have $g \ne g_2$, and the algorithm partitions $A' \cup (B'\backslash\{g_2\})$ into two blocks according to Lemma 4.1. The subset of $A' \cup (B'\backslash\{g_2\})$ to the left of g has value at least $u(G)/3$, while the subset to the right of g together with g has value at most $u(G)/3$. It follows that in the resulting partition, g must belong to B', which is the middle bundle in the allocation (A', B', C'). □

We now proceed to establish the lemma. Denote by u the common valuation of the agents, and assume without loss of generality that $u(g_1) \ge u(g_2) \ge u(g_3)$. We consider three cases.

- *Case 1*: There exists a good $g \in \{g_1, g_2, g_3\}$ such that $u(g) \ge u(G)/3$. Then it must be the case that $u(g_1) \ge u(G)/3$. Set $A = \{g_1\}$, arrange the remaining goods in a line with g_2 and g_3 at the two ends, and partition the goods into two contiguous blocks according to

Lemma 4.1. Set B and C to be the two blocks and return the allocation (A, B, C). Clearly, all three goods belong to different bundles. By Lemma 4.1, a_2 and a_3 do not envy each other up to one good. As a_1 receives only one good, a_2 and a_3 do not envy her up to one good. Since $u(A) \geq u(G)/3$, we have $u(B \cup C) \leq 2u(G)/3$. By Lemma 4.1 again, there is a good in a_2's bundle such that if we remove it, then the remaining value of a_2 is at most $u(B \cup C)/2 \leq u(G)/3$. This implies that a_1 does not envy a_2 up to one good. A similar argument holds for a_1 towards a_3.

- *Case 2*: There exists a good $g \notin \{g_1, g_2, g_3\}$ such that $u(g) \geq u(G)/3$. Set $A = \{g_3, g\}$, arrange the remaining goods in a line with g_1 and g_2 at the two ends, and partition the goods into two contiguous blocks according to Lemma 4.1. Set B and C to be the two blocks and return the allocation (A, B, C). Clearly, all three goods belong to different bundles. By Lemma 4.1, a_2 and a_3 do not envy each other up to one good. Moreover, since $u(g_3) \leq u(g_1), u(g_2)$, both agents do not envy a_1 when g is removed from a_1's bundle. A similar argument as in Case 1 shows that a_1 does not envy a_2 or a_3 up to one good.
- *Case 3*: $u(g) < u(G)/3$ for every good g. Arrange the goods in a line starting with g_1 and g_2 at the left and right ends, respectively. Then, keeping g_3 aside, add one good at a time to the left end (to the right of g_1) until the total value of the goods at the left end exceeds $u(G)/3$. Since $\max(u(g_1), u(g_2), u(g_3)) < u(G)/3$, this occurs when we add some good $g \notin \{g_1, g_2, g_3\}$. Add g_3 to the right of g. If $u(L_{g_3}) \geq u(G)/3$ and $u(L_{g_3} \cup \{g_3\}) \leq 2u(G)/3$, add the remaining goods arbitrarily and run Algorithm 2; Lemma A.1 implies that g_3 belongs to the middle bundle of the resulting allocation. Else, $u(\{g_3, g\}) \geq u(G)/3$. Set $A = \{g_3, g\}$ and remove these two goods from the line. Add the remaining goods arbitrarily to the line, and partition the goods into two contiguous blocks according to Lemma 4.1. Set B and C to be the two blocks and return the allocation (A, B, C). A similar argument as in Case 2 shows that the resulting allocation is EF1.

Algorithm 2 uses $O(\log m)$ queries, and by Lemma 4.1, partitioning into two contiguous blocks according to the lemma also uses $O(\log m)$ queries. In Case 3, we can find g by adding all goods except g_3 to the line and using binary search; this takes $O(\log m)$ queries. To determine whether there exists a good g with $u(g) > u(G)/3$, arrange the goods in a line, and use binary search to find the leftmost good g_l such that $u(L_{g_l} \cup \{g_l\}) > u(G)/3$ and the rightmost good g_r such that $u(R_{g_r} \cup \{g_r\}) > u(G)/3$. Such a good g must be one of g_l and g_r. This also takes $O(\log m)$ queries. The remaining operations of the algorithm only requires a constant number of queries. Hence the total number of queries is $O(\log m)$.

Proof of Theorem 4.4

We first show that the algorithm computes an EF1 allocation. We consider three cases. In Cases 2 and 3, assume without loss of generality that T_1 is the first bundle picked from among T_1, T_2, T_3, followed by T_2 and then T_3.

- *Case 1*: The algorithm terminates in Step 1. Both a_2 and a_3 get their favorite bundles and therefore do not envy any other agent, while a_1 does not envy any other agent up to one good no matter which bundle she gets.
- *Case 2*: The algorithm terminates in Step 2 or 3. This means that $u_3(A') \geq u_3(B), u_3(C)$ (either before or after finding large goods). In this case, the allocation is $(C \cup T_2, B \cup T_3, A' \cup T_1)$. Since a_3 gets her favorite bundles A' and T_1, she does not envy any other agent. Next, a_2 gets her favorite bundle B, and the allocation (T_1, T_2, T_3) of T is computed according to her valuation, so she does not envy any other agent up to one good. Furthermore, note that a_3's bundle $A' \cup T_1$ is a subset of A, and a_1 would not envy a_3 up to one good even if a_3 were to get the whole bundle A. Also a_1 prefers T_2 to T_3 and the allocation (A, B, C) is computed according to her valuation, so she does not envy a_2 up to one good.
- *Case 3*: The algorithm terminates in Step 6. Denote by T'_i the bundle among T_1, T_2, T_3 allocated to agent a_i. In this case, the allocation is $(S_1 \cup T'_1, S_2 \cup T'_2, S_3 \cup T'_3)$. Note that any identified large good always remains large. Since $S_2 \cup T'_2 \subseteq A$ and the allocation (A, B, C) is computed according to a_1's valuation, a_1 does not envy a_2 up to one good. Since a_1 prefers T'_1 to T'_3, she also does not envy a_3 up to one good. Next, a_3 prefers S_3 to both S_1 and S_2, and the allocation (T_1, T_2, T_3) of T is computed according to her valuation, so she does not envy any other agent up to one good. If there are fewer than three identified large goods, then T consists of at most two (large) goods. Since a_2 prefers $S_2 \cup T'_2$ to B and C, and both T'_1 and T'_3 contain at most one good, a_2 does not envy any other agent up to one good. Else, each T'_i contains an identified large good; let g be the large good in T'_2. We have $u_2(S_2 \cup \{g\}) \geq u_2(B) \geq u_2(C)$. Moreover, a_2 chooses her favorite bundle from T_1, T_2, T_3 if we were to remove the identified large good from each bundle. Therefore she does not envy any other agent up to one good. Hence the allocation is EF1.

We now show that the algorithm can be implemented to use $O(\log m)$ queries. Step 1 can be done using Algorithm 2 with $O(\log m)$ queries. Step 2 can be done with $O(\log m)$ queries by arranging the goods in A in a line and using binary search to find the leftmost good g such that $u_2(L_g \cup \{g\}) > u_2(B)$, and by using Algorithm 2. Finding a large good in Step 3 can be done similarly using binary search. Step 4 can be done using Lemma 4.3 with $O(\log m)$ queries, and Steps 5 and 6 can be done using a constant number of queries. Hence the total number of queries is $O(\log m)$.

Proof of Theorem 5.1

1. Note that in the envy cycle elimination algorithm (Lipton et al., 2004), it suffices to query the value of each agent for the n bundles in each partial allocation in order to construct the envy graph. Since there are m partial

allocations, this takes $O(nm)$ queries. The cycle elimination step does not require additional queries because the bundles remain the same and the algorithm already knows the value of every agent for every bundle.

2. Fix an ordering of the goods to be allocated, and assume that we allocate them from left to right. Let a_i be an agent with no incoming edge in the envy graph corresponding to the current (partial) allocation. Let g be the leftmost unallocated good such that if we allocate all goods up to and including g to a_i, then the value of some agent for a_i's bundle increases. We then allocate all of these goods to a_i at once. This is a correct implementation of the algorithm because before g is allocated, the value of any agent for any bundle in the partial allocation (and thus also the envy graph) remains unchanged.

   By monotonicity, we can find the good g with $O(n \log m)$ queries using binary search. Since there are n bundles and the value of each agent for each bundle can change up to $k-1$ times, the number of value changes is at most $n^2(k-1)$. Hence the total number of queries is $O(n^3 k \log m)$.

B Omitted Example

Assume that there are 14 goods $g_1, g_2, \ldots, g_{14}$ lying on a line in this order. The common valuation u is such that $u(g_1) = 8$, $u(g_2) = 10$, and $u(g_i) = 1$ for $i = 3, 4, \ldots, 14$. In other words, the values of the goods on the line are

$$8, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1$$

in this order, where there are 12 goods of value 1.

In this example, the left cut point falls within g_2, while the right cut point falls between g_4 and g_5. Assume that the agents a_1, a_2, a_3 receive the left, middle, and right bundle respectively. If we round the left cut point to the left, then a_1 gets value 8 and envies a_3 even after removing any one good. On the other hand, if we round the left cut point to the right, then a_1 has value 18 and a_2 envies her even after removing any good. Note that even if we add the possibilities of rounding one cut point, ignoring the other cut point, and then dividing the remaining goods into two parts according to Lemma 4.1, the same example still shows that none of these additional possibilities works. On the other hand, Algorithm 2 returns the allocation $(\{g_1\}, \{g_2, \ldots, g_6\}, \{g_7, \ldots, g_{14}\})$, which is EF1.

C Additional Results for Any Number of Agents

First, we consider the class of valuations where a subset of larger size is always weakly preferred to a subset of smaller size. In other words, getting a larger bundle cannot make an agent worse off. This class of valuations applies to settings where all of the goods are roughly equally valuable and there are only minor differences within each agent's values for the goods. While this is a rather restricted class of valuations, the valuation used by Plaut and Roughgarden (2018) to show that computing an EFX allocation takes an exponential number of queries in the worst case belongs to this class. The following result says that if we relax the fairness notion to EF1, then it is possible to compute a fair allocation using a much smaller number of queries that does not even depend on the number of goods and is only quadratic in the number of agents.

Theorem C.1. *For any number of agents with monotonic valuations such that $u_i(G_1) \le u_i(G_2)$ for all i and all $G_1, G_2 \subseteq G$ with $|G_1| < |G_2|$, there exists a deterministic algorithm that computes an EF1 allocation using $O(n^2)$ queries in the worst case.*

Proof. Let $q = \lfloor m/n \rfloor$ and $r = m - nq$. Divide the goods arbitrarily into n bundles of q goods and r leftover goods. Order the agents in arbitrary order, and let each of the first $n - r$ agents choose the bundle for which she has the most value among the remaining bundles. Then, give each of the remaining r agents one of the remaining bundles along with one of the r leftover goods. Since we only need to know the value of the agents for at most n bundles, the algorithm can be implemented using $O(n^2)$ queries.

We claim that the resulting allocation is EF1. Indeed, it follows from the assumption on the agents' valuations that no agent envies another agent with fewer or the same number of goods when a good is removed from the latter agent's bundle. Moreover, because of the agents' choices, each agent a_i among the first $n-r$ agents does not envy any agent a_j among the last r agents when the leftover good is removed from a_j's bundle. This completes the proof. □

Next, we consider the setting where the agents have identical monotonic valuations. It is known that an EFX allocation always exists in this setting (Plaut and Roughgarden, 2018). The following lemma shows that if we relax the fairness notion to EF1, we can find a fair allocation that is moreover contiguous. Since contiguity is useful in several situations (see the remark preceding Algorithm 2), the result may also be of independent interest.

Lemma C.2. *Assume that the goods lie on a line. For any number of agents with identical monotonic valuations, there exists a contiguous EF1 allocation.*

Proof. Suppose that the goods lie in the order $g_1, g_2, \ldots, g_m$, and denote by u the common valuation. For any block G' of consecutive goods with g_l and g_r as the leftmost and rightmost goods respectively, let $w(G') = \min\{u(G' \backslash \{g_l\}, G' \backslash \{g_r\})\}$. (If G' is empty, set $w(G') = 0$.) One can check that $w(G_1) \le w(G_2)$ for any two blocks $G_1 \subseteq G_2$. Let S be the set of values of all blocks of consecutive goods (including the empty block, of which we define the value to be 0). For any value $x \in S$ and any $k \in \{1, 2, \ldots, n\}$, define $T_k(x)$ to be the set of all $j \in \{1, 2, \ldots, m\}$ for which there exists k consecutive blocks $G_1, G_2, \ldots, G_k$ starting from the leftmost good g_1 such that $w(G_i) \le x \le u(G_i)$ for all $1 \le i \le k$ and the block G_k ends with the good g_j. Our goal is to show that $m \in T_n(x)$ for some $x \in S$; this will immediately imply the desired result.

We claim that for any x and k, $T_k(x)$ forms a (possibly empty) block of consecutive integers. To prove the claim, we fix x and induct on k. The base case $k = 1$ follows from the observation that both u and w are monotonic. For the inductive step, assume that $T_k(x) = \{t, t+1, \ldots, t+l\}$ for some t and l. Then $T_{k+1}(x)$ consists of all j such that the block G' from g_i to g_j satisfies $w(G') \leq x \leq u(G')$ for some $i \in \{t+1, t+2, \ldots, t+l+1\}$. Hence it suffices to show that if $j < m$ and g_j is the rightmost good such that the inequalities are still satisfied when the block starts at g_i and ends at g_j, then the inequalities are still satisfied when the block starts at g_{i+1} and ends at g_l, for at least one $l \in \{j, j+1\}$. We consider three cases.

- *Case 1*: $j = i$. By definition of j, we have $u(g_{i+1}) > x$. Therefore the block consisting of only g_{i+1} satisfies the inequalities.
- *Case 2*: $j = i + 1$. By definition of j, we have $u(\{g_{i+1}, g_{i+2}\}) > x$. If $u(g_{i+1}) \leq x$, the block starting at g_{i+1} and ending at g_{i+2} satisfies the inequalities; else, the block consisting of only g_{i+1} satisfies the inequalities.
- *Case 3*: $j \geq i + 2$. Denote by G' the (possibly empty) set $\{g_{i+2}, g_{i+3}, \ldots, g_{j-1}\}$. By definition of j, we have $u(G' \cup \{g_i, g_{i+1}, g_j\}) > x$, $u(G' \cup \{g_{i+1}, g_j, g_{j+1}\}) > x$, and $\min\{u(G' \cup \{g_i, g_{i+1}\}), u(G' \cup \{g_{i+1}, g_j\})\} \leq x$. If $u(G' \cup \{g_{i+1}, g_j\}) \geq x$, then since $u(G' \cup \{g_{i+1}\}) \leq \min\{u(G' \cup \{g_i, g_{i+1}\}), u(G' \cup \{g_{i+1}, g_j\})\} \leq x$, the block starting at g_{i+1} and ending at g_j satisfies the inequalities. Else, we have $u(G' \cup \{g_{i+1}, g_j\}) < x$. It follows that the block starting at g_{i+1} and ending at g_{j+1} satisfies the inequalities.

This concludes the inductive step and hence the claim. It also follows that if $j < m$ is the largest element of $T_k(x)$, we can construct k blocks starting at g_1 and ending at g_j all of which satisfies the inequalities for x by greedily taking each block to be the longest block such that the inequalities are satisfied.

Next, we show that for any x and k, if $T_k(x)$ is nonempty and does not contain m, then the intersection of $T_k(x)$ and $T_k(y)$ is nonempty, where y is the smallest element of S larger than x. (Note that y must exist since $T_k(u(G))$ is either empty or contains m.) We prove by induction on k that the largest element of $T_k(x)$ also belongs to $T_k(y)$. For the base case $k = 1$, suppose that the largest element of $T_1(x)$ is i. This means that $u(\{g_1, g_2, \ldots, g_i\}) > x$. By definition of y, we have $u(\{g_1, g_2, \ldots, g_i\}) \geq y$, which implies that $i \in T_1(y)$. For the inductive step, assume that the statement holds for k, and that $T_{k+1}(x)$ is nonempty and does not contain m. Let j be the largest element of $T_{k+1}(x)$. By the remark following the claim, we know that we can greedily construct $k+1$ blocks starting at g_1 and ending at g_j each of which satisfies the inequalities for x. In particular, the kth block will end at g_i, where i is the largest element of $T_k(x)$. By the inductive hypothesis, $i \in T_k(y)$. A similar argument to the one used in the base case shows that $j \in T_{k+1}(y)$, as claimed.

Finally, note that $T_n(0)$ is nonempty. If $m \notin T_n(x)$ for all $x \in S$, the previous paragraph implies that $T_n(x)$ is nonempty for all x. But this is impossible since $T_n(u(G))$ is either empty or contains m, so it must be that $m \in T_n(x)$ for at least one x, as desired. □

Like Lemma 4.2, Lemma C.2 guarantees the existence of an EF1 allocation with the extra property that if some agent envies another agent, then the envy can be eliminated by removing one of the goods at the end of the latter agent's block.

We now leverage Lemma C.2 to show that for agents with identical monotonic valuations, it is possible to compute an EF1 allocation that is moreover contiguous using a number of queries that depends only logarithmically on the number of goods. For this result we assume that the value of an agent for any subset of goods is an integer that is at most some value K. This is a realistic assumption for practical purposes; for instance, Spliddit lets users specify their value for each good as an integer between 0 and 1000.

Theorem C.3. *Assume that the goods lie on a line. For any number of agents with identical monotonic valuations such that the value of an agent for any subset of goods is an integer at most K, there exists a deterministic algorithm that computes a contiguous EF1 allocation using $O(n \log m(n \log m + \log K))$ queries in the worst case.*

Proof. Using the notation from the proof of Lemma C.2, we know that $m \in T_n(x)$ for some $x \in S$. Since $S \subseteq \{0, 1, 2, \ldots, K\}$, we may take S to be this set. By monotonicity, we can use binary search to find x such that $m \in T_n(x)$. For each value of x, we try to construct k blocks that satisfy the inequality for x greedily in two ways, one by taking as few goods as possible for each block, and the other by taking as many goods as possible. If the former construction does not take all goods by the kth block while the latter construction does not leave enough goods for the kth block, we know that this value of x works. The existence of such x is guaranteed by Lemma C.2. Using binary search allows us to try $O(\log K)$ values of x, and for each value of x we make $O(n \log m)$ queries. Hence the step of finding x takes $O(n \log m \log K)$ queries.

Once we have x, it remains to construct the n blocks $G_1, G_2, \ldots, G_n$ that satisfy the inequalities $w(G_i) \leq x \leq u(G_i)$ for all i. If we have constructed $j - 1$ blocks, we use binary search to find the jth block such that the remaining blocks can also be constructed to satisfy the inequalities. Again, we can check whether the remaining blocks can be constructed by trying to construct the blocks greedily in two ways; this takes $O(n \log m)$ queries. Since we construct n blocks and we use binary search to determine each block, the total number of queries in this step is $O(n^2 \log^2 m)$. Combining the queries in the two steps yields the desired result. □

In particular, if n and K are constant, the bound in Theorem C.3 becomes $O(\log^2 m)$.