Several Stories About High-Multiplicity EFx Allocation Technical appendix

A Hardness of EFx + PO problem

We reduce the NP-complete problem PARTITION [3], to the problem of finding an EFx + PO allocation, thus demonstrating the hardness of the latter one.

> PARTITION
> **Input:** A set of positive integers $S = s_1, \ldots, s_n$.
> **Task:** Does there exist a partition of S into two sets S_1 and S_2 such that the sums of the numbers in the sets are equal?

Theorem 1. *The problem of existence of an* EFx + PO *allocation is* NP*-hard.*

Proof. We construct an input for the problem of finding an EFx + PO allocation with two agents from the input of the PARTITION problem. For each number s_i in the PARTITION problem, we create an item g_i that has a value of $10 \cdot s_i$ for each player. We also introduce two additional items b_1 and b_2, such that the first agent values b_1 as 1 and b_2 as 0, while the second agent values them in the opposite way (see Table 1).

Agents \ Objects	$\mathbf{g_1}$	$\mathbf{g_2}$	$\mathbf{g_3}$	$\ldots$	$\mathbf{g_n}$	$\mathbf{b_1}$	$\mathbf{b_2}$
1	$10s_1$	$10s_2$	$10s_3$	$\ldots$	$10s_n$	1	0
2	$10s_1$	$10s_2$	$10s_3$	$\ldots$	$10s_n$	0	1

Table 1: The input for the EFx + PO problem

If there exists a solution for PARTITION $S = S_1 \sqcup S_2$, then consider the following allocation: the first player's bundle will be $A_1 = S_1 \cup b_1$, and the

second player's bundle will be $A_2 = S_2 \cup b_2$. It is easy to see that

$$v_1(A_1) + \min_{g \in A_2} v_1(g) = (v_1(S_1) + 1) + 0 \geq v_1(S_2),$$

$$v_2(A_2) + \min_{g \in A_1} v_2(g) = (v_2(S_2) + 1) + 0 \geq v_2(S_1).$$

Thus, this allocation is EFx, but it is also PO because the items b_1 and b_2 are assigned to players who value these items not as 0. Moreover, any partition of the items g_i is PO since both players have the same valuations for these items.

Conversely, if there exists an EFx + PO allocation for a given input, then in such allocation, the item b_1 must necessarily be assigned to the first player and the item b_2 to the second player; otherwise, the solution would not be Pareto-optimal. Let us write this allocation in general form as $A_1 = \widetilde{S_1} \cup b_1$ and $A_2 = \widetilde{S_2} \cup b_2$, where $S = \widetilde{S_1} \sqcup \widetilde{S_2}$. Then, due to the fact that the allocation satisfies the EFx property, the following inequalities hold:

$$(v_1(\widetilde{S_1}) + 1) + 0 = v_1(A_1) + \min_{g \in A_2} v_1(g) \geq v_1(A_2) = v_1(\widetilde{S_2}),$$

$$(v_2(\widetilde{S_2}) + 1) + 0 = v_2(A_2) + \min_{g \in A_1} v_2(g) \geq v_2(A_1) = v_2(\widetilde{S_1}).$$

We know that the values of each item g_i are multiplied by 10 for both players, which means that the inequalities written above are strict, since the digits in the units place are 1 on the left-hand side and 0 on the right-hand side. Therefore, if we remove 1 from the left-hand sides, both inequalities will still hold but become non-strict. Thus, we have:

$$\begin{cases} v_1(\widetilde{S_1}) \geq v_1(\widetilde{S_2}) \\ v_2(\widetilde{S_2}) \geq v_2(\widetilde{S_1}) \end{cases} \implies \sum_{s_1 \in \widetilde{S_1}} s_1 = \sum_{s_2 \in \widetilde{S_2}} s_2.$$

The last step is true because $v_1(\widetilde{S_1}) = v_2(\widetilde{S_1})$ and $v_1(\widetilde{S_2}) = v_2(\widetilde{S_2})$. Thus, the sets $\widetilde{S_1}$ and $\widetilde{S_2}$ are the solution for the Partition problem.

□

Note that the Partition problem is W[1]-hard parameterized by the size of the minimum set $\min(|S_1|, |S_2|)$ [4], so we can make the following remark.

Corollary 1. *The problem of existence of an* EFx + PO *allocation* W[1]*-hard parameterized by the size of the minimum bundle* $\min(|A_1|, \ldots, |A_m|)$.

B Integer Linear Programming

Theorem 2. *The problem of searching for* EFx + PO *allocation allows for an* FPT*-algorithm based on the number of agents n, the number of* EFx *allocations s, and the number of item types k. Its running time is:*

$$O^*((2snk + 4kn^2)^{5snk+10kn^2} s)$$

The proof of this theorem will be presented next, for convenience, it is divided into parts. Section B.1 describes the method for searching EFx allocations, followed by a description of how to check an EFx allocation for PO, and the proof is concluded by introducing additional constraints in the ILP problem necessary to ensure that the search method does not return previously checked allocations. The asymptotics of the algorithm's operation depend on the number of variables in the problem, so at the end of section B.3 we estimate their number, thereby obtaining the final assessment of the runtime.

B.1 EFx-allocation search

The algorithm's operation is divided into stages, the first of which is the search for the EFx allocation. Building on previous work, we decided to reduce this to solving an integer linear programming problem. The fairness conditions can be rewritten as:

$$0 \leq x_i^a \leq m_i, \; \forall a \in N \; , \; \forall i \in K, \tag{1}$$

$$\sum_{a \in N} x_i^a = m_i, \; \forall i \in K, \tag{2}$$

$$\sum_{i=1}^{k} u_i^a x_i^a + \min_{j:\, x_j^b > 0} u_j^a \geq \sum_{i=1}^{k} u_i^a x_i^b, \; \forall a, b \in N. \tag{3}$$

Here, the set K is the set of types; the variable x_i^a represents the quantity of items of type i given to agent a; m_i is the number of items of type i; u_i^a denotes the value of the i-th type for agent a. Equation (1) constrains x_i^a by the total number of items of type i, equation (2) ensures that each item is assigned to someone, and equation (3) is the fairness condition for any pair of agents. Since the ILP approach is quite frequently used in the field, a similar condition is found in many papers (e.g., [2]). Our own results and suggestions are described below. It's important to note that this method

can also be applied to the EF property and its various modifications. Simply replace equation (3) with the condition from another definition of fairness; nothing else needs to be changed.

The above conditions can be simplified for the case of two agents:

$$0 \leq x_i \leq m_i, \ \forall i \in K. \tag{4}$$

$$\sum_{i=1}^{k} a_i x_i + \min_{j:\, m_j - x_j \neq 0} a_j \geq \sum_{i=1}^{k} a_i (m_i - x_i). \tag{5}$$

$$\sum_{i=1}^{k} b_i (m_i - x_i) + \min_{j:\, x_j \neq 0} b_j \geq \sum_{i=1}^{k} b_i x_i. \tag{6}$$

In this case, x_i is the quantity of items of type i for the first agent, a_i is the value of the item of type i for the first agent, and b_i is for the second. Such a transition is possible since (in terms from the general form) $x_i^1 = m_i - x_i^2$. This allows for a reduction in the number of variables.

We only need to understand what to do with the minimum condition that arises in inequalities (5), (6). In fact, we can eliminate it by adding the following conditions (similarly for the second agent):

$$\sum_{i=1}^{k} a_i x_i + a_j \geq \sum_{i=1}^{k} a_i (m_i - x_i), \ \forall j \in M : \ m_j - x_j \neq 0.$$

To identify j for which $m_j - x_j \neq 0$, we introduce two new binary variables for each item: l_j, r_j. If $l_j = 1$, then $x_j \in [0, m_j - 1]$, otherwise $r_j = 1$ and $x_j = m_j$. Thus, the equations for each item look as follows:

$$0 \leq l_j, r_j \leq 1.$$
$$l_j + r_j = 1.$$
$$x_j - m_j \leq -l_j.$$
$$x_j - m_j \geq -m_j \cdot l_j.$$

It can be noted that in the ILP conditions, this constraints set exactly the necessary conditions – only one variable from l_j and r_j is equal to one and indicates the membership of the corresponding part of the set $[0, m_j]$. Similar conditions allow defining the variables z_j and z_j^+, which correspond to $x_i = 0$ and $x_i > 0$, respectively.

Thus, the constraints for both agents can be written for any item j as:

$$\sum_{i=1}^{k} a_i x_i + l_j \cdot a_j + r_j \cdot max\ weight \cdot k \geq \sum_{i=1}^{k} a_i (m_i - x_i)$$

$$\sum_{i=1}^{k} b_i (m_i - x_i) + z_j^+ \cdot b_j + z_j \cdot max\ weight \cdot k \geq \sum_{i=1}^{k} b_i x_i$$

To each sum, two terms were added: if l_j or z_j^+ are equal to 1, then we get the usual equation from the constraints above; otherwise, we add a number guaranteed to be larger than the right-hand side to not reduce the solution space. The number chosen for this purpose is $max\,weight \cdot k$ (where $max\ weight$ – is the maximum item weight in the input data) since there are k terms on the right side that do not exceed $max\ weight$ in value.

After this mapping, the problem can be solved by ILP solvers. The well-known Lenstra algorithm is FPT based on the number of variables, and its runtime is $O(p^{2.5p} \cdot L)$ where p is the number of variables and L is the length of the input (in bits). The number of variables in the resulting integer programming problem, in turn, polynomially depends on n and k. Each obtained solution will be checked for Pareto optimality after solving the ILP, described in the next section. If it is not suitable, we add a constraint that prevents the solver from returning the same EFx allocation and continue the search.

B.2 Pareto-optimality check

Since the general condition for EFx + PO-allocation would be computationally challenging, we divided the task into two parts. Each found EFx allocation should be checked for Pareto optimality. Initially, we reduced the task to ILP, introducing variables and maximizing them based on the possible number of items for the first and second agents. This approach allowed us to get a variable, the sign of which determined whether the allocation is PO. However, the mentioned maximization turned out to be difficult to implement, so we decided to use another approach.

We reduce the task of checking the Pareto optimality of the allocation to ILP in the same way as in the article [1]. When we get an EFx allocation, we get to know all x_i^a. Then solving the following task will return the allocation y, which strictly dominates the found EFx allocation:

$$\sum_{i \in K} u_i^a \cdot y_i^a \geq \sum_{i \in K} u_i^a \cdot x_i^a, \; \forall a \in N. \tag{7}$$

$$\sum_{i \in K} \sum_{a \in N} u_i^a \cdot y_i^a \geq 1 + \sum_{i \in K} \sum_{a \in N} u_i^a \cdot x_i^a. \tag{8}$$

$$0 \leq y_i^a \leq m_i, \; \forall a \in N, \forall i \in K. \tag{9}$$

$$\sum_{a \in N} y_i^a \leq m_i \tag{10}$$

Inequality (7) is a condition that the set of each agent in the allocation y is valued no less, (8) is that the total value has increased, i.e., there is an agent in which the inequality was strict, (9) and (10) are the conditions that y is an allocation.

B.3 EFx restrictions

In the case when the EFx allocation turns out to be not PO, constraints need to be imposed so that the solver doesn't return it again. To this end, we add new conditions to the original problem from B.1. Let the solver return the allocation y, we checked it for Pareto optimality and it didn't fit. Then, for each y_a^i, we introduce two binary variables $d_{i,a}^-$ and $d_{i,a}^+$ and introduce the following conditions:

$$x_i^a \leq y_i^a + (m_i + 1)(1 - d_{i,a}^+). \tag{11}$$

$$y_i^a \leq x_i^a - 1 + (m_i + 1)d_{i,a}^+. \tag{12}$$

$$y_i^a \leq x_i^a + (m_i + 1)(1 - d_{i,a}^-). \tag{13}$$

$$x_i^a \leq y_i^a - 1 + (m_i + 1)d_{i,a}^-. \tag{14}$$

Equations (11) and (12) account for the condition $x_i^a \leq y_i^a$. If this is met, then $d_{i,a}^+ = 1$, otherwise 0. Similarly, equations (13) and (14) lead to $d_{i,a}^- = 1 \Leftrightarrow y_i^a \leq x_i^a$. It follows that in the case of equality $x_i^a = y_i^a$, we have $d_{i,a}^- + d_{i,a}^+ = 2$. Since we don't want a complete match, we add the last constraint:

$$\sum_{i \in K} \sum_{a \in N} d_{i,a}^- + d_{i,a}^+ \leq 2kn - 1.$$

On the first run, there are kn variables corresponding to the number of items of each type for each agent, and $4kn^2$ variables for if-conditions.

As we add no more than $2kn$ variables at each step, in every search for EFx allocation we have at most $(2s+1)kn + 4kn^2$ variables, allowing us to obtain the asymptotics from theorem 2, since the number of EFx allocation searches is no more than s. It is worth noting that checking the obtained allocations for PO doesn't affect the asymptotics since it is invoked no more than s times and only requires kn variables.

C Number of EFx allocations

Lemma 1. *The total number of allocations in the problem with m objects and two agents does not exceed $(\lceil \frac{m+k}{k} \rceil)^k$, where k is the number of different types of items.*

Proof. Let m_i be the number of objects of type i. Then the total number of allocations is equal to

$$\prod_{i=1}^{k}(m_i + 1),$$

since the first agent can have from 0 to m_i objects of type i. It is a known fact that the maximum product under a fixed sum occurs when the numbers are equal.

In this case, each of the agents receives no more than $\lceil \frac{m}{k} \rceil + 1$ objects of each type, and the product does not exceed

$$(\lceil \frac{m+k}{k} \rceil)^k.$$

□

Note that in the general case (for n agents) the upper bound is $(\lceil \frac{m+k}{k} \rceil)^{k(n-1)}$. The following lemma provides a construction for an example.

Example 1. *The number of* EFx*-allocations can be equal to $\frac{m^k}{2^k k^k}$ on some inputs with two agents.*

Proof. Consider a problem with two agents and k types of items. Let the first type have value a_1 for the second player and 0 for the first player, and for all other types, the i-th item has value a_i for the first player and 0 for the second player. Let m_i be the number of items of type i and x_i be the number of items of type i held by the first player.

We write the EFx condition for both players as follows:

$$a_2x_2 + ... + a_kx_k + \min_{j:x_j \neq m_j} a_j \geq (m_2 - x_2)a_2 + ... + (m_k - x_k)a_k \quad (15)$$

$$(m_1 - x_1)a_1 \geq x_1a_1 \quad (16)$$

In the second condition, there is no minimum because it is equal to zero (except in the case when all items, except for the first type, are given to the second player, but this will only weaken the constraints). It is easy to see that both conditions are satisfied when $x_1 \leq \frac{m_1}{2}$ and $x_i \geq \frac{m_i}{2}$ if $i \geq 2$. In this case, the number of EFx-allocations satisfying the condition at least $\frac{m^k}{2^k k^k}$.

□

D Experiments

Since one of the generation parameters is the maximum weight, we generated two datasets of 5000 samples each with the only difference in parameters being $\max \text{weight} = 10$ in one and $\max \text{weight} = 1000$ in the other. Due to computational constraints, the enumeration took place for $k \in [2, 5]$, $m \in [2, 23]$ (for smaller k, $m \in [2, 31]$).

The following metrics were collected for display in the charts:

- average number of EFx allocations,
- 99% quantile of the number of EFx allocations,
- the maximum estimate from Lemma 1.

The results are shown in figs. 1 and 2:

The estimate has a stepwise appearance due to rounding up in the formula. At a glance, there's a significant difference between the average number of allocations and even the 99% quantile from the upper estimate, and the relative deviation only increases with the increasing number of types.

To study the impact of the weight constraint on the number of allocations, we introduced a new metric called relative difference. This is the ratio of the difference in the average number of solutions to the overall average number of solutions:

$$\text{relative difference} = \frac{\overline{solutions}_1 - \overline{solutions}_2}{\overline{solutions}},$$

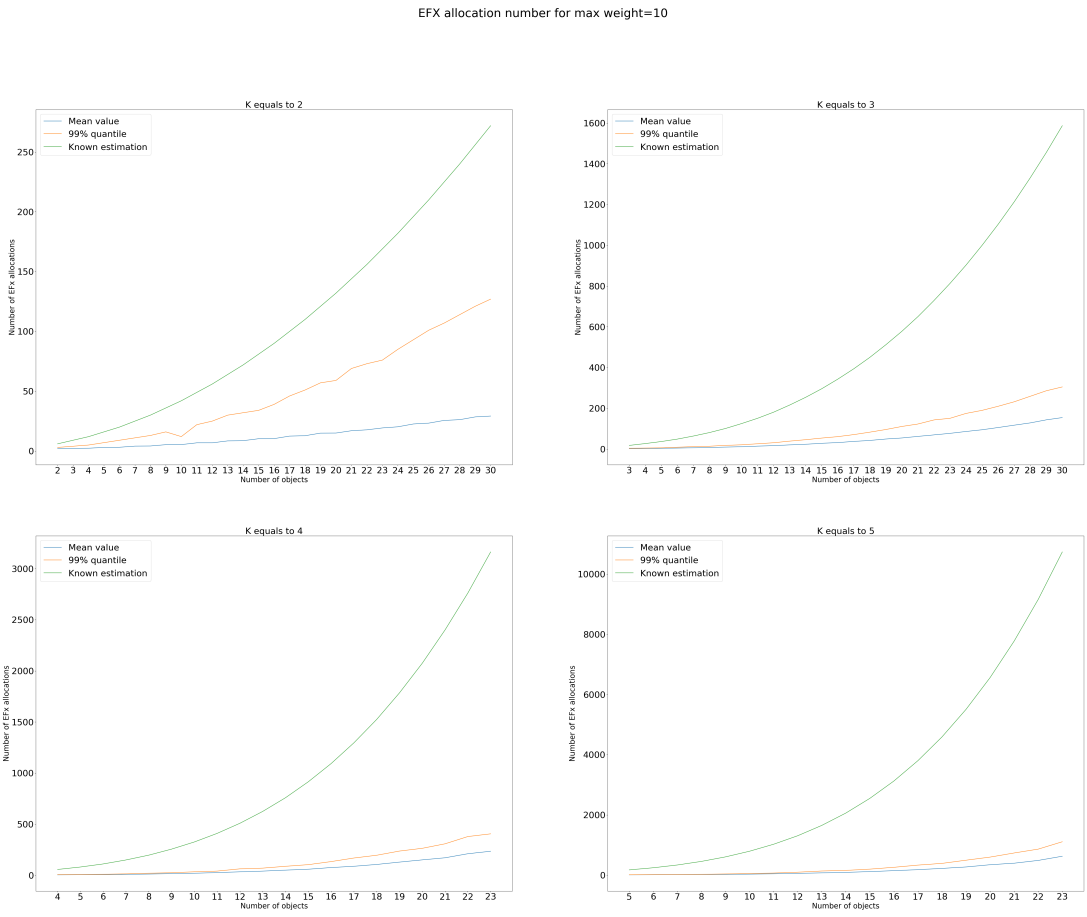

Figure 1: Number of EFx allocations for $max\ weight = 10$

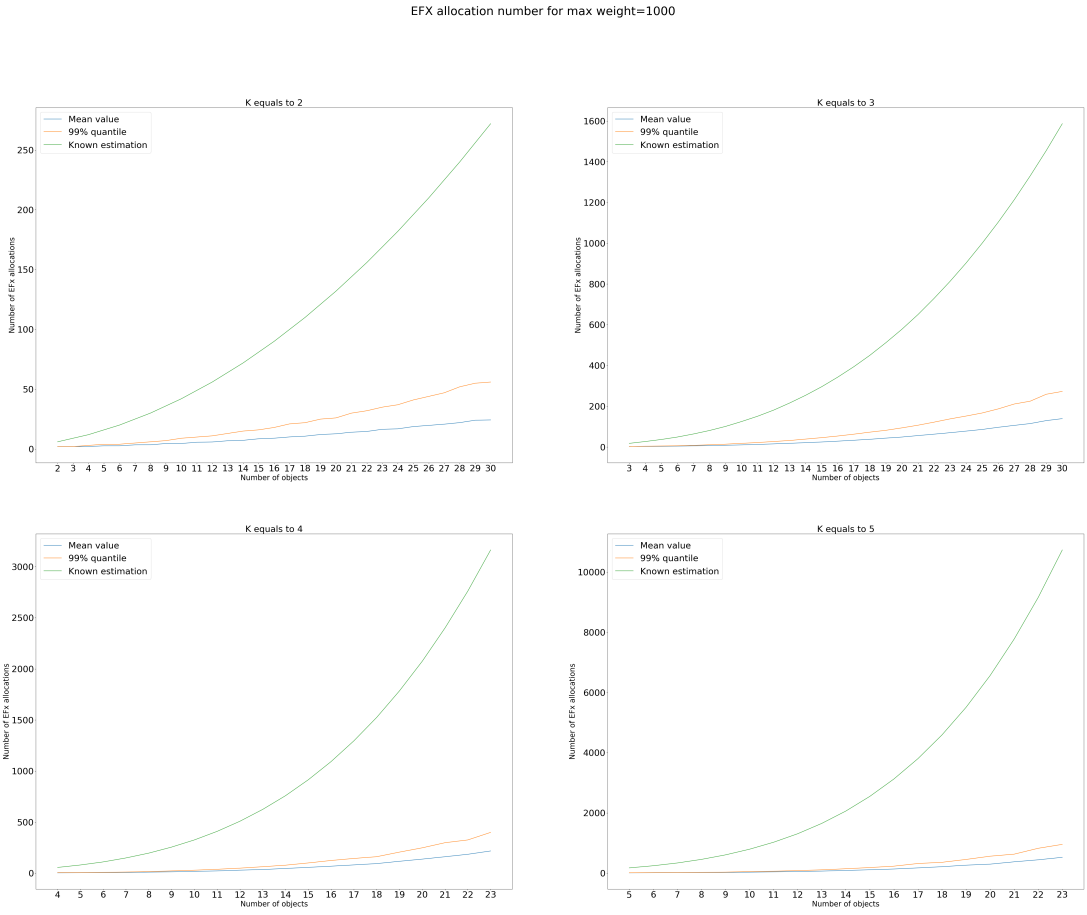

Figure 2: Number of EFx allocations for $max\ weight = 1000$

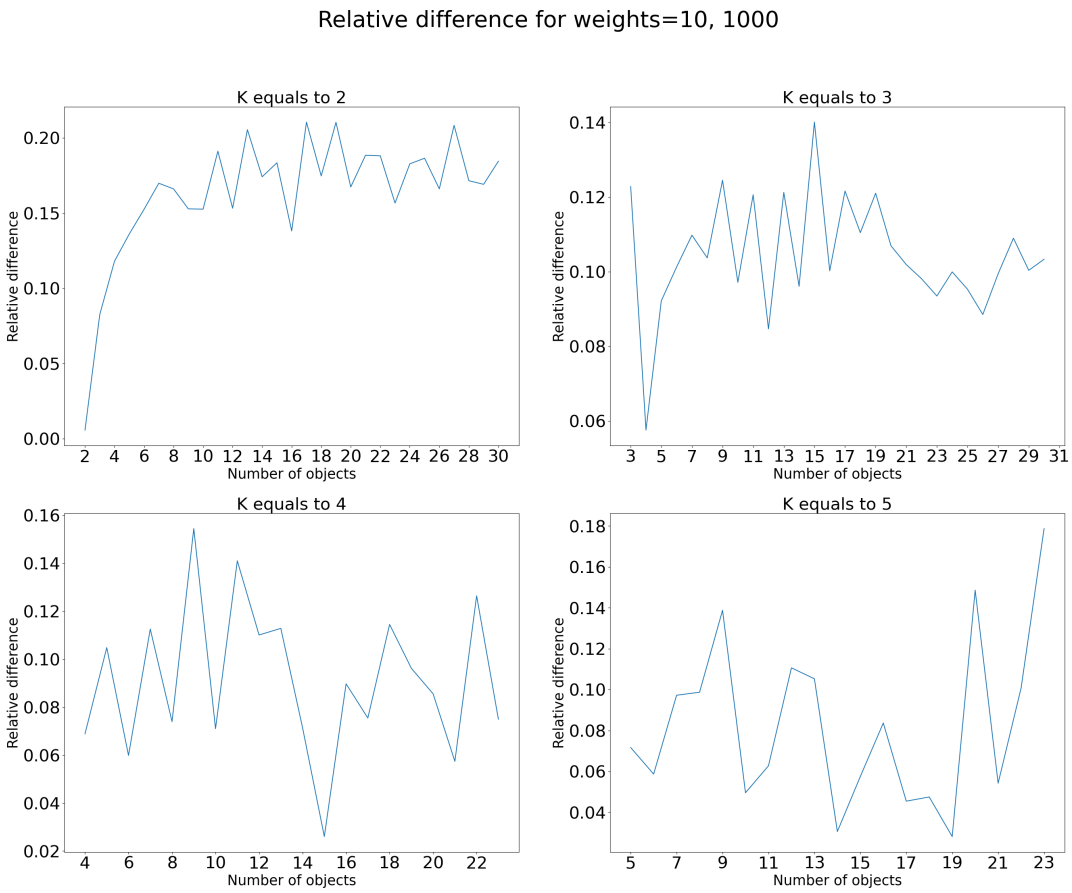

Figure 3: Relative difference for weights $10, 1000$

where $\overline{solutions}_1$ is the average number of solutions in the first sample, and $\overline{solutions}_2$ is in the second. In our case, both datasets are of equal size, so:

$$\overline{solutions} = \frac{\overline{solutions}_1 + \overline{solutions}_2}{2}.$$

It can be observed that as the number of types increases, the impact of weight differences decreases. We believe this is due to the fact that, with a small number of types, the weight difference plays a significant role that cannot be offset by the size of the sets. For instance, there are only 2 EFx allocations for an input of 2 types and 12 items: $\{(1, 1) : 11, (1000, 1000) : 1\}$

(allocations and bundles are described in the format "type — number of items of this type"). If an agent's bundle is: $\{(1,1) : a, (1000,1000) : b\}$, where $a, b > 0$, they would be envied because the minimum condition from the definition of EFx will not be met. However, by changing the boundary from 1000 to 10, a similar input becomes $\{(1,1) : 11, (10,10) : 1\}$ and the number of allocations of interest to us increases.

This effect is present when the number of types and items is small and disappears as these parameters increase. If you sort the values of items for each player and choose the difference between neighboring values, the maximum value of this difference will be the cause of the observed effect. As the number of types increases, the variance of this metric decreases, hence the effect disappears. The average number of allocations decreases for similar reasons (e.g., for 2 types and 12 items, the number of EFx allocations with a weight limit of 10 is twice as large).

About the ratio EFx **and** PO**.** Also indirectly the complexity of our algorithm for finding EFx + PO based on ILP is affected by the *percentage of* EFx *allocations that are* PO (denote this percentage as p). If the ILP solver returns solutions equiprobably, then with probability p our algorithm stops at the first found EFx allocation. Several experiments on this topic can be found in the Jupyter Notebook.

References

[1] R. Bredereck, A. Figiel, A. Kaczmarczyk, D. Knop, and R. Niedermeier, *High-multiplicity fair allocation made more practical*, in Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '21, Richland, SC, 2021, International Foundation for Autonomous Agents and Multiagent Systems, p. 260–268.

[2] R. Bredereck, A. Kaczmarczyk, D. Knop, and R. Niedermeier, *High-multiplicity fair allocation: Lenstra empowered by n-fold integer programming*, in Proceedings of the 2019 ACM Conference on Economics and Computation, EC '19, New York, NY, USA, 2019, Association for Computing Machinery, p. 505–523.

[3] M. E. Dyer, A. M. Frieze, R. Kannan, A. Kapoor, L. Perkovic, and U. V. Vazirani, *A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem*, Comb. Probab. Comput., 2 (1993), pp. 271–284.

[4] M. R. Fellows and N. Koblitz, *Fixed-parameter complexity and cryptography*, in Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, G. Cohen, T. Mora, and O. Moreno, eds., Berlin, Heidelberg, 1993, Springer Berlin Heidelberg, pp. 121–131.