On Fair Division under Heterogeneous Matroid Constraints

by Abed El Kareem Massarwa

advised by Prof. Erel Segal-Halevi

August 2024

Contents

Chapter 1: Summary

In this research paper we focus mainly on fair allocation of items which are indivisible (cant be broken into pieces) such as (books, cars, etc. …) among people with different preferences/capabilities
(capacities), for example : handing different tasks among a team of developers in a tech company , some may prefer front-end ,others may prefer back-end, some may have plenty of time to work on it , others may have other responsibilities in life 😊

our main goal is to design algorithms to handle such cases.

The main motivation lies in the fact that such scenarios are practical and we face them in our daily life !.

As part of the obstacles we faced the fact when it comes to indivisible goods its hard to achieve a perfect EF (Envy-Free) allocation, meaning everyone is satisfied completely with the bundle of items he got.

Hence we focus on the term EF-1 (Envy-Free up to 1 item/good)
meaning every agent is satisfied with what he got on one condition, which says 'I'm satisfied but u have to take down 1 item from agent i' , in which agent i is the agent our agent envies

Researchers in this paper focus on bundles which are form independent-sets in the matroid (explained later)

There were mention-worthy previous conducted research papers in this field which led to various results :

1. Biswas and Barman (2018, 2019): They dealt with additive valuation (which we use extensively in our research) but failed to find EF-1 under different or identical matroid constraints given different valuations.

2. Lipton, Markakis, Mossel, & Saberi (2004): Found that it is possible to efficiently achieve EF-1 but without constraints.

3. Oxley (2006): Took a slightly different approach from what we will discuss in this research. They focused on submodular valuation, meaning the value of a particular bundle is not the sum of the values of each item but the best value of a sub-bundle. In 2020, other researchers improved this to a binary valuation (0 for not wanting an item, 1 for wanting it). However, the problem with such a definition is the removal of items after allocation, which is not suitable in our case, as we want a complete allocation that includes all items.

An open problem is the existence of complete EF-1 allocations among heterogeneous agents when the heterogeneity is both in the agents' capacities and their valuations.

Impossibilities

The researchers encountered certain scenarios where it is definitively impossible to achieve certain outcomes, such as:

1. **Different Partition Constraints for Each Agent:** This makes finding F-EF1 (Feasible Envy-Free up to one item) impossible.
2. **Going Beyond Matroid Constraints:** This does not contribute to the goal.
3. **Stronger Definition than EF-1 (EFX):** Moving to a stronger definition than EF-1 (EFX) is also not helpful.

As a result, the researchers focus on finding F-EF1 allocations under the following conditions:

- **Identical Matroid Constraints with Different Capacities:** Matroids that are identical in structure but allow for different capacities for each agent.

- **BO Matroids (Beyond Orderable):** Where all agents have identical constraints, though different valuations may be allowed.

Brief Overview of the Algorithms to Follow:

1. The challenge in the research of Barman and others mentioned earlier is that they could not handle different capacities for each agent because the process of cycle removal in an envy graph is not guaranteed. This is because it involves exchanging items between agents, which may not satisfy the constraints. Therefore, in this research, the focus is on definitions that altogether prevent the situation where a cycle exists in the graph, saving us from being unsure whether we have violated the constraints.
2. **BO Matroids:** This is a broad class of matroids to which most matroids belong. In this class, the researchers will develop an algorithm in Chapter 8 of the paper under identical constraints, possibly with different valuations.

Related Research:

There are several different studies related to this paper. We will briefly present them and highlight the differences from our work. The various studies are categorized according to different constraints:

1. **Balanced Allocation:** This is an allocation where all agents receive the same number of items, give or take one extra item.
2. **Capacity Constraints:**
    - Some papers explored different aspects, such as (Garg et al. 2010; Long, Wong, Peng, & Ye 2013; Lian, Mattei, Noble, & Walsh 2018).
    - The constraints may differ for each agent, but there is only one category.

- **Ferraioli, Gourvès, and Monnot (2014):** Each agent must receive K items.
- **Round Robin Algorithm:** Finds EF-1 allocations for any number of agents with additive valuations.
- **Kyropoulou, Suksompong:** They have an algorithm that finds an EF-1 allocation for two agents with monotone valuations.
- **Gafni, Huang, Lavi, and Talgam-Cohen (2021):** Explored fair allocation under related definitions where resources may be duplicated.
- Some explored fair allocation under a ranking definition.

3. **Matroid Constraints:**
   - **Gourvès, Monnot, and Tlilane (2014); Gourvès and Monnot (2019):** Require that the union of the bundles allocated to all agents be an independent set of the matroid. This requires leaving some items unallocated, which is not allowed under our environmental conditions.

4. **Budget Constraints:**
   - Some researchers explored budget constraints for each agent, requiring that the possible bundles meet the condition that the sum of their prices is less than or equal to the agent's budget.

5. **Downward Closed Constraints:**
   - **Li and Vetta (2021):** Studied fair allocation with downward-closed constraints, including budget constraints, within the general matroid.

6. **Conflict-Free Constraints:**
   - **Li, Li, and Zhang (2021):** Studied fair allocation with scheduling constraints, where each resource is treated as a time interval, and possible bundles are those without conflicting resources.
   - **Hummel and Hetland (2022):** Explored this more generally through a "conflict graph," where each possible bundle does not include two adjacent nodes in the graph (sharing the same edge).

7. **Connectivity Constraints:**

- **Barrera, Nyman, Ruiz, Su, and Zhang (2015); Bil'o, Caragiannis, Flammini, Igarashi, Monaco, Peters, Vinci, and Zwicker (2018); Suksompong (2019):** Studied fair division under a different definition, where resources are arranged in a sequence, and each agent is supposed to receive a connected sub-sequence in this order.
- **Bouveret, Cechlarová, Elkind, Igarashi, and Peters (2017); Bei, Igarashi, Lu, and Suksompong (2019):** Studied something similar, but this time under the constraint of receiving a subgraph of the overall graph.

8. **Non-Additive Constraints:**
   - **Bei, Garg, Hoefer, and Mehlhorn (2017); Anari, Mai Gharan, and Vazirani (2018):** Studied the allocation of items divided into models, where each model has a quantity under additive constraints between the models, but marginal returns constraints between the units (meaning as we take more of the same model, the valuation decreases).
   - **Garg, Hoefer, and Mehlhorn (2018):** Studied additive-budgetary valuation where each agent has a certain limit, and the valuation for bundles is defined as MIN(value of the limited value and additive value of the items in the bundle), meaning it's not worth taking more items because, in their eyes, it holds the same value.
   - Some also explored submodular valuation (adding items to a smaller group may have a more positive impact on valuation).

Definitions and Concepts

Allocations and Constraints:

Let M be the set of items to be allocated among agents N . An allocation is denoted as $X = (X_1, \dots, X_n)$, where X_i is the allocation for agent i . Each allocation X_i is a subset of M , and we ensure that the intersection between any two agents' allocations is empty, meaning there are no duplicate items.

An allocation is called a ****complete allocation**** if the union of all agents' allocations equals M, meaning we have successfully divided all items with nothing left over. In the remainder of the paper, the symbol $[n]$ represents $\{1, \ldots, n\}$.

In this paper, we focus on constrained definitions under what is called a ****matroid**.** Each agent will have their own matroid, through which suitable bundles are defined for them to take and be satisfied with.

Definition 1: Matroid

A **matroid** is an ordered pair $M = (M, I)$where:

- Mis the set of items.
- I is a subset of the power set of M, which is non-empty and contains independent sets that satisfy the following properties:
    1. Every subset of an independent set is also in I.
    2. There exists g such that for any two different sets of the same size, the union with the smaller set is also in I.

Partition Matroid:
As the name suggests, this matroid specifically refers to the division of items in M into categories, where each category characterizes the agent i's ability to hold items from it. This restricts I to only contain bundles that satisfy the condition that the number of items from each category does not exceed the maximum defined for it.

Uniform Matroid:

This is essentially a special case of a partition matroid with only one category

Feasible Allocation:

A **feasible allocation** is an allocation $X = (X_1, \ldots, X_n)$ such that:

1. X_i belongs to I of the agent.

2. The allocation is complete, meaning the union of all agents' allocations equals M

Valuations and Fairness Terms:

An **additive valuation** is, as the name implies, the valuation for a specific bundle as the sum of the valuations of each item in the bundle according to agent i.

Envy and Envy-Freeness:

Agent i is said to envy agent j if and only if $v_i(X_i) < v_i(X_j)$.

Maximal Feasible Subset:

Defined as the set of sub-bundles of bundle T whose value is maximal (there may be more than one subset with the MAX value). This definition is useful in the case of $F - EF1$.

Feasible Valuation:

Simply defined as $V_i(\text{best feasible subset})$ as found in the previous definition.

Given an allocation X:

- Agent i is F-envy-free with respect to agent j if $\hat{v}_i(X_i) \geq \hat{v}_i(X_j \setminus Y)$, where $Y \subseteq X_j$ and the size of Y is at most 1.
- **Feasible Envy-Free (F-EF1):** There is no agent who envies another up to one item.

Positive Feasible Envy:

Defined as $X(i \to j) := \max(0, \hat{v}_i(X_j) - \hat{v}_i(X_i))$. If the difference is negative, there is no envy, so we take 0.

Envy Graph:

An envy graph is a graph where the nodes represent agents, and there is an edge from node i to node j if and only if $v_i(X_i) < v_i(X_j)$.

Pareto Efficient Allocation:

This is an allocation that cannot be Pareto improved, meaning there is no better allocation where the new values for all agents' bundles are greater than or equal to the values of the bundles in the current allocation.

Max-Nash-Welfare:

Nash-welfare is a function applied to a feasible allocation that calculates the root of the product of the valuations of each agent for their bundle. The maximum Nash-welfare is simply the allocation that maximizes the result of this function.

Algorithms:

Algorithm 1: Per Category Round Robin As its name suggests, the algorithm relies on looping over the categories, which can be problematic if the input is a set of agents with the same division constraint but not necessarily the same capacity. The steps are as follows:

1. Start with a random order of agents.
2. Initialize the allocation variables to be empty sets, just to be safe.
3. For each category hhh, loop with the specific order of the agents.
4. Apply Round Robin (each agent in order picks the most beneficial item for them).
5. Draw an envy graph, and as long as there are no cycles, continue.
6. Before exiting the loop, reverse the order to a topological sort, ensuring that the agent envied by no one gets to choose first the next time, giving them a better chance of getting a more beneficial item.

Repeat the same process until you finish looping through all the categories.

This algorithm guarantees EF-1 allocation under the conditions of identical partition matroid constraints, not necessarily with identical valuations for the items. Here, items can be swapped to eliminate envy cycles because the partition constraints are identical.

Algorithm 2: Capped Round Robin What distinguishes this algorithm is that it guarantees finding F-EF1, assuming it is a uniform matroid (one category), and it allows different valuations and capacities for each agent.

Explanation of the Steps:

Explanation of the Steps:

- Input: Category C_h and set K_{ih} (maximum capacity for each agent) and some order σ.
- Initialize: $L \leftarrow C_h$, $P \leftarrow \{i : k_{hi} = 0\}$, $t \leftarrow 0$, for all $i \in [n]$, $X_{hi} \leftarrow \emptyset$.
- While L is not empty:
  - $i = \sigma[t]$
  - If $i \notin P$:
    - $g = \arg\max_{g \in L}(v_i(g))$ (meaning the item with the maximum valuation in the eyes of agent i)
    - $X_{hi} = X_{hi} \cup \{g\}$
    - $L = L \setminus \{g\}$
    - If $K_{hi} = |X_{hi}|$, add agent i to P (list of those who finished).
    - End-if
  - End-if
  - $t = t + 1 \mod n$ (cyclically).
- End-while
- Return X_h.

Explanation in a Few Words: As mentioned earlier, this algorithm is limited to one category and allows different valuations and capacities for each agent. Starting with a certain order, as long as there are items to divide, the first agent in that order picks the item with the highest value in their eyes and adds it, assuming they have not yet reached their maximum capacity. At each addition, check if they have reached the maximum, and block them from choosing again. Repeat this process for each agent, running cyclically, meaning when you finish with the last agent and still have items to divide, you return to the first one on the list.

Algorithm 3: CRR with Two Categories This time, we extend the previous algorithm to work with two categories, meaning we expanded from a uniform matroid to a partition matroid with two categories similar for everyone, with possibly different capacities. This is handled in the implementation of the original CRR.

Pseudocode:

- Let σ be some order of the agents.
- Apply Algorithm 2 (CRR) on it.
- Reverse the order.
- Apply Algorithm 2 (CRR) on the reversed order.
- The union of the results of steps 2 and 4 gives us X, an F-EF1 allocation.

The proof of Algorithm 3 is very similar to that of 2, just with two categories this time.

Algorithm 4: Different Capacities, Identical Valuation, Same Category This is somewhat similar in structure to RR, where each iteration uses a sub-process. This time it involves CRR in each iteration with a topological order inversion to Algorithm 1. Here, a lemma proves that we won't need to remove cycles from the envy graph because they won't exist under the given constraints (which is good for us because in the case of different capacities, we risk violating the capacity constraint when swapping bundles between agents).

Pseudocode:

- Initialize the allocations X_i to empty sets σ random order.
- For each C_h:
  - Apply CRR(C_h, σ).
  - For each agent i, combine their allocation with X_i.
  - Change σ to be the topological order of the feasible envy graph.
- End-for

Lemma: For every definition under identical valuations (which may have different capacities), the envy graph of any feasible allocation will always be acyclic.

Algorithm 5: Different Capacities, Binary Valuation, Same Category

Initialization for each i, X_i to an empty set.

- For each category:
  - For each i, initialize X_{hi} to an empty set.
  - $T_h = \max(K_{ih} \text{ for every } i)$, meaning the maximum capacity.
  - For each t from 1 to T_h:
    - Build a graph of agent-resource G_{ht}.
    - Build an envy graph given X.
    - Change σ to be the topological order of the feasible envy graph.
    - Find a maximum matching (priority matching) according to σ and G_{ht}.
    - For each agent assigned in G_{ht}, add the items assigned to them to their X_{hi}.
  - End-for
  - If there are still items in h that were not allocated, randomly assign them to agents who still have capacity.
- End-for

Definitions Related to the Algorithm:

Item-Agent Graph: This is a bipartite graph in any sub-allocation X given a category h, containing on the left side the agents who have not reached their maximum capacity and on the right side the items in category h that have not yet been allocated to anyone. An edge from an agent to an item means the agent wants that item ($v_i(\text{item}) = 1$).

Maximum (Preferred) Matching: Starting with a matching given $G = (V, E)$, a matching is in simple terms a subset of the edges E such that each vertex appears at most once at the ends of the edges (in our case, when talking about an item-agent graph, an agent in any matching cannot take 2 items, and moreover, one item cannot appear to 2 agents in any matching, which is good and logical in our case). A maximum matching is simply the matching with the most units in the vector, defining it to be the vector of agents in our case $[n]$ (in two words, the best matching is the one where we allocated to the most agents in each iteration). The order in the vector is important.

Brief Explanation of Algorithm 5: The algorithm loops over the categories, where in each category, a graph of agent-item is drawn, and from this graph, the maximum (best) matching is found, involving the topological order of the envy graph (guaranteed by a lemma that there won't be cycles to deal with). Then, according to the matching, allocate the items to each agent. Repeat the same process at most until $T_h = \max(k_{hi})$, the maximum capacity among the agents. Finally, move to the step of randomly assigning the leftovers to agents. If you think a bit, you can conclude that after T_h iterations, in the worst case, some may not have reached their maximum capacity because they didn't get the items they wanted according to the maximum matching, meaning the leftover items are certainly valued at 0 for them; otherwise, they would have gotten them in the first stage.

Algorithm 1(Per-category round robin)

Reminder of pseudo-code:

ALGORITHM 1: Per-Category Round Robin, (Biswas & Barman, 2018)

initialize:
$\sigma \leftarrow$ an arbitrary order over the agents.
$\forall i \in [n]\ X_i \leftarrow \emptyset$
for *every category* h **do**
  Run round robin with C^h, σ;
  Let X_i^h be the resulting allocation for agent i;
  $\forall i \in [n]\ X_i \leftarrow X_i \cup X_i^h$;
  Draw envy graph for current allocation;
  Remove cycles from the graph, switching bundles along the cycles;
  Set σ to be a topological order of the graph;
end

Example: (includes cycle of size 3)

$\sigma = \{1,2,3\}$ (agents order)

For each i in [n] $X_i = \emptyset$ (empty initial allocations)

C=$\{C^1 = \{m1\}, C^2 = \{m2\}, C^3 = \{m3\}\}$ (categories as in partition matroid)

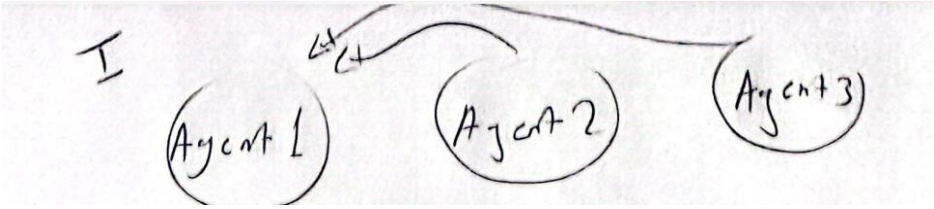
For each i in [n] $K_i^1 = 3\ , K_i^2 = 3$ (same capacities , for cycle removal to work)

Valuations={'Agent1':{'m1':1,'m2':2,'m3':3},'Agent2':{'m1':1,'m2':1,'m3':3},'Agent3':{'m1':10,'m2':1,'m3':5}}

Iteration I :

1. Starting with category 1
2. Running round robin with category 1 and σ
3. agent 1 goes first gets the most valuable item in his perspective (m1)
4. no more items in Category1

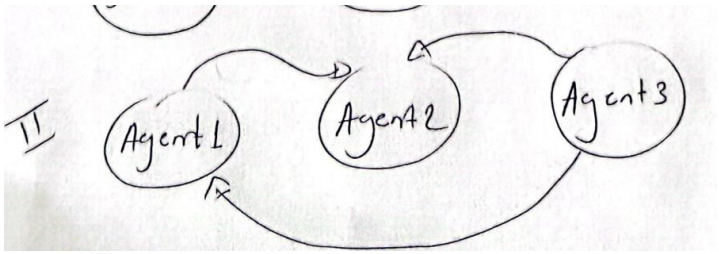
5. Drawing envy graph : agent 1 has an item everyone else envies him , yet no envy cycles found



6. Topological sort the σ : since 1 is envied by 2 and 3 , he wont get to be first in the topological sort so its $\sigma = \{3,2,1\}\ OR\ \sigma = \{2,3,1\}$
7. For the sake of our pre-determined scenario we assume topological sort algorithm picks $\sigma = \{2,3,1\}$ (to make sure we get the cycle of size 3)

Iteration II:

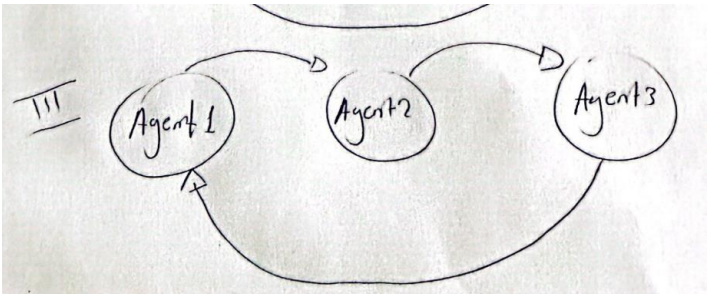
1. Running on category 2
2. RR(category2, $\sigma = \{2,3,1\}$)
3. Since we only have m2 in category 2 the first one in order gets it
4. According to order agent 2 starts , gets m2 ,
5. Draw envy graph : now agent2 has m2 , agent 1 values m2 more than his item so he envies agent2 , agent 3 now envies both 1 & 2 yet no cycles to be found



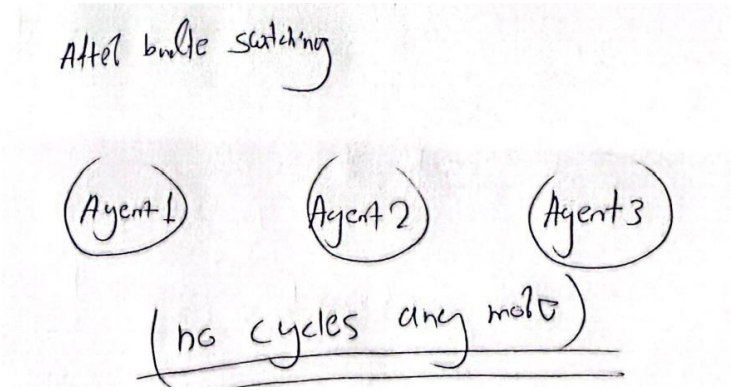
6. topological sort : this time is unique since we only take the node with indegree 0 (with no edges pointing to it) in this case we only got agent 3 so , , $\sigma = \{3,1,2\}$

Iteration III:

1. Running on category 3
2. RR(category2, $\sigma = \{3,1,2\}$)
3. Agent 3 starts , we only have m3 in this category , he gets m3
4. Draw envy graph : now we have agent 1 -> agent2 -> agent3->agent1 cycle of size 3 , needs to be eliminated !



5. we run helper_remove_cycles() on the graph , switching bundles along the cycle , here is how the graph looks like after !



6. topological sort : after invoking cycle-elimination function we're left with a graph with no edges at all ! , hence assuming there would be another category say category 4 , σ can be any order !

Final allocation

$X = \{agent1 = \{m2\}, agent2 = \{m3\}, agent3 = \{m1\}\}$

We did good in here 😊 this example also shows that in best case scenario we can have a F-EF allocation which is even better than F-EF1

Algorithm 2(capped-round robin)

4.1 Uniform Matroids

As a warm-up, we present a simple algorithm for a setting with a single category. We call it Capped Round Robin (CRR). CRR is a slight modification of round robin, where if an agent reached her capacity — she is skipped over (Algorithm 2).

ALGORITHM 2: Capped Round Robin

Input: Category C^h with capacities k_i^h for every $i \in [n]$, and an order σ over $[n]$.

Initialize: $L \leftarrow C^h$, $P \leftarrow \{i : k_i^h = 0\}$, $t \leftarrow 0$, $\forall i \in [n]\ X_i^h \leftarrow \emptyset$.
while $L \neq \emptyset$ **do**
  $i \leftarrow \sigma[t]$.
  if $i \notin P$ **then**
    $g = \text{argmax}_{g \in L}\, v_i(\{g\})$.
    $X_i^h \leftarrow X_i^h \cup \{g\}$. `// Agent` i `gets her best unallocated item in` C^h
    $L \leftarrow L \setminus \{g\}$.
    if $|X_i^h| == k_i^h$ **then**
      $P \leftarrow P \cup \{i\}$ `// Agent` i `cannot get any more items from` C^h
    end if
  end if
  $t \leftarrow t + 1 \mod n$.
end while
return X^h

CRR finds a F-EF1 allocation whenever the constraints of all agents are *uniform matroids*, i.e., all items belong to a single category (but agents may have different capacities and different valuations).

Example:

$\sigma = \{1,2,3,4\}$ (agents order)

For each i in [n] $X_i = \emptyset$ (empty initial allocations)

L=$C^1 = \{m1, m2, m3, m4, m5, m6, m7\}$ (categories as in partition matroid)

$t = 0$

$K_1^1 = 0$, $K_2^1 = 1$,$K_3^1 = 2$, $K_4^1 = 3$

$P = \{1\}$ (agent 1 is already in the blocked list since he has capacity 0)

Valuations={'Agent1':{'m1':1,'m2':2,'m3':3, 'm4':4, 'm5':5, 'm6':6, 'm7':0}
,'Agent2':{'m1':6,'m2':5,'m3':4, 'm4':3, 'm5':2, 'm6':1, 'm7':0}
'Agent3':{'m1':1,'m2':2,'m3':5, 'm4':6, 'm5':3, 'm6':4, 'm7':0}
'Agent4':{'m1':5,'m2':4,'m3':1, 'm4':2, 'm5':3, 'm6':6, 'm7':0}}

Iteration I :

1. starting with agent 1 , agent 1 is already in P , so he's skipped
2. t+=1mod n

Iteration II :

1. agent 2 turn, agent 2 has capacity of 1
2. agent 2 gets m1 since its the most valuable item in his eyes
3. agent 2 is done we add him to P (blocked-list since capacity is 0)
4. t+=1mod n

Iteration III :

1. agent 3 turn, agent 2 has capacity of 2
2. agent 3 gets m4 since its the most valuable item in his eyes
3. t+=1mod n

Iteration IV :

1. agent 4 turn, agent 4 has capacity of 3
2. agent 4 gets m6 since its the most valuable item in his eyes
3. t+=1mod n

Iteration V :

1. back to agent 1, agent 1 is blocked skipping
2. t+=1mod n

Iteration VI :

1. back to agent 2, agent 2 is blocked skipping
2. t+=1mod n

Iteration 7 :

1. back to agent 3, agent 3 has remaining capacity=1
2. gets the most valuable item in his eyes -> m3
3. is added to blocked list
4. t+=1mod n

Iteration 8 :

1. back to agent 4, agent 4 has remaining capacity=2
2. gets the most valuable item in his eyes -> m2
3. t+=1mod n

etc .. we go again over the agents skipping all till agent 4 again we give him m5 and add him to P (blocked list !) ,next iteration no agent is eligible to get items anymore ! , hence we remain with item m7 unallocated ! , but still our final allocation is certainly F-EF1

$$X = \{agent1 = \emptyset, agent2 = \{m1\}, agent3 = \{m3, m4\}, agent4 = \{m2, m5, m6\}\}$$

Algorithm 3(two categories capped-round robin)

4.2 Two categories

While CRR may not find an F-EF1 allocation for more than one category, we can extend it to two categories by running CRR with reverse order on the second category; see Algorithm 3.

Theorem 3. *When all agents have partition-matroid constraints with at most two categories, the same categories but possibly different capacities, an F-EF1 allocation always exists and can be found efficiently.*

ALGORITHM 3: Back-and-Forth CRR

1: $\sigma \leftarrow$ an arbitrary order over the agents.
2: Run Capped Round Robin with C^1, σ. Let X_i^1 be the outcome for each agent $i \in N$.
3: $\sigma' \leftarrow$ reverse(σ).
4: Run Capped Round Robin with C^2, σ'. Let X_i^2 be the outcome for each agent $i \in N$.
5: **return** $X_i 1 \cup X_i^2$ for all $i \in N$.

Example:

$\sigma = \{1,2,3\}$ (agents order)

For each i in [n] $X_i = \emptyset$ (empty initial allocations)

$C^1 = \{m1, m2, m3, m4\}$
$C^2 = \{m5, m6\}$

$K_1 = \{K_1^1 = 3, K_1^2 = 0\}$, $K_2 = \{K_2^1 = 0, K_2^2 = 2\}$,$K_3 = \{K_3^1 = 0, K_3^2 = 5\}$

Valuations={'Agent1':{'m1':1,'m2':2,'m3':3, 'm4':4, 'm5':5, 'm6':6}
,'Agent2':{'m1':6,'m2':5,'m3':4, 'm4':3, 'm5':2, 'm6':1}
'Agent3':{'m1':5,'m2':3,'m3':1, 'm4':2, 'm5':4, 'm6':6}

Since this algorithm relies heavily on the previous one , we're not going to show each step when calling the previous algorithm inside it

1. $\sigma = \{1,2,3\}$
2. $CRR(\sigma, C^1)$
3. $X^1 = \{X_1^1 = \{m2, m3, m4\}, X_2^1 = \emptyset, X_3^1 = \emptyset\}$ (since agent 2 and agent 3 have 0 capacity for category 1)

4. $\sigma = reverse(\sigma)$
5. $CRR(\sigma, C^2)$
6. $X^2 = \{X_1^2 = \emptyset, X_2^2 = \{m5\}, X_3^2 = \{m6\}\}$

Final allocation :

$$X = \{agent1 = \{m2, m3, m4\}, agent2 = \{m5\}, agent3 = \{m6\}\}$$

Algorithm 4(per-category capped-round robin)

ALGORITHM 4: Per-Category Capped Round Robin

Input: M, C, k_i^h for every $i \in [n], h \in [l]$
Output: an allocation X which is $F-EF1$

1: Initialize: $\sigma \leftarrow$ an arbitrary order over the agents; $\forall i \in [n]\ X_i \leftarrow \emptyset$.
2: **for all** $C^h \in C$ **do**
3: Run Capped Round Robin with C^h, σ.
4: $\forall i \in [n]\ X_i \leftarrow X_i \cup X_i^h$.
5: Set σ to be a topological order of the feasible-envy graph (which is acyclic by Lemma 1).
6: **end for**

Lemma 1. *For any setting with identical valuations (possibly with different capacities), the feasible envy graph of any feasible allocation is acyclic.*

Note : this algorithm relies heavily on capped round robin (algorithm1) , main difference is , agents may have different capacities unlike algorithm 1 & must have identical valuations , in order to achieve F-EF1 with acyclic envy-graph, hence its safe to do topological sort without hesitation and checking for envy-cycles 😊

Example: (3 agents 3 categories, different capacities)

Example:

$\sigma = \{2,3,1\}$ (agents order)

For each i in [n] $X_i = \emptyset$ (empty initial allocations)

$C^1 = \{m1, m2, m3, m4\}$
$C^2 = \{m5, m6, m7\}$
$C^3 = \{m8, m9\}$

C=$\{C^1, C^2, C^3\}$

$K_1 = \{K_1^1 = 1, K_1^2 = 4, K_1^3 = 4\}$, $K_2 = \{K_2^1 = 4, K_2^2 = 0, K_2^3 = 4\}$
,$K_3 = \{K_3^1 = 4, K_3^2 = 4, K_3^3 = 0\}$

Valuations={for each agent i:{'m1':11,'m2':10,'m3':9,'m4':1,'m5':10,'m6':9.5,'m7':9,'m8':2,'m9':3}} (identical valuations)

Iteration I :

1. Running CRR(σ, C^1) , starting with agent 2 with respect to order
2. Agent 2 has capacity , so he gets m1
3. Going for agent 3 agent 3 gets m2
4. Agent 1's turn gets m3 (note : gets blocked and added to P since he has no capacity after acquiring m3)
5. Agent 2 turn as we do the modulo length in the loop of RR , gets m4
6. Drawing envy graph : we get a graph in which agent 1 & agent 3 envy agent 2 , and agent 1 also envies agent 3 since he picked first , makes sense ! , no cycles as its proven
7. Heading to topological sort of the graph to get the new σ= {1,3,2}

$$X^1 = \{agent1 = \{m3\}, agent2 = \{m1, m4\}, agent3 = \{m2\}\}$$

Iteration II :

1. Running CRR(σ, C^2) , starting with agent 1 with respect to order
2. Agent 1 gets m5
3. Agent 3 turn gets m6
4. Agent 2 is blocked he has capacity 0 for category 2
5. Going back to agent 1 gets m7
6. Done no more items in the category
7. Updating the envy graph # TODO insert envy-graph
8. Topological sort is σ= {2,3,1}

$$X^2 = \{agent1 = \{m5, m7\}, agent2 = \emptyset, agent3 = \{m6\}\}$$

Iteration III :

1. Running CRR(σ, C^3) , starting with agent 2 with respect to order
2. Agent 2 gets m9
3. Agent 3 is blocked , capacity=0
4. Agent 1 gets m8
5. Done no more items in the category
6. No need for envy graph and sort since there is not 4'th category

$X^3 = \{agent1 = \{m8\}, agent2 = \{m9\}, agent3 = \emptyset\}$

We're done no categories left , final allocation is combined

$X = \{agent1 = \{m3, m5, m7, m8\}, agent2 = \{m1, m4, m9\}, agent3 = \{m2, m6\}\}$

Algorithm 5(Iterated-priority matching)

ALGORITHM 5: Iterated Priority Matching

Initialize: $\forall i \in [n]\ X_i \leftarrow \emptyset$.
for each category h **do**
  $\forall i \in [n]\ X_i^h \leftarrow \emptyset$.
  $T^h := \max_{i \in N} k_i^h$.
  for $t = 1, \ldots, T^h$ **do**
    Construct the agent-item graph G_t^h (see Definition 13).
    Construct the feasible-envy graph corresponding to $\mathbf{X}$.
    $\sigma \leftarrow$ a topological order on the feasible envy-graph.
    Find a priority matching in G_t^h according to σ (see Definition 14).
    For every agent i who is matched to an item g_i: $X_i^h \leftarrow X_i^h \cup \{g\}$.
  end for
  Allocate the unmatched items of C^h arbitrarily to agents with remaining capacity.
  $\forall i \in [n]\ X_i \leftarrow X_i \cup X_i^h$.
end for

Example: (3 agents , 3 categories , with common interests, and remainder unallocated items at the end)

For each i in [n] $X_i = \emptyset$ (empty initial allocations)

$C^1 = \{m1, m2, m3\}$
$C^2 = \{m4, m5\}$
$C^3 = \{m6\}$

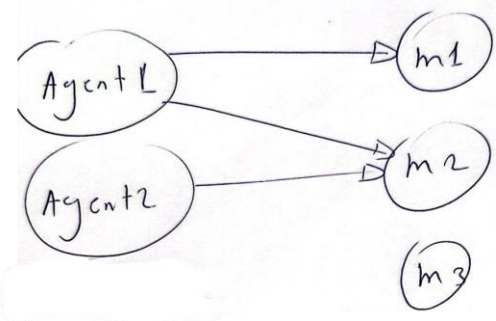
C=$\{C^1, C^2, C^3\}$

$K_1 = \{K_1^1 = 1, K_1^2 = 1, K_1^3 = 1\}$, $K_2 = \{K_2^1 = 1, K_2^2 = 1, K_2^3 = 1\}$
,$K_3 = \{K_3^1 = 0, K_3^2 = 0, K_3^3 = 1\}$

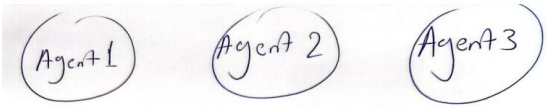
Valuations=
{'Agent1':{'m1':1,'m2':1,'m3':0, 'm4':1, 'm5':1, 'm6':1}
,'Agent2':{'m1':0,'m2':1,'m3':0, 'm4':1, 'm5':1, 'm6':1}
'Agent3':{'m1':0,'m2':0,'m3':0, 'm4':0, 'm5':0, 'm6':1}

Iteration I :

1. Starting with category 1 , order ={1,2,3}
2. T^1 = maximum capacity out of all agents in category 1 = 1
3. Running from 1 to .... T^1
4. Building agent-item graph (we pay attention to common interest in m2)
   definition : agent item graph is a bipartite graph in which on one side we have agents on the other we have items , and every agent which wants item i (values it binarily=1) we add an edge from that agent to that item



5. No need for envy graph now since no allocated items yet

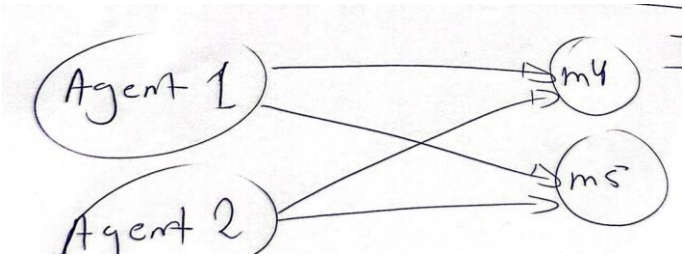


6. Topological sort would be any kind of order we assume its {1,2,3}
7. Finding priority matching with respect to the order
   definition : priority matching as its name says , does matching with respect to priority=order , maximizing the vector of 1's with respect to order
8. In our case priority matching is (agent1,m1) (agent2,m2)
9. According to the matching we allocate the items to the matched agents
10. We're done since T=1
11. Unallocated item m3 cant be allocated since no one has more capacity thus its left behind

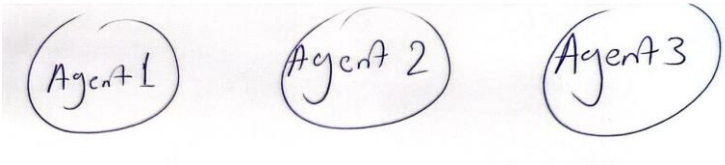
$$X^1 = \{agent1 = \{m1\}, agent2 = \{m2\}, agent3 = \emptyset\}$$

Iteration II :

1. Category 2
2. T^2 = maximum capacity out of all agents in category 1 = 1
3. Running from 1 to .... T^2
4. Building agent-item graph (common interest in both m4,m5)



5. Build envy-graph : again , no feasible envy, since 3 has no capacity

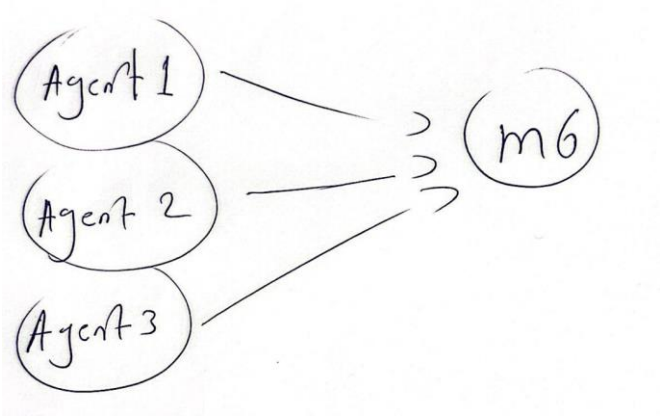


6. Topological sort : no envy we maintain same order = {1,2,3}
7. Finding priority matching with respect to the order
   since agent 3 has no capacity he isn't taken in calculation
8. In our case priority matching is (agent1,m5) (agent2,m4) , vector is (1,1)
9. According to the matching we allocate the items to the matched agents
10. We're done since T=1

$X^2 = \{agent1 = \{m5\}, agent2 = \{m4\}, agent3 = \emptyset\}$

Iteration III :

1. Category 3
2. T^3 = maximum capacity out of all agents in category 1 = 1
3. Running from 1 to .... T^3
4. Building agent-item graph (all agents want m6)



5. Build envy-graph : again , no feasible envy

6. Topological sort : no envy thus keep same order = {1,2,3}
7. Finding priority matching with respect to the order
8. In our case priority matching is (agent1,m6) , vector is (1,0,0)
9. According to the matching we allocate the items to the matched agents
10. We're done since T=1

$X^2 = \{agent1 = \{m6\}, agent2 = \emptyset, agent3 = \emptyset\}$

Done , final allocation :

$X = \{agent1 = \{m1, m5, m6\}, agent2 = \{m2, m4\}, agent3 = \emptyset\}$

Note : item 3 remains unallocated , but that's fine , what matters is final allocation is F-EF1

Note : run the example on website for further clarification if needed .

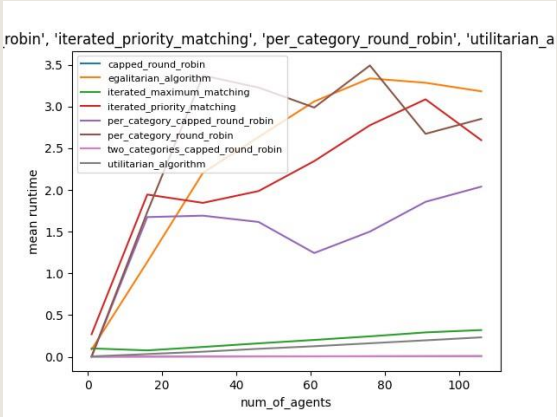
Chapter 2: Experiments

Runtime experiments

RUNTIME

We can easily identify the the slowest algorithm (due to nested looping & advanced graph algorithms like matching algorithm) , its algorithm5 iterated priority matching , (optimized using cython)

Two categories round robin (pink) and capped round robin (we believe its hiding behind the pink line) are the fastest due to the fact they only work on a limited number of categories compared to the other algorithms which works on up to 120 categories

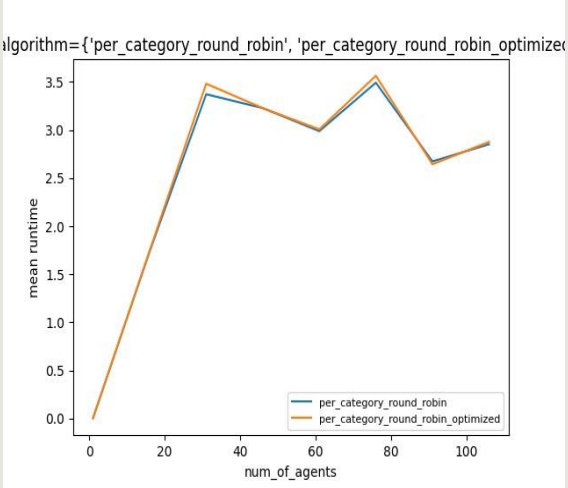


14

RUNTIME OPTIMIZING ALGORITHM 1

Though we did our best to optimize algorithm 1 , we didn't reach impressive results , but we can pay attention that sometimes the optimized is a little bit beneath the non-optimized algorithm , saving maybe couple of micro -seconds

This could be due to the fact that our overall algorithms run fairly fast

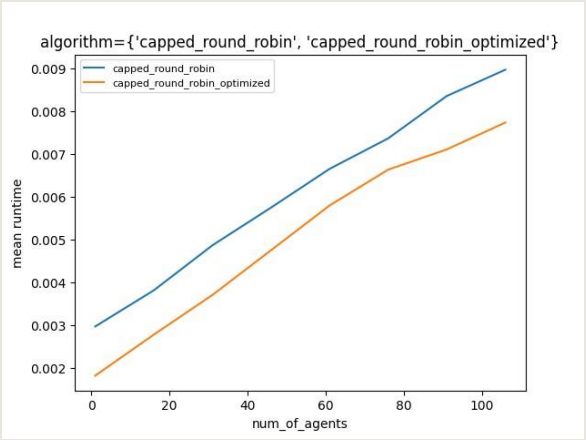


15

RUNTIME OPTIMIZING ALGORITHM 2

Now here we have some impressive results (but pay attention its in micro -seconds not seconds) , at least we can clearly see optimized runs faster

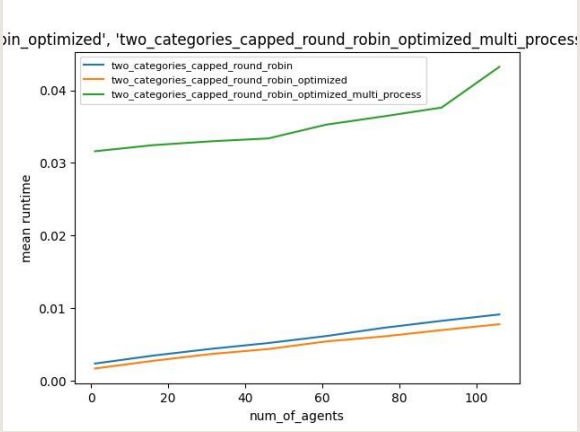
Algorithm has been cythonized ,part of it (helper_categorization_friendly_picking_sequence) has been additionally optimized with cache



16

RUNTIME
OPTIMIZING ALGORITHM 3

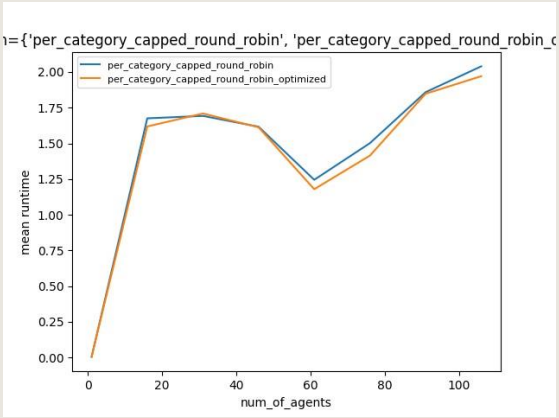
Again we have fair results (but pay attention its in micro-seconds not seconds) , at least we can clearly see optimized runs faster , excluding the multi-process , we chose to keep it just to show that multi -processing method in here didn't get us the desired results



17

RUNTIME
OPTIMIZING ALGORITHM 4

Again we have fair results, at least we can pay attention optimized runs faster .

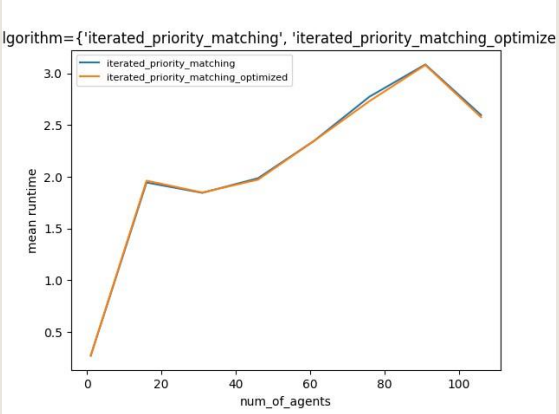


18

RUNTIME
OPTIMIZING ALGORITHM 5

Again we have fair results, at least we can pay attention optimized runs faster .

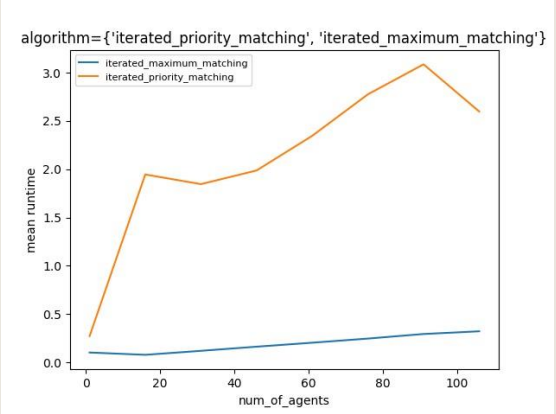
Optimized has been cythonized , it also sometimes relies on picking_sequence which has been cythonized + cached



19

RUNTIME
ITERATED PRIORITY MATCHING VS ITERATED MAX MATCHING

We can clearly see our algorithm iterated priority matching (algorithm 5) runs fairly slower than iterated max matching (iterated max matching has been implemented by another dev)



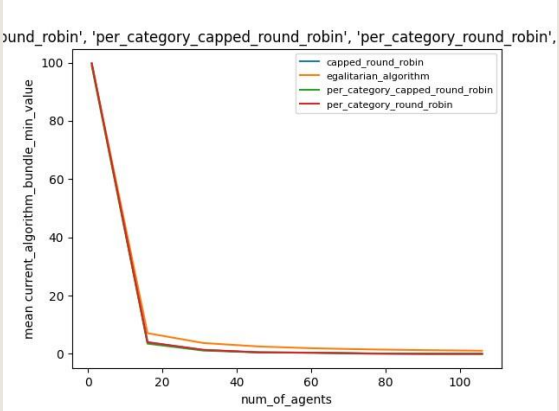
20

Egalitarian value experiments

EGALITARIAN VALUE SINGLE CATEGORY

Egalitarian algorithm , as its name states , it's the best in its own task which is to maximize the minimum value of bundle (cares for the poor)

We can clearly see it tops the other algorithms meaning the graph makes sense !

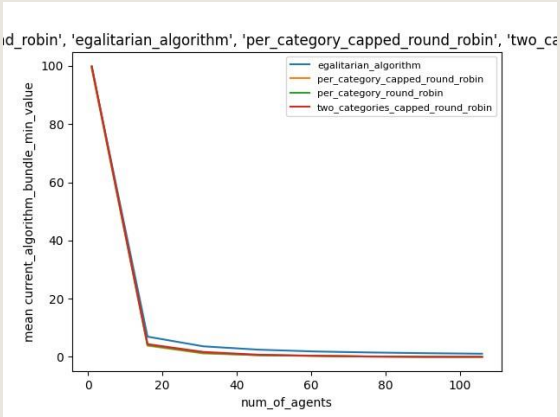


21

EGALITARIAN VALUE MANY CATEGORIES

Egalitarian algorithm , as its name states , it's the best in its own task which is to maximize the minimum value of bundle (cares for the poor)

We can clearly see it tops the other algorithms meaning the graph makes sense !

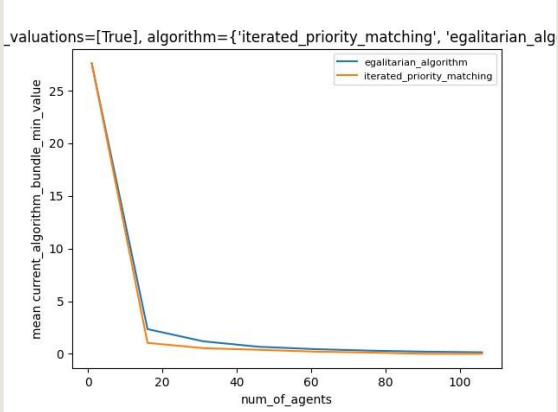


22

EGALITARIAN VALUE
BINARY VALUATIONS

Egalitarian algorithm , as its name states , it's the best in its own task which is to maximize the minimum value of bundle (cares for the poor)

We can clearly see it tops the other algorithms meaning the graph makes sense !

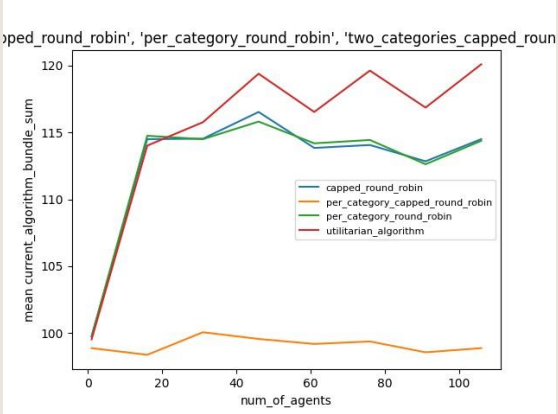


23

Utilitarian value experiments

UTILITARIAN VALUE
SINGLE CATEGORY

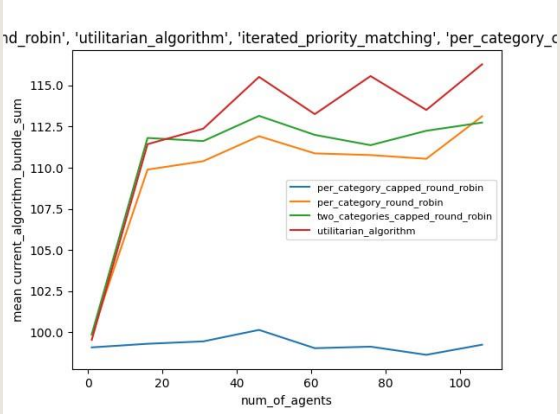
Utilitarian algorithm is known for being the best in its task , maximizing the bundle values (cares for the rich/whoever values more)



24

UTILITARIAN VALUE MULTIPLE CATEGORIES

Utilitarian algorithm is known for being the best in its task , maximizing the bundle values (cares for the rich/whoever values more)

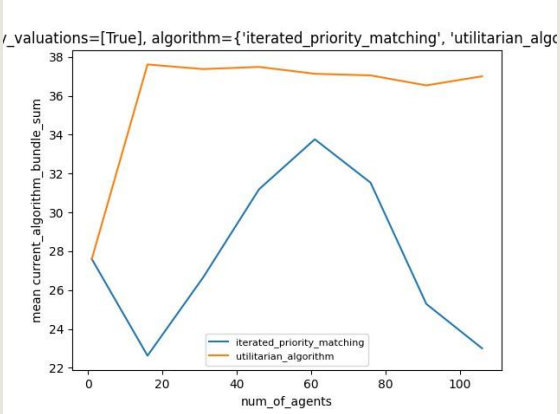


25

UTILITARIAN VALUE BINARY VALUATIONS

Utilitarian algorithm is known for being the best in its task , maximizing the bundle values (cares for the rich/whoever values more)

We can clearly see out of all algorithms iterated priority matching is the worst in trying to maximize the sum of bundles



26

Chapter 3: Final Thoughts

This research topic inspired us to delve more into it , we implemented its code contributed to Fairpyx library , built a website showcasing it in an elegant way ! , the fact such algorithms solve daily life scenarios is itself a source of motivation for further research 😊