Fair allocation: implementing and evaluating an algorithm for competitive allocation of chores

Haijing Ge

4th Year Project Report
Artificial Intelligence and Mathematics
School of Informatics
University of Edinburgh

2022

Abstract

We implement and evaluate an algorithm that computes solutions for a competitive allocation of chores, intending to cover every distinct disutility profile. Competitive allocations are appealing due to many reasons, including the celebrated envy-freeness and Pareto-efficiency. However, computing competitive allocation for chores is difficult as the solutions can be wildly multi-valued and discontinuous, corresponding to all critical points of Nash Social Welfare on the Pareto frontier.

In this thesis, we adopt an agent-based approach and show that the algorithm proposed by [10] works in practice. We also explore our implementation's feasibility, surrounding four different themes, namely integrity and privacy, time complexity, incentive compatibility, and the problem of having no solution or multiple solutions. We find that our algorithm becomes very hard to compute after both the numbers of agents and chores reach five, but the time complexity also depends on the value function. This influence was not considered by the original paper at all, hence it is a discovery of our own and we are able to give a detailed explanation for the result. We also find that integrity and privacy can be maintained during the process, and the incentives for lying can be partially restrained. We hope for more follow-up work on all these topics, and especially on dealing with the no-solution and multi-solution problem, validating the optimism appeared in our experiments.

Research Ethics Approval

Declaration

Acknowledgements

I want to deliver thanks to my supervisor Prof.Kousha Etessami, without whose guidance I could not have explored this amazing field of algorithmic game theory; and to my families, without whose support I would not be able to pursue what fascinated me.

Table of Contents

Chapter 1

Introduction: from competitive equilibrium to fair allocation

"The lion bade the fox to make a division. Gathering the whole into one great heap, the fox reserved but the smallest mite for himself. 'Ah, friend,' said the lion, 'who taught you to make so equitable a division?' 'I wanted no other lesson,' replied the fox, 'than the ass' fate.' " (*Aesop's fables*)

1.1 Motivation to implement a chore division algorithm

An equilibrium can be roughly described as a stable state where some balance of variables is achieved and will remain still, unless an external force takes its presence. Considering that when an allocation is fair, no agents will have incentives to dispute and change their behaviour, we gain an intuition that fairness and equilibrium are inherently linked concepts as depicted in [21].

Imagine a number of agents debating about how to distribute a set of *divisible* goods. If every agent has already earned a certain income, it will be natural for them to wonder if a price for each good can be found, together with a partition of goods, so that everyone can afford their favourite bundle from that partition. If the answer is positive, we claim that a *competitive equilibrium* exists and the prices are found as *market-cleaning prices*.

Those agents have good reasons to pursue such an allocation: a) no one will envy others due to that they have bought their most preferred bundle (*Envy-freeness*); b) it has been proven that no another allocation exists that can bring one agent strictly more happiness and others at least the same amount of happiness (*Pareto-efficiency or Pareto-Optimality*); c) since each agent's income/budget can be settled through mutual agreement beforehand and prices are shared by the public, this allocation can be argued fair in its own right. It is also possible to embody a notion of egalitarianism further by prescribing a equal budget to every agent, in which case the allocation rule is named *Competitive Equilibrium with Equal Income* (CEEI), or the *competitive rule*, proposed by Varian in 1974 [48], a pioneer in economic design using general equilibrium theory.

As early as in 1874, the economist Léon Walras [49] has set off to solve problems around the balance between supply and demand using a clearing price. Hence a *competitive equilibrium* is known as *Walrasian* too. And the model we set up in the second paragraph is often referred to as the *Fisher market* in literature, attributing to the economist Irving Fisher. In 1954, its generalised model *Arrow-Debreu* created by Kenneth Arrow and Gérard Debreu [1] marked a further step in the field with additional mathematical maturity. Around the same time, political thinkers' discussions about fairness and equality inspired economists to incorporate various ideas into their models, which we will discuss more about in the next chapter.

By Moulin's description [38], the last few decades have witnessed a "renaissance" in studying fair allocation's rules. We can give its credit to the birth of Internet, as the prevalent interactions between users online when they are divvying computing resources and data have demanded computer scientists to pay attention to the methodology of fair allocation as well as its normative properties. An example of the former is evaluating different division rules' computational complexities and an example of the latter is the systematic quest for numerical evaluations of the trade-offs between normative requirements. Thus the field expanded quickly in the last few decades and shifted much attention to more specific structures.

However, the problem of chore allocation was largely ignored. Only in 1978, three decades after Hugo Steinhaus became the first to formalise the divisible good allocation or the cake-cutting problem [45], Gardner [26] introduced the frame of chore division. Nevertheless, in real life, not only resources, commodities and wealth but also duties, chores, and costs have to be allocated. The situation emerges from a myriad of backgrounds ranging from housework division, divorce dissolution to greenhouse gas emissions reduction [47], border settlement.

Motivated by the pervasive needs to allocate bads in reality, the desirable qualities of competitive equilibria, and the great advancement in fair allocation made by algorithmic game theorists but having not focused too much on chores, we carried out our project to implement and evaluate the algorithm proposed by Branzei et al.[10] that computes all competitive allocation of chores (all in the sense of covering all disutility profiles). We believe in our work's importance as the algorithm has a special status of being the first and only known solution to the hard question of computing competitive chore allocations. Amazingly, it achieves a polynomial running time when either the number of items or agents is fixed.

We had three major objectives in mind:

1. to validate [10]'s algorithm design through our independent implementation and benchmarking;
2. to analyse the algorithm's feasibility not only regarding its time complexity but also all kinds of *practicality* from a mechanism design's perspective;
3. to contribute insight to competitive allocation of chores' conceptual problems such as different normative concepts' understandings using the results of our experiments.

1.2 Summary of our contribution

By the end of this thesis, readers can see that we have met all our objectives. A summary of our contribution is given here:

- We have implemented the algorithm with our own consideration by
  - opting for an agent-based method and creating the class Agent;
  - designing the class Graph as well as its two conversions to achieve goals of different phases efficiently;
  - reducing the number of graphs to use after phase I by removing two split graphs in the two-agent sub-problem and many more in the general case;
  - computing the candidate utility profile based on Dijkstra's algorithm instead of the suggested Bellman-ford algorithm;
  - solving the maxflow problem in phase III with Dinic's algorithm instead of the suggested Edmond-Karp's algorithm.
- Regarding the algorithm's efficiency,
  - we gave a new complexity analysis of our algorithm, and gave bounds for the number of operations in each phase, which all differed from ones shown in the original paper although the overall complexity remained unchanged;
  - we deduced that improvement could mostly be made in phase I as in later phases we had already used state-of-art algorithms from others;
  - we showed that both the duality trick and graph removal were very helpful;
  - we conducted systematic experiments exploring how different value functions influence the time complexity and provided an explanation for the phenomena;
  - we found out that the problem size was the major factor to decide scalability, with m and n both reaching five an upper bound for fast solutions.
- And when investigating the algorithm's feasibility, we added our findings to the understanding of the competitive rule for chores:
  - values and budgets are treated very differently in the allocation but they have a relation when two agents' value vector differ by a scalar;
  - in order to guarantee the algorithm's integrity, publishing budgets and allocation results is essential and sufficient;
  - the secrecy of personal values can be maintained;
  - the ILB property does not help much with incentive compatibility;lying to reduce one's own disutility is hard due to uncertainties and values' privacy whereas lying to manipulate the allocation is more achievable but could be limited;

- the probability of having no solutions seem quite low and it is probable to get a solution among more than one allocations that minimises the total disutility, the worst-off agent's disutility and the Nash product.

1.3 Road map of the thesis

In Chapter 2, we take a glimpse over the vast landscape of fair allocation and discuss the closely related work on competitive allocation with goods and chores. We end by stressing the importance of building applications with previous work shown.

In Chapter 3, we give formalised definitions for our problem and introduce few theorems that motivated the original design of the algorithm. The reader will learn the main computational result as polynomial with some restraints, and how the algorithm achieves the final goal after three phases.

We discuss the implementation details in Chapter 4. The description of every phase's procedures will be given, followed by a complete complexity analysis in the end. We also include other choices from defining new classes to using Python.

In Chapter 5, we give a comprehensive evaluation about our algorithm's feasibility regarding its integrity and secrecy, time complexity and scalability, incentive compatibility and problem of having no solution or multiple solutions through a combination of mixed approaches.

We end our thesis in Chapter 6. We summarise the lesson we have learned first, then highlight our originality as well as the difficulties we faced. We finish by suggesting various work that can be done in the future.

Chapter 2

Prior work in literature and practice

Our project targets a specific kind of allocation problems, namely the allocation of divisible bads, with additive disutilities and no money transfer. To help readers grasp the scope of fair allocation, as well as to locate where our problems fit inside the vast landscape, we start by characterising the problem with its elements. Next, we present previous state-of-art research in competitive allocation of goods, some of which has motivated similar approaches for chores. Then we shows that our problem is harder than its that of allocation of goods, and relate it to previous work in allocating bads and the mixed. We will finish this chapter by highlighting the significance of algorithm implementation, through a discussion over the limited applications existing.

2.1 Defining an allocation problem with its elements

Fair allocation is about dividing a set of objects among a set of agents. The dimension of the problem expands rapidly as agents can be modeled in different ways, objects have disparate intrinsic properties, and the criteria for reconciling the conflicts of people's wishes or "fairness" diverge. Therefore, we need to characterise a problem first with its three elements: agents, objects, and criteria.

2.1.1 Agents

An agent may feel a desire, a distaste, or indifference towards an single item, which can be captured as either a positive or negative number, or zero. But a cardinal utility function is not necessary. In the cases where every agent holds the same attitude towards all items, or in other words, the items are all goods or bads (chores), ordinal utility functions can also be used. Certainly, goods and bads can be both involved, and one item can be regarded positive by some but negative by others. Research on this kind of allocation, i.e. allocating a *mixed manna* was initialised by Bogomolnaia et al. [8, 6] few years ago and has been followed up quickly ever since [2, 13].

The simplest *additive* utility function is used in this project, indicating that an agent's total utility is equal to the sum of how much they feel for each part. Although it may appear that the additive domain has already reflected the whole story that we accumulate happiness and unhappiness, the reality is always more sophisticated. Due to the limitation of space, we will only mention other kinds of utility functions in the Conclusion. Notice that we used the wording *value functions* to indicate how agents express their felt disutility about every single item. This has to be separated from *utility functions*.

Computing equilibria is in general hard [24]. This is why restricting a problem with a simpler class of utility functions is usually necessary. Competitive allocations on classic Arrow-Debreu domain has been proven losing many appealing properties that only problems on degree-1 homogeneous utilities blissfully possess.

2.1.2 Objects

What to be allocated can be categorised as either *indivisible* (like room allocation, you have to allocate a room as a whole) or *divisible* (either physically, if it can be divided infinitely like money or land, or probabilistically, or by time-sharing); a *good*, *bad* or *mixed* (if some agents desire it whereas others do not); *homogeneous* (if only the amount of such object matters to an agent) or *heterogeneous* (if sub-parts of it matter differently to an agent, like a cake with distinctive fruits on top); *disposable* (if it need not be allocated) or *non-disposable* (if it has to be allocated to someone, which is often the case for chores).

In our problem, a chore is deemed divisible, bad, homogeneous and non-disposable.

2.1.3 Fairness and other properties

An equal division of objects may be simple but inefficient. This is why contested notions of fairness are created so as to justify the unequal allocation outcomes.

One of the most direct ideas *Envy-Freeness* proposed by Foley [25] has already been mentioned in the introduction chapter. *Propotionality* is another, proposed by Steinhaus [45], indicating that each agent should get a share at least as valuable as the average of all things. We can also prioritise improving the worst's condition through equalising the value of the bundle each receives. This rule is called *egalitarian equivalent* (EE) [41], often compared with the utilitarian approach which advocates enhancing the total sum. We will come back to them at the final section of Chapter 5.

Besides the above ideas that focus on the *ex post* effect (hence sometimes known as *welfarist*), we may also consider *ex ante* "fairness". The competitive rule gives individuals an equal right to trade, entailing fairness of access to resources [22]. Another straightforward example is the lottery. When agents have the same probability to win the final prize, even if the outcome is highly unequal, they can still consider the mechanism fair.

All these notions are still not enough. Firstly, there are times the criteria cannot be satisfied (particularly with indivisible items), in which cases relaxations and approximations by some coefficients are proposed. For example, proportionality can be relaxed to *maximin share fairness* (MMS) [11]. Envy-freeness is extended to *Envy-freeness up to one good* [35], *Envy-freeness up to any good* [12] and more. Secondly, there are other properties rather than “fairness” per se that we want to include. *Pareto-Optimality* mentioned in Introduction ensures us a sense of efficiency. However, it often conflicts with fairnses [3, 46]. Solidarity between agents also sounds appealing. Moulin and Thomson [39] used *Resource Monotonicity* (RM) and *Population Monotonicity* (PM) to frame it. Finally, *incentive compatibility* is also a ubiquitous concept in literature, proposed by Hurwicz in 1972 [31] to describe a mechanism where every agent will end up the best by being truthful. If one can remain the best by being honest regardless of other’s behaviour, the mechanism is well-known as *dominant-strategy incentive-compatible* (DSIC) or *strategyproof* [50]. Another mechanism is *Bayesian-Nash incentive-compatibility* (BNIC) if everyone remains the best by staying truthful when everyone else is also truthful. We will elaborate on incentives of lying in Chapter 5. Closely related to them is a fresh concept called *Independence of Lost Bids* (ILB) created by [7, 5], analogous to *Maskin Monotonicity* in voting theories. If agent i was not allocated to an object j at all, their marginal utility v_{ij} can be perceived as a “lost bid”. ILB states that changing i’s lost bid v_{ij} will not impact the result, as long as the bid remains lost. It might be a good news as an agent should not benefit from a small misreport. [5] suggested that competitive allocation of chores satisfies ILB.

2.2 Competitive allocation of all goods

The competitive rule dominates the world of divisible goods allocation because of the celebrated Eisenberg-Gale’s convex programming algorithm [23]. Their result, together with [17, 44, 9], demonstrates that competitive allocations can be computed easily (as in polynomial time) by maximising Nash social welfare, $\mathcal{N}_b(\mathbf{u}) = \prod_{i=1}^{n} |u_i|^{|b_i|}$ a weighted product of utilities, if it is in the Fisher model with the domain being 1-homogeneous or concave and continuous utilities. Other efficient algorithms were created for the Fisher market later, adopting different techniques like primal-dual [15], network flow [40], and auction-based [28]. The competitive rule’s answer is therefore single-valued, envy-free and efficient. Moreover, maximising Nash social welfare is a good idea by itself since it balances the utilitarian and egalitarian’s ideas. Separate studies have been conducted on it [42, 12, 36], and particularly in indivisible cases.

However, computing competitive allocation in general is not guaranteed. With indivisible resources, a CEEI may not even exist [48]. For divisible goods, economists originally dwelt in the large domain of Arrow-Debreu preferences, where the relation between the Nash product of utilities and the Competitive rule is lost, and several impossibility results were found [32, 29, 33]. In 1991, the complexity class PPAD (polynomial parity arguments on directed graphs) was defined by [18], motivated largely by the classification problem for Nash equilibria. Ingenuous construction and enumeration were needed for those hard questions , and they partly inspired our algorithm’s design [16, 43].

2.3 Fair allocation with chores

2.3.1 Allocating only chores

Is a chore division problem analogous to its dual of allocating goods? For the egalitarian rule, it turns out to be true. However, for the competitive rule, we should not expect so as the two conditions expressed in inequalities have changed directions: one is that the disutilities expressed by positive numbers need to be minimised rather than be maximised; the other is that the budgets viewed as liabilities should be fulfilled and potentially exceeded rather than work as a upper limit.

As a matter of fact, now the competitive rule selects all the critical points of the Nash product among the efficient and feasible disutility profiles as opposed to only the maximum for goods [8, 5]. The solution set is therefore wildly multi-valued, disconnected and non-convex, making itself hard to compute and impossible for us to select based on some continuous parameters such as agents' disutilities. Properties such as *RM* and *PM* vanish in the global scale. And determining whether an instance of chore division with fixed earnings admits a competitive equilibrium is strongly NP-hard even under the additive domain [14]. The paper also derived a simple sufficient condition for the existence, and for them, it is shown that finding a competitive division is still PPAD-hard. Similar result has been found in the exchange model setting, that computing a CE is PPAD-hard [13]. Despite all these, we still do not want to abandon the competitive rule as it is envy-free, Pareto-efficient and retains ILB discussed in the previous section.

2.3.2 Competitive allocation of a mixed manna

The study of competitive allocations of both disposable goods and non-disposable chores was started only five years ago but good results have been achieved. The paper [8] finds that a mixed manna problem eventually becomes one of the two types we have seen before: either goods overwhelm bads then the rule behaves just as if it were allocating goods; or bads overwhelm goods, then the rule behaves just like it is a chore-allocation problem. We may regard it as good news since solving a mixed manna division problem will not be much harder than solving a chore division problem. Continued from this finding, [27] has successfully extended the chore division algorithm we implemented to this mixed case.

2.4 Existent applications for fair allocation

While it is true that more and more fruitful results have been achieved when considering agents reporting in a low-dimensional domain, just as we indicated in the introduction, a wide gap between research and practice still remains. We find much work still highly theoretical, focusing only on the relations between different properties or the hierarchy of computing hardness on the highest level. And among the few constructive solutions proposed, usually only an abstract description is given. Without any concrete implementation at hand, it is impossible for further evaluation and experimentation.

Nevertheless, we have found few online applications:

- SPLIDDIT [30] `http://www.spliddit.org`
- Adjusted Winner `https://pages.nyu.edu/adjustedwinner`
- The Fair Division Calculator1 `https://math.hmc.edu/su/fairdivision`.

Among them, SPLIDDIT is the most versatile one, providing tools for situations from rent split, indivisible goods division to egalitarian allocation of tasks. In their good division settings, the users are asked to give values summing up to a thousand. This unnatural requirement (as it is usually hard to express someone's values in such an accuracy level) also motivated us to explore how agents' expressivity impacts on the allocation. Readers shall see more discussion in Chapter five.

A couple of real-world problems have been confronted: allocating courses for students in Wharton Business school [11], allocating unused classrooms in public schools to charter schools in California [34], scheduling and communicating over a network [19], allocation of public housing units [4].

However, not all of them are successfully tackled. Let us share Budish [11]'s remark here: "something is funny in these mechanisms is that they all use exactly equal incomes, even though we know that a CEEI need not exist. Each of these mechanisms is making a variant of the following conceptual error: they treat fake money as real money that directly enters the utility function. (Currency is fake if it has no value outside the allocation problem at hand). This causes the mechanism to allocate incorrect bundles i.e., not the bundle the student actually demands at the realized prices and creates incentives to misreport [...] All other known mechanisms are either unfair ex post or manipulable even in large markets, and most are both manipulable and unfair."

This should serve as a good lesson that theory and practice is never antithetical to each other. Without mistakes made in practice, we may not be able to realise the flaws in our understanding. And with only practice, we are able to give more thorough considerations thanks to a more constrained environment and gather empirical data, such as a program's actual running time, which tends to differ from a theoretical analysis.

To our best knowledge, no application of competitive allocation of chores is available yet. Hence, we are highly motivated to carry out the work discussed in the following chapters.

Chapter 3

Understanding the algorithm that computes all competitive allocations of chores

Hopefully our readers have been convinced of the value of our work after seeing a lack of constructed solutions in the fair allocation field, and especially in allocation of chores.

We are entering the main part of the thesis now. In this chapter, we give an overview of the algorithm [10] that we implemented. For the sake of consistency and convenience, we will mostly borrow their terminologies. Therefore, unless stated otherwise, the definitions, theorems and lemmas in this chapter can be found in their paper.

We start by formalising the problem, so as to create a specific environment in which all other definitions and theories can live. After that, we unveil quickly the main complexity result. Then we take a closer look at the algorithm which can be divided into three phases. We shall describe their separate goals and procedures as well as the motivations.

3.1 Preliminaries

Suppose that we have $[n] = \{1, \cdots, n\}$, a set of agents, and $[m] = \{1, \cdots, m\}$, a set of chores to distribute.

Each agent's additive values can be specified through a vector, where $v_{i,j} > 0$ represents the disutility agent i will get if she takes up one unit of that chore j. In the original paper, all values are set to be negative, underlining the fact that chores are unwanted. Without seeing the necessity of the over-emphasis, we switch every value's sign as we find more ease with manipulating positive numbers and believe that calculation can become less mistake-prone. It is trivial for us to consider cases where some values are zero. Since it means some agent is indifferent with respect to a specific chore, we can simply allocate the entire chore to that agent. We will mostly use the word "*value*" in the thesis to represent agent's *potential* disutility generated by each chore, separating it from "*disutility*", interpreted as the *actualised* disutility about the allocation result.

In the previous chapter we have seen that the competitive rule is defined as the competitive equilibrium from equal incomes. Although this most egalitarian situation is highly likeable, it is even more useful for us to implement the algorithm in a more general setting: *competitive equilibrium with fixed income shares* (CEFI). Remark 2 of Theorem 1 in [8] has guaranteed us that our algorithm works. The potentially unequal budgets, represented by the vector $b \in R^n$, can be viewed as agents' dissimilar liabilities. For example, if an agent works full time while another agent works half of the time, they can be modeled with budgets 1 and 0.5 respectively. For the same reasons as for values, we switched the sign of budgets from the original paper.

Definition 3.1 *A chore division problem $(\boldsymbol{v}, \boldsymbol{b})$ is a pair of a matrix of values $v \in R^{n\times m}$ and budgets $b \in R^n$.*

We denote a *bundle* of chores as a vector $\mathbf{x} = (x_1, \cdots, x_m) \in R^m$, where x_j represents the amount of chore j within. Since every chore is divisible, without loss of generality, we assume that there is one unit of each chore. An *allocation* $\mathbf{z} = (\mathbf{z}_i)_{i\in[n]}$ is a set of bundles where agent i receives bundle z_i and all the chores are distributed: $\sum_{i=1}^n z_{i,j} = 1$ for each $j \in [m]$. The disutility of agent i in an allocation $\mathbf{z}$ is $u_i(\mathbf{z}_i) = \sum_{j=1}^m v_{i,j} \cdot z_{i,j}$. The entire disutility profile at an allocation $\mathbf{z}$ is $u(z) = (u_1(z_1), \cdots, u_n(z_n))$. Given a vector $\mathbf{p} = (p_1, \cdots, p_m) \in R^m$, where p_j represents the price of a chore j, the price of a bundle $\mathbf{x} = (x_1, \cdots, x_m)$ of chores is given by $\mathbf{p}(\mathbf{x}) = \sum_{j=1}^m p_j \cdot x_j$.

Definition 3.2 *(Competitive Allocation of Chores) An allocation $z = (z_1, \cdots, z_m)$ for a chore division problem $(\boldsymbol{v}, \boldsymbol{b})$ with strictly positive matrix of values and budgets is competitive if and only if there exists a vector of prices $\boldsymbol{p} = (p_1, \cdots, p_m) \in R^m$ such that: for each $i \in [n]$: Agent i's bundle minimises its disutility among all bundles $u_i(z_i) \leq u_i(x)$ while finishing its budget (fulfilled its duty): for each bundle x with $p(x) \geq b_i$.*

Beyond the above definition, economists have come up multiple ways to characterise competitive allocations. We list the most useful ones below as they motivate the design of different stages of the algorithm.

Theorem 3.1 (Proposition 46,[10]) *Fix a matrix $\boldsymbol{v}$ and a vector of budgets $\boldsymbol{b}$, consider an allocation z of chores . The following statements are equivalent:*

1. *the allocation z is competitive*
2. *(Characterisation by inequalities) $u(z)$ has strictly positive components and $z_{i,j} > 0$ implies*
$$\frac{v_{i,j} b_i}{u_i(z_i)} \geq \frac{v_{i',j} b_{i'}}{u_{i'}(z_{i'})}$$
for all $i' \in [n]$
3. *(Variational characterisation) $y = z$ maximizes the weighted utilitarian welfare $W_{\tau(\boldsymbol{u},\boldsymbol{b})}(y) = \sum_{i\in[n]} \tau_i(\boldsymbol{u},\boldsymbol{b}) \cdot u_i(y_i)$ where $\tau(\boldsymbol{u},\boldsymbol{b}) = (\frac{b_i}{u_i})_{i\in[n]}$ over all feasible allocations y.*
4. *(Analog of the Eisenberg-Gale characterisation) The utility profile $u(z)$*
   - *belongs to the set on the Pareto frontier, and*

- *is a critical point of the Nash product $\mathcal{N}_b$ on the feasible set of utilities $\mathcal{U}(v)$.*

Pareto frontier is just the set of all utility profiles of the Pareto optimal allocations, whose definition has been mentioned in the introduction. More formally speaking,

Definition 3.3 *(Pareto optimality of chore allocations). An allocation z is Pareto optimal if there is no other allocation z' in which $u_i(z_{i'}) \leq u_i(z_i)$ for every agent $i \in [n]$ and the inequality is strict for at least one agent.*

3.2 Main result: polynomial time with fixed number of agents or chores

For a given matrix of values $\mathbf{v}$ and a vector of budgets $\mathbf{b}$, we denote the set of all competitive allocations by $CA(\mathbf{v}, \mathbf{b})$, and the set of all competitive utility profiles by $CU(\mathbf{v}, \mathbf{b})$, where we have $CU(\mathbf{v}, \mathbf{b}) = \{u(\mathbf{z}) \,|\, \mathbf{z} \in CA(\mathbf{v}, \mathbf{b})\}$.

In the following theorem we claim that polynomial running time can be achieved if n or m is fixed.

Theorem 3.2 *Suppose one of the parameters, the number of agents n or the number of chores m, is fixed. Then for any tuple $(\boldsymbol{v}, \boldsymbol{b})$, where $\boldsymbol{v} \in R^{n\times m}$ is a matrix of values and $\boldsymbol{b} \in R^n$ a vector of budgets,*

- *the set $\boldsymbol{CU}(\boldsymbol{v}, \boldsymbol{b})$ of all competitive utility profiles*
- *a set of pairs $(\boldsymbol{z}, \boldsymbol{p})$ such that the allocation $\boldsymbol{z}$ is competitive with the price vector $\boldsymbol{p}$ and for any $\boldsymbol{u} \in \boldsymbol{CU}(\boldsymbol{v}, \boldsymbol{b})$ there is a pair such that $\boldsymbol{u(z)} = \boldsymbol{u}$*

can be computed using $O(m^{\frac{n(n-1)}{2}+3})$ operations if n is fixed, or $O(n^{\frac{m(m-1)}{2}+3})$, for fixed m. This shows that the algorithm runs in strongly polynomial time.

3.3 Dividing the algorithm into three phases

Definition 3.4 ([10]) *(Consumption graph) For a feasible allocation* $\mathbf{z}$*, we associate the consumption graph G_z with a non-oriented bipartite graph with parts [n] and [m], where an agent $i \in [n]$ and a chore $j \in [m]$ are connected through an edge if and only if the agent is allocated to some part of the chore: $z_{i,j} > 0$.*

These consumption graphs will play a huge part in our implementation as the inspirations for this algorithm's design are that a competitive utility profiles can be recovered if one of its corresponding consumption graph is provided, and that a utility profile can be checked for its competitiveness efficiently, plus the allocation can also be obtained as a byproduct during the checking procedure. Let us now expand these ideas into more details.

3.3.1 Encoding the Pareto frontiers

In order to retrieve every utility profile, our generated consumption graph family has to large enough. Naturally, the first candidate coming to mind will just be all bipartite graphs consisting of [n] agents and [m] chores. It serves the purpose evidently but an exponential running time is also unavoidable (there will be 2^{mn} graphs in total). The goal of this phase is to find a smaller set of graphs to save the computational cost.

Definition 3.5 *(Rich family of graphs). Consider a chore division instance **(v, b)**. A family of bipartite graphs $\mathcal{G}$ is rich if for any competitive utility profile $u \in \boldsymbol{CU}(\boldsymbol{v},\boldsymbol{b})$ there is a competitive allocation z with $u(z) = u$ such that the consumption graph G_z belongs to the family $\mathcal{G}$.*

Recall the fourth item of Theorem 3.1: being on the Pareto frontier is one necessary condition for a utility profile to be competitive. In other words, if we can encode the Pareto frontier in some way, we have a chance to succeed in finding a smaller but rich family of graphs.

Luckily enough, there is a bijection between faces of the Pareto frontier and the family of Maximal Weighted Welfare graphs, stated as Lemma 17 in [10]. Moreover, they can be computed in polynomial time with the regard to the number of agents. For those details, readers have to wait til the next chapter.

Definition 3.6 *(Maximal Weighted Welfare Graph). Let $\tau \in R^n_{>0}$ be a vector of weights. Consider the ([n], [m])-bipartite graph such that agent $i \in [n]$ and chore $j \in [m]$ are linked if $\tau_i \cdot |v_{i,j}| \leq \tau_{i'} \cdot |v_{i',j}|$ for each agent $i' \in [n]$. It is defined as a Maximal Weighted Welfare (MWW) graph and we will denote it by $G_\tau = G_\tau(\boldsymbol{v})$*

After noticing that the canonical bijection between bipartite graphs on $([n],[m])$ and $([m],[n])$ and that we have no need to distinguish them, the authors of [10] have proposed an ingenuous trick: we can choose between computing the super set $\mathcal{G}(v^T)$ and the super set $\mathcal{G}(v)$ depends on which runs faster. This is very important as now we are able to extend the algorithm to dual cases where the number of chores are small. We will come back to this "duality trick" later.

3.3.2 Calculating the utility profile for candidate graphs

Let us now recall the second item of Theorem 3.1. As it suggests that whenever an agent i is allocated the chore j, the inequality

$$\frac{v_{i,j}b_i}{u_i} \geq \frac{v_{i',j}b_{i'}}{u_{i'}}$$

holds for all $i' \in [n]$, we can derive a relation between agents (i,i') who share the same chore:

$$\frac{v_{i,j}b_i}{u_i} = \frac{v_{i',j}b_{i'}}{u_{i'}}$$

by simply applying the inequalities twice.

Since we are equipped with consumption graphs now, it is a very good news to us as the equation can be extended along a path of all connected agents. In addition, inside the connected component of i, consider a path

$$\mathcal{P} = (i_1, j_1, i_2, j_2, ..., i_L, j_L, i_{L+1} = i'),$$

where $L \geq 1$. We define the influence of the agent i' on the agent i as a product of values' ratios along the path:

$$\pi_{i,i'} = \prod_{k=1}^{L} \frac{|v_{i_k,j_k}|}{|v_{i_{k+1},j_k}|}$$

Proposition 32 of the original paper [10] proved that utility profiles can be recovered using the mentioned relation and influences:

Theorem 3.3 *Fix a division problem (v,b) and a graph G. We use N^i to represent all the agents inside the connected component of the agent i in the graph G. And $\bar{u}_i$ is the utility profile for the equal allocation where each agent gets allocated to $1/n_j$ share of the chore j where n_j is just the total number of agents it is connected to, or its degree in the graph theory's sense. If there exists a competitive allocation $\mathbf{z}$ with the consumption graph $G_z = G$, the following formula holds for $u = u(\mathbf{z})$*

$$u_i = (\frac{b_i}{\sum_{i' \in N^i} b_{i'}}) \cdot \sum_{i' \in N^i} \pi_{i,i'} \cdot \bar{u}_{i'}$$

We can stop worrying now as finding connected components in a bipartite graph is not hard. A depth-first search algorithm suffices and its details will be shown in the next chapter. The influences between agents can also be found during the process, hence this formula can be computed efficiently.

3.3.3 Recovering the competitive allocation and prices using its variational characterisation

The third item of Theorem 3.2 tells us that whether a given allocation is competitive or not can be determined by its utility profile. It is another remarkably valuable suggestion because it provides both a way to check an allocation's competitiveness and a way to find the competitive allocation via solving an optimisation problem depending on the freshly produced **u** and **b**. What is more ideal is that we are able to transform the optimisation problem to a graph problem by the following theorem:

Theorem 3.4 (Corollary 35, [10]) *The following statements are equivalent:*

- *An allocation z maximizes the weighted utilitarian welfare $W_{\tau(\boldsymbol{u},\boldsymbol{b})}$*
- *G_z is a subgraph of $G_{\tau(\boldsymbol{u},\boldsymbol{b})}$*

And the problem can be transformed again into our familiar maximal flow problem, inspired by the construction in [20]:

Construct a network $N(v,u,b)$ by adding a source node s and a terminal node t to a complete bipartite graph with parts ([n], [m]): the source s is connected to all the agents [n] and the terminal node t is connected to all the chores [m].

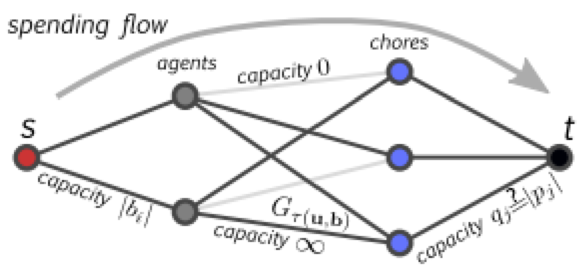

Figure 3.1: An example of a constructed network from [10]

Let $q_j = \min_{i\in[n]} |\frac{b_i \cdot v_{i,j}}{u_i}|$ denote the minimal weighted disutility.

The capacity of each edge $w(s,i), i \in [n]$ is the budget b_i; for all edges (i,j), we set $w(i,j) = \infty$ if this edge exists in $G_{\tau(u,b)}$, and $w(i,j) = 0$ otherwise; we set $w(j,t) = q_j$ for all edges $(j,t), j \in [m]$. Note that no flow $\mathbf{F}$ in this network can exceed the amount $\sum_{i\in[n]} b_i$, which is the total capacity of all edges (s,i).

And the recipe of checking a utility profile's competitiveness as well as to compute a allocation emerges.

Theorem 3.5 (Proposition 36, [10]) *A utility profile $\mathbf{u}$ is competitive if and only if the two following conditions are satisfied:*

- $\sum_{i\in[n]} b_i = \sum_{j\in[m]} q_j$
- *a maximal flow $\boldsymbol{F}$ in N(v, u, b) has magnitude $\sum_{i\in[n]} b_i$.*

Any such flow defines a competitive allocation $z = z(\boldsymbol{F})$ by $z_{i,j} = F_{i,j}/q_j$, prices $p_j = q_j$ with $u = u(z)$ and vice versa.

We have finally reached our goal of getting all the competitive utility profiles and their corresponding allocations.

Chapter 4

Implementation details

In this chapter, we demonstrate our implementation details of the algorithm introduced previously. We have chosen Python as our working language because of its simple syntax leading to legibility, its support for both object-oriented and procedural paradigms that we have used, and its comprehensive libraries facilitating our programming, especially in phase I.

4.1 Architecture choices

4.1.1 Adopting an agent-based approach

Recall Definition 3.1, a chore division problem is determined by a matrix of values and a vector of budgets. While processing matrices and vectors directly is fast, the obscurity and inflexibility arising from hard-coding makes this representation far from ideal. Since both values and budget are generated agent by agent, it does not cost much more space to store them separately in our created class **Agent**, with which it suffices to define the problem.

Such an agent-based approach provides many merits: 1) it models real-world scenarios faithfully and offers us an individual perspective that is highly useful for us to evaluate the algorithm's feasibility in terms of agents' personal experiences like privacy or incentives; 2) we are given a more intuitive and more secure way to look up and change an agent's values and budget by calling the agent directly, without the need to get access to the total information set and risk mixing up different elements; 3) we can define frequently used methods such as *evaluate* to avoid repetitive statements in our code.

class Agent
name: string values: int[m] budget: int
evaluate: return the total dis-utility generated by the received bundle

4.1.2 Connecting the three phases

As we have seen in Chapter 3, the workflow of the algorithm is rather straightforward: we generate a rich family of graphs first, then we can simply use a "for" loop to compute a candidate utility profile for each graph and check its competitiveness.We defined four major functions $GRAPHGENERATOR$, $DUALGRAPHGENERATOR$, $PROFILE$, and $IFCOMPETITIVE$ to serve these purposes.

Suppose we have N agents and M chores in the problem. Running the following algorithm shall give us an answer:

Algorithm 1 Compute all the utility profiles and one corresponding allocation

```
1: $n \leftarrow N$
2: $m \leftarrow M$
3: $chores \leftarrow range(m)$    ▷ We need give each chore a distinct label, w.l.o.g. they are tagged by integers here.
4: $agents \leftarrow [Agent1, ...AgentN]$    ▷ We have omitted the procedure of constructing those agents.
5: if $m > n$ then
6:     $graphfamily \leftarrow GRAPHGENERATOR(agents)$
7: else
8:     $graphfamily \leftarrow DUALGRAPHGENERATOR(agents)$
9: end if
10: for $graph$ in $graphfamily$ do
11:     $profile \leftarrow PROFILE(graph, agents)$
12:     $result \leftarrow IFCOMPETITIVE(profile, graph)$
13: end for
```

4.1.3 Different graph representations

Since our algorithm is centred around graphs, it is important to for us to pick the right representations of them in order to achieve each of our goals most efficiently. As our consumption graphs are both undirected and bipartite, storing them as either chores' or agents' adjacency lists will take up the least space.

Based on this idea, we defined the following class **Graph** and used a dictionary to represent every chore's adjacency list. The choice of using dictionaries instead arrays is due to the former's flexibility. Hard coding numbers is always very confusing as it requires agents be remembered by an order, therefore dealing with arrays directly should be avoided. And the reason for using chores' adjacency lists instead of agents' adjacency lists is that it makes calculating a chore node's degree easier as it will just be the length of its list. The corresponding method is defined mainly for Phase II, getting one of the ingredients to compute the candidate utility profiles. The other two methods $update$,$ifLinked$ will also be widely used. In order to save up more space, the adjacency list remembers only the name of each agent rather than its entire inner structure. Hence, we have also defined a separate function to call the actual agent by its name, if the list of real agents like $agents$ appeared in Algorithm 1 is given.

class Graph
edges: dict{chore: linked agents' names as a list}
update(chore j, a list of Agents): update a chore's adjacency list
ifLinked(Agent i, chore j): return whether i and j is linked
degree (chore j): return the number of agents connected to j

Because the three phases of the algorithm work fundamentally differently, we have to opt for a different data structure in each phase to accomplish a different goal. In Phase II, we need to generate a new representation containing both agents' and chores' neighbours with agents' values as edge's weights. Even though part of the information is repeated, in order to find all the connected agents fast enough, it is crucial to make agents' neighbours ready to fetch. In Phase III, we need to convert a graph to a network in order to calculate the max-flow. Fortunately, both conversions are very easy, especially with the help of methods in our Graph class. Thus their procedures *graphToList* and *graphToNet* are omitted.

4.2 Phase I: Generate a rich family of "Maximal Weighted Welfare (MWW)" graphs

4.2.1 Generating graphs for two agents

[10] proposed to begin the problem with the simplest two-agent case. Their MWW graph structure have been found by [5]. We ask readers to read the paper if they are interested in the proof and we shall proceed directly to the graph-constructing procedure defined as *BASICGRAPHGENERATOR*.

Reorder all the chores, from those that are relatively harmless to agent 1 to those that are harmless to agent 2: the ratios $v_{1,j}/v_{2,j}$ must be weakly increasing for $j \in [m]$. Then immediately we are given two types of graphs in MWW(v):

- $k/(k+1)$-split, for $k \in [m]$: agent 1 is linked to all chores $1,...,k$ (if any) and agent 2 is linked to all remaining $k+1,...,m$. No other edges exist.
- k-cuts, for $k \in [m]$: agent 1 is linked to chores $1,...,k-1$, agent 2 to chores $k+1,...,m$, and all chores j for which $\frac{v_{1,j}}{v_{2,j}} = \frac{v_{1,k}}{v_{2,k}}$ are connected to both agents. No other edges exist.

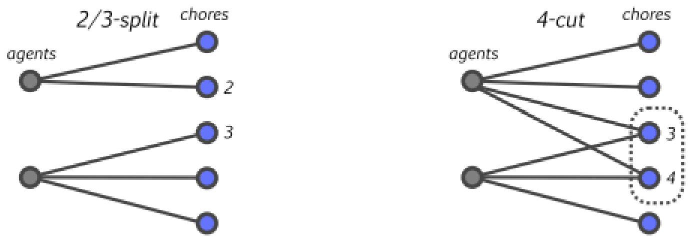

Figure 4.1: Examples for a split and a cut taken from [10]

Lemma 4.1 ([10]) *For two agents, any graph $G \in MWW(v)$ is either a $k/(k+1)$-split or a k-cut. Any k-cut is contained in MWW(v). A $k/(k+1)$-split is contained in MWW(v) if and only if one of the following holds: k = 0, or k = m, or $v_{1,k}/v_{2,k} < v_{1,k+1}/v_{2,k+1}$.*

Therefore, there should be maximally $2m+1$ graphs ($m+1$ split graphs and m cut graphs) generated according to the paper. We discarded the 0-split and m-split graphs too because of our implicit no-idle-agent requirement. Using the hindsight from the next section that an agent and a chore are linked in the general setting if only they are linked in graphs generated for every agent pair, any graph based on a 0-split or m-split will lead to some agent assuming nothing. We have now reduced the number of graphs to $2m-1$ in the basic case. Our algorithm's running time is going to be shorter than what the paper suggested.

According to the paper's suggestion, since m cut graphs are all contained in the set, the graph number is also in $\Theta(m)$. However, we have found out that there could be duplicates among those cuts. Those scenarios depend largely on value functions so we are afraid that no fixed new bounds can be given here. More details are waiting in the fifth chapter.

Algorithm 2 Generating MWW(v) graphs for two agents

```
 1: procedure BASICGRAPHGENERATOR(Agent a, Agent b)
 2:     $ratios \leftarrow a.values/b.values$
 3:     sort $chores$ according to the ascending order of $ratios$
 4:     $graphs \leftarrow []$
 5:     for k in range(m) do
 6:         $cut \leftarrow Graph(chores, [[a.name]$ if $i \leq k$ else $[b.name]$ for i in $chores])$
 7:         for i in range(m) do
 8:             if $ratios[i] = ratios[k]$ then
 9:                 cut.update($chores[i], [a.name, b.name]$)
10:             end if
11:         end for
12:         $graphs$.append($cut$)
13:         if $k > 0$ and $ratios[k-1] < ratios[k]$ then
14:             $split \leftarrow Graph(chores, [[a.name]$ if $i < k$ else $[b.name]$ for i in $chores])$
15:             $graphs$.append($split$)
16:         end if
17:     end for
18:     return $graphs$
19: end procedure
```

4.2.2 Reducing the problem to two-agent sub-problems

A small spoiler in the last section has revealed the recipe to generate graphs when $n \geq 3$. We will have to work on $n(n-1)/2$ auxiliary two-agent problems first, where a pair of agents i, i' divides the whole set of chores [m] between themselves. Then we pick an MWW graph $G^{\{i,i'\}} \in MWW(v^{\{i,i'\}})$ for each pair of agents and construct a graph G for the original problem by the rule mentioned before: there is an edge between agent i

and a chore j if and only if this edge is presented in $G^{\{i,i'\}}$ for all agents $i' \neq i$. After enumerating over all combinations of graphs $G^{\{i,i'\}}$ we obtain a set $\mathcal{G} = \mathcal{G}(v)$.

We can visualise the process with a large table, where each row represents the MWW graphs generated by one pair. Numeration is done through choosing a graph from each row. For preparation, we also need to remember which pair each row corresponds to, and the number of graphs each row contains. The number of enumeration times goes exponential to $(2m-1)^{n(n-1)/2}$ as there can be $O(2m-1)$ choices for each pair, and $n(n-1)/2$ pairs in total. Only when we fix the number of agents n and treat it as an exponential can the algorithm be declared polynomial.

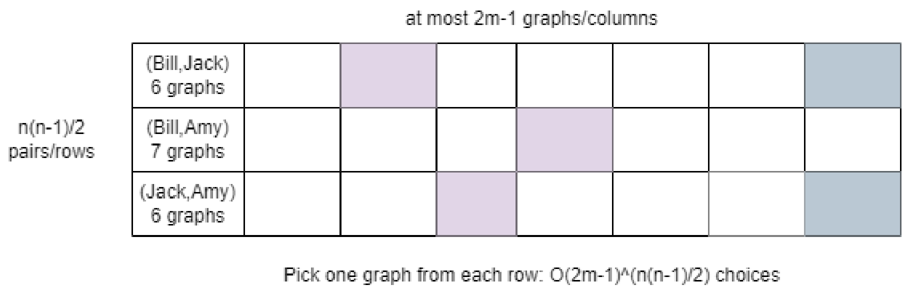
at most 2m-1 graphs/columns
n(n-1)/2 pairs/rows
(Bill,Jack) 6 graphs
(Bill,Amy) 7 graphs
(Jack,Amy) 6 graphs
Pick one graph from each row: O(2m-1)^(n(n-1)/2) choices

Figure 4.2: An illustration: Bill, Jack and Amy are dividing four chores. One of the final graph is being generated when enumerating on the indices (1,3,2)

We used Python's `Itertools` library to facilitate the procedure: `combinations(agents,2)` gives us all the pairs of agents; `product` gives us all the Cartesian product of the arrays which is needed to go over all pairs.

Algorithm 3 Generating MWW(v) graphs for over two agents

```
procedure GRAPHGENERATOR(agents)
    for every pair $(a,b)$ in $combinations(agents,2)$ do
        $pairgraphs \leftarrow BASICGRAPHGENERATOR(a,b)$
        $superset$.append($pairgraphs$)
        $labels$.append(($a.name, b.name$))
        $indices$.append (range(len($pairgraphs$)))
    end for
    $graphs \leftarrow []$
    for every combination in $product(*indices)$ do ▷ pick one graph for each pair
        $finalgraph \leftarrow$new $Graph()$
        for every pair of agent and chore $(i,j)$ do
            $hasEdge \leftarrow$ True
            for every $graph$ whose $label$ contains $i.name$ do
                if not $graph$.ifLinked($i,j$) then
                    $hasEdge \leftarrow$False
                    break
                end if
            end for
            if hasEdge then
                add an edge between $(i,j)$ to $finalgraph$
            end if
        end for
        $graphs$.append($finalgraph$)
    end for
    return $graphs$
end procedure
```

4.2.3 Swapping agents and chores when necessary

Following the discussion in 3.3.2 about Agent-Chore duality, we automatically get another algorithm to compute a rich graph family by swapping chores and agents. Our *DUALGRAPHGENERATOR* procedure starts with creating a dual problem and then simply feeds it into *GRAPHGENERATOR*. Note graphs defined by the Graph class are dictionaries with chores being keys and their connected agents the values. We need to swap the agents and chores back so as to keep the same format.

Now we have two choices at hand and we can pick the method that produces fewer graphs to save time.

Theorem 4.1 *The number of competitive utility profiles is at most*

$$\min\{(2m-1)^{n(n-1)/2};(2n-1)^{m(m-1)/2}\}.$$

4.3 Phase II: Compute the candidate utility profile for each graph

After obtaining all the consumption graphs, we enter the second phase to generate a candidate disutility profile for each graph. Recall Subsection 3.33, the major challenge is to find the connected components of each graph and to calculate one agent's influence over another. It is enough to find $\pi_{i_0,i}$ for a fixed i_0 in each connected component of G and then define $\pi_{i,i'}$ as $\frac{\pi_{i_0,i'}}{\pi_{i_0,i}}$. The original paper proposed to imitate a shortest path algorithm like Bellman-Ford, which can deal with negative weights. Since we have changed the sign of all the values, we believe that basing on Dijkstra's solution is better due to its smaller time complexity.

In fact, our algorithm is even easier than Dijkstra's algorithm since we are not pursing an optimised solution like a shortest path. The corollary below guarantees us that if our current graph has a feasible competitive allocation,the influences will be path-independent as two different paths will form a cycle with product being one. And if the graph does not have a feasible competitive allocation, it will be eliminated anyway.

Corollary 4.1 *If an allocation z is Pareto optimal and its corresponding consumption graph G_z contains a cycle C, then $\pi(C) = 1$.*

Although we want only the connected agents, we have to treat chores and agents equally during the whole searching and updating process since two agents are only connected via a chore in the bipartite graph. The connected chore nodes will be cut down in the last step.

Algorithm 4 Tracing all the connected agents of agent i

```
procedure CONNECTEDAGENTS(Graph $G$, agents, (source node) agent i)
    $weights \leftarrow GRAPHTOLIST(G, agents)$
    $unvisited \leftarrow \{node$ :None for $node$ in $chores + agents\}$
    $visited \leftarrow \{\}$
    $current \leftarrow i$
    $currentPi \leftarrow 1$
    $unvisited[current] \leftarrow currentPi$
    while True do
        for $neighbour$, $weight$ in $weights[current].items()$ do
                    ▷ Iterating through the connected nodes of the current node
            if $neighbour$ not in $visited$ then
                if $neighbour$ is an agent then
                    $unvisited[neighbour] \leftarrow currentPi/weight$
                else
                    $unvisited[neighbour] \leftarrow currentPi \cdot weight$
                end if
            end if
        end for
        $visited[current] \leftarrow currentPi$
        delete $unvisited[current]$
        if $unvisited$ is empty then
            break
        end if
        $newcurrent \leftarrow [node$ for $node$ in $unvisited.items()$ if $node[1]$ != None]
        if $newcurrent$ !=[] then
            $current, currentPi \leftarrow newcurrent[0]$
        else
            break
        end if
    end while
    for $node$ in $visited$ do
        if $node$ is not an agent then
            $visited$.pop($node$)
        end if
    end for
    return $visited$
end procedure
```

Now we are ready to calculate the disutility profile. Recall the formula:

$$u_i = \left(\frac{b_i}{\sum_{i' \in N^i} b_{i'}}\right) \cdot \sum_{i' \in N^i} \pi_{i,i'} \cdot \overline{u}_{i'}$$

It is easier to calculate it by parts as $u_i = (\frac{b_i}{sumb}) \cdot rightsum$, where the $rightsum$requires us to calculate the utility profile for agents by diving chores equally. We name this special utility profile as *eutilities*.

Algorithm 5 Recovering the utility profile

```
1: procedure PROFILE(Graph G, agents)
2:     calculate $eutilities$
3:     $utilityprofile \leftarrow \{\}$
4:     $unvisited \leftarrow agents.copy()$
5:     while $unvisited$ is not empty do
6:         $current \leftarrow unvisited[0]$
7:         $connected \leftarrow CONNECTEDAGENTS(graph, current)$
8:         $sumb \leftarrow$sum([$agent.budget$ for $agent$ in $connected$])
9:         for $agent$ in $connected$ do
10:            $rightsum \leftarrow$ sum([$(connected[i]/connected[agent]) \cdot eutilities[i]$ for i in $connected$])
11:            $utilityprofile$.update($\{agent : (agent.budget/sumb) * rightsum\}$)
12:            $unvisited$.remove($agent$)
13:        end for
14:    end while
15:    return $utilityprofile$
16: end procedure
```

4.4 Phase III:Check each candidate profile for competitiveness

As indicated by Theorem 3.5, the goal of this phase is to check two conditions. Since $\sum_{i\in[n]} b_i = \sum_{j\in[m]} q_j$ is easier to check, after finishing computing q, we run a loop first to test this condition. If it is satisfied, we move forward to solve the maximal flow problem. The paper suggests the Edmonds-Karp algorithm which has linear running time in the number of vertices, and quadratic in the number of edges. We decided to implement Dinic's algorithm instead as it includes some additional techniques that reduce the running time from $O(|V||E|^2)$ to $O(|V|^2|E|)$ i.e.,$O(mn(n+m)^2)$ here. Since Dinic's algorithm is very well-known, and there is no hardness in checking the two conditions, we have placed the pseudocode for this part in Appendix A.

4.5 The complexity is determined by the combinatorial enumeration

4.5.1 A running time analysis for the entire workflow

Recall the workflow presented in 4.1.2, the algorithm's running time is the sum of time spent on or $DUALGRAPHGENERATOR$ in Phase I and the time spent on $PROFILE$ and $IFCOMPETITIVE$ for each graph times the number of graphs in the loop.

In Phase I, we begin by calling $BASICGRAPHGENERATOR$ to generate graphs for every pair of agents. This method will do $O(m \cdot \log m)$ operations to sort the chores depending on the ratios of two agents' values and then it can generate the rich family of O(m) graphs each with constant operations. Therefore, the total time of running it is $O(m \cdot \log m + m) = O(m \cdot \log m)$.

Hence the number of operations needed to generate the super-set of graphs is $O((m \cdot \log m) \cdot (\frac{n(n-1)}{2}))$ Then to create graphs $\mathcal{G}$ by numerating on the super-set, we exhaustively look over $O(m^{\frac{n(n-1)}{2}})$ combinations. And for every pair of agent and chore (mn pair in total), we need to check $n-1$ graphs that belong to the agent. The total number of operations is $O(m^{\frac{n(n-1)}{2}} \cdot mn \cdot (n-1))$. Hence, if n is fixed, we view it as a constant and the complexity reduces to $O(m^{\frac{n(n-1)}{2}+1})$.

$DUALGRAPHGENERATOR$ works in the very similar manner, differing from the $GRAPHGENERATOR$ by an extra step of swapping agents and chores first, which can be finished in $O(mn)$ time by simply transposing the value matrix. Hence the complexity of $DUALGRAPHGENERATOR$ is $O(n^{\frac{m(m-1)}{2}+1} + mn) = O(n^{\frac{m(m-1)}{2}})$.

In Phase II, we will run $GRAPHTOLIST$ first to convert the graph defined in our Graph class to a weighted adjacency list, which can accomplished by $O(nm)$ operations. Then we want to compute the disutility profile for every agent, which only takes constant steps after we have found all the connected components and calculated the influences between them (hence $O(n)$), the latter requires $O((n+m)^2)$ operations since there are possibly $n+m$ nodes to loop through and for every node it is possible to check $n+m$ nodes in the depth-first search algorithm we have defined. Thereby, the total running time is bound by $O(nm + (n+m)^2 + n) = O((n+m)^2)$.

In Phase III, it takes $O(mn)$ operations to convert the graph to a network using our method net. Then we run the dinic algorithm whose complexity is $O(nm(n+m)^2)$. Hence, the complexity of this step is bound by $O(mn + nm(n+m)^2) = O(nm(n+m)^2)$.

In this way, we can see clearly the time complexity of the last two phases of the algorithm is bounded by $|\mathcal{G}| \cdot (O((n+m)^2) + O(nm(n+m)^2)) = |\mathcal{G}| \cdot O(nm(n+m)^2)$, where $|\mathcal{G}|$ is bounded both by $(2m-1)^{\frac{n(n-1)}{2}}$ and $(2n-1)^{\frac{m(m-1)}{2}}$. Hence, the complexity can be converted into $O(m^{\frac{n(n-1)}{2}+3})$ for fixed n or $O(n^{\frac{m(m-1)}{2}+3})$ for fixed m, which indeed agrees with the result shown in the paper (see Theorem 3.2) even though we have amended part of the procedure in each phase.

4.5.2 Limited improvement can be done in Phase I

We would like to convince the readers that the space for improving the algorithm is very limited. In first phase, we cannot avoid the combinatorial checking and for every pair we will have minimally m graphs due to the inclusion of all k-cuts. For phase II and III, we have already adopted the most state-of-art algorithms for finding the connected components and computing the max-flow.Hence it is very unlikely to improve these procedures too. The only place we can do a trick is to somehow reduce the number of graphs in the loop.

Therefore, we decided to

1. remove graphs if some chore remains unallocated;
2. remove graphs if some agent is not allocated to any chore, therefore being idle;
3. remove duplicated graphs

after running the *GRAPHGENERATOR*. These three checks are very easy to implement. Although they seem trivial, the experiment result showed that after these three checking, we successfully got rid of a great number of graphs, a very good news for real life. We are entering Chapter 5 now to see more details.

Chapter 5

Exploring the algorithm’s feasibility

How feasible is it to use our algorithm for computing competitive chore allocations? And how feasible is it to employ the algorithm as a “fair” allocation mechanism? After running benchmark tests on all types of problems that appeared in our experiments (these tests are in alignment with the competitive allocation’s definition: we check if all partitions of chores sum up to one, if it satisfies Envy-Freeness, and if everyone fulfills her liability (i.e. spends all her budget)), we were ensured the correctness of our implementation and turning to these big questions eagerly.

Centring around different aspects of feasibility, we organised this chapter into four parts. In the Integrity and secrecy section, we ask: can we always make sure no attacker has changed anything? And can we protect the agents’ privacy at the same time? Then we wonder, how fast can we get the solutions? How big our problem is allowed to be? Following the time complexity discussion, we investigate if agents have incentives to lie in the process. In the end, we recognise that competitive allocation of chores has the problem of potentially yielding no solutions or multiple very discontinuous solutions and start to explore how bad the situation can be.

Because no similar approach has been taken before to evaluate competitive allocation of chores with such concreteness, we prioritised the diversity of our research directions to see if our problems were worth investigating instead of trying to give a very thorough and systematic analysis on one aspect.

5.1 Integrity and secrecy

5.1.1 Publishing budgets and allocation results is necessary and sufficient for ensuring the computation’s integrity

As mentioned at the beginning of this chapter, three conditions need be checked to guarantee the computation’s correctness. Because some attacker may have changed some agent’s values or budget in the system, we need to ask agents themselves to validate the allocation’s integrity. Each agent’s values are probably her most sensitive information and fortunately, they do not need to be exposed as we can simply ask every agent for her envy-freeness. The resulted allocation bundles, prices and each agent's budget have to be public in contrast as one agent can only decide if she envies others' bundles based on the ratio of their budgets. There is no point for an agent with double responsibility to say she is envious of someone with single unit of responsibility. The other two conditions also take partitions, budgets and prices into account.

Publishing these statistics also brings about other benefits. As we discussed before, one part of what makes competitive rule fair is that the budgets are agreed beforehand. A lack of transparency certainly provokes unfairness at the first place. Broadcasting every agent's allocation is also often trivial since not everyone does her chores secretly. And it helps agents supervise each other finishing their duties so that the allocation will not become meaningless.

5.1.2 Values can remain private

We have not yet made sure that individual's values will not be revealed from the unveiled information. Now we would like to guarantee readers that it is true as we have found infinite (v, b) pairs corresponding to the same allocation.

Let us consider a simple scenario: four people are dividing two chores and they all have the same opinions towards the chores and the same amount of responsibility. We may define the problem by $v = [[1,1],[1,1],[1,1],[1,1]]$ and $b = [1,1,1,1]$. Our algorithm gives us a partition $[0.5,0],[0.5,0],[0,0.5],[0,0.5]$ and together with two prices [2,2]. If we double everyone's budget, i.e. $b = [2,2,2,2]$, we get the same partition with prices being [4,4]. This is expected as the budgets are viewed as a virtual currency with no intrinsic value in the competitive rule. As long as the ratios of budgets remain the same, nothing changes except the prices' absolute values. The sum of budgets always equals to the sum of prices as we set the unit of every chore to be one.

What may seem more surprising is that if now $v = [[1,1],[2,2],[3,3],[4,4]]$ and $b = [1,1,1,1]$, the resulted allocation is still the same. Similar to what Moulin [38] believed, "but the intensity of preferences is immaterial; the intensity cannot be measured objectively and thus must be ignored.", the rule does not distinguish two agents if their values only differ by a scalar and their budgets are equal. When their bundles are swapped, the new allocation is still competitive. Furthermore, if two agents' values differ by a scalar, their disutilities are determined by that scalar times the ratio of their budgets: $\frac{U_i}{U_j} = \frac{V_i}{V_j}\frac{B_i}{B_j}$. In this case, the agent's subjective measurement of pain is disregarded during allocation. Even when the chores really cause the second agent twice the trouble than the first one due to the fact that their budgets are equal, the second agent still has to face the doubled disutility. We have illustrated a slightly more complicated instance C.1 in the appendix. Readers are invited to check the correctness of our proposed equation.

Note that this property also discourage a specific kind of strategic manipulation since reporting the values as v_1 or $2v_1$ does not matter to the final share (see exercise 2.8 in [37]). But we need to stress that the ratio has to be the same for every chore. Otherwise, other chores will step in and change everything.

5.2 Time complexity

Due to the time limit, the discussion from now on is restricted to the *equal income* case with all agents' budgets set to one. We hypothesised that three factors could influence the algorithm's running time: the number of chores m, the number of agents n, and the value functions of v. In the sections below, we have labelled cardinal value functions as $[a,b]$, meaning every agent was free to express its disutility towards one chore using a natural number inside $[a,b]$. Hence, our input values vectors were generated using `numpy.random.randint(a,b+1,m)` for each agent. For ordinal value function labelled simply as *ordinal*, we meant that every agent had to order their preferences from 1 to m. The lower the value indicates the less disutility the chore can bring about to the agent. So the agent prefers it more. The inputs in this scenario are generated by setting agents' values: `numpy.random.permutation(range(1,m+1))`.

Knowing an algorithm's running time depends on a machine's central processing unit (CPU)'s power, we have run all the experiments over the same machine — model name: Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz CPU(s): 48.

5.2.1 Using the duality trick and removing bad graphs are critical

Before diving into two main results, we would like to use this subsection as a starter to show readers that both the duality trick and graph removals are very important for speeding up the algorithm.

In Figure 5.1, we illustrate that due to the power of $(2m-1)^{\frac{n(n-1)}{2}}$ or $(2n-1)^{\frac{m(m-1)}{2}}$ grows fast, the duality trick is very effective even for small size of problems. The right graph plotted the actual running time of the algorithm when $n=5$ with values in $[1,5]$. We notice that even when m was as small as four, the running time was over one hour ($2^{12}=4096$ seconds ≈ 68.3 minutes) without using the duality trick. But it could be finished within seconds otherwise. This is indeed predicted by the difference between the number of graphs generated by different algorithms. The left graph presents the maximal numbers of graphs generated by two different graph-generators theoretically and the lines look very similar to what are on the right.

Figure 5.2 breaks down the MWW graphs produced by *GRAPHGENERATOR* (left) and *DUALGRAPHGENERATOR* (right). As only the red part of the pie represents the graphs useful for our computation, it highlights the efficiency of graph removal. Moreover, it tells us how the duality trick works. In the first row, as $n=m=3$, the numbers of generated graphs are the same. Graphs with idle agents correspond to graphs with unallocated chores after using duality, and the reverse is the same. Things change a bit when n,m become unequal as it means the numbers of graphs generated are determined to be different. The total graph numbers differ in scale in the second row although the number of "distinct answers" are similar (32 versus 23), This is because in either case the generated graph family is rich. Hence their difference is in proportion: only 0.1% graphs generated are needed on the left, compared to 9.0% on the right, which is still quite small.

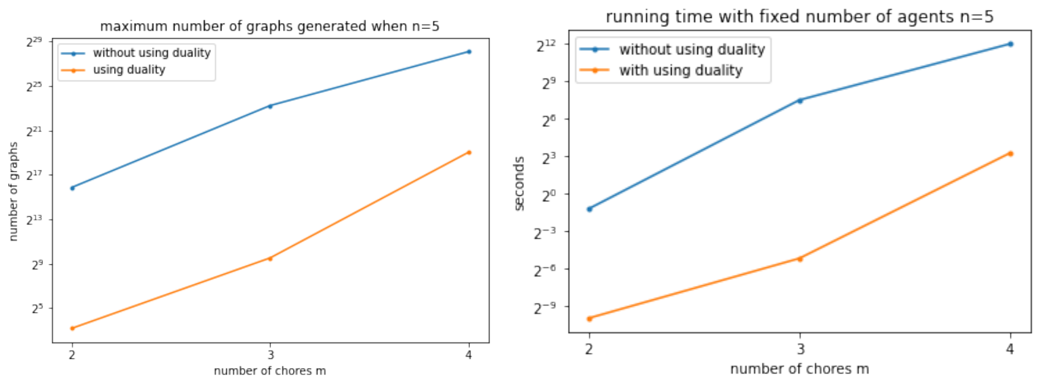

Figure 5.1: Duality is essential. The left and right graph's similarity reflects that time complexity is dominated by the combinatorial enumeration.(average of ten trials)

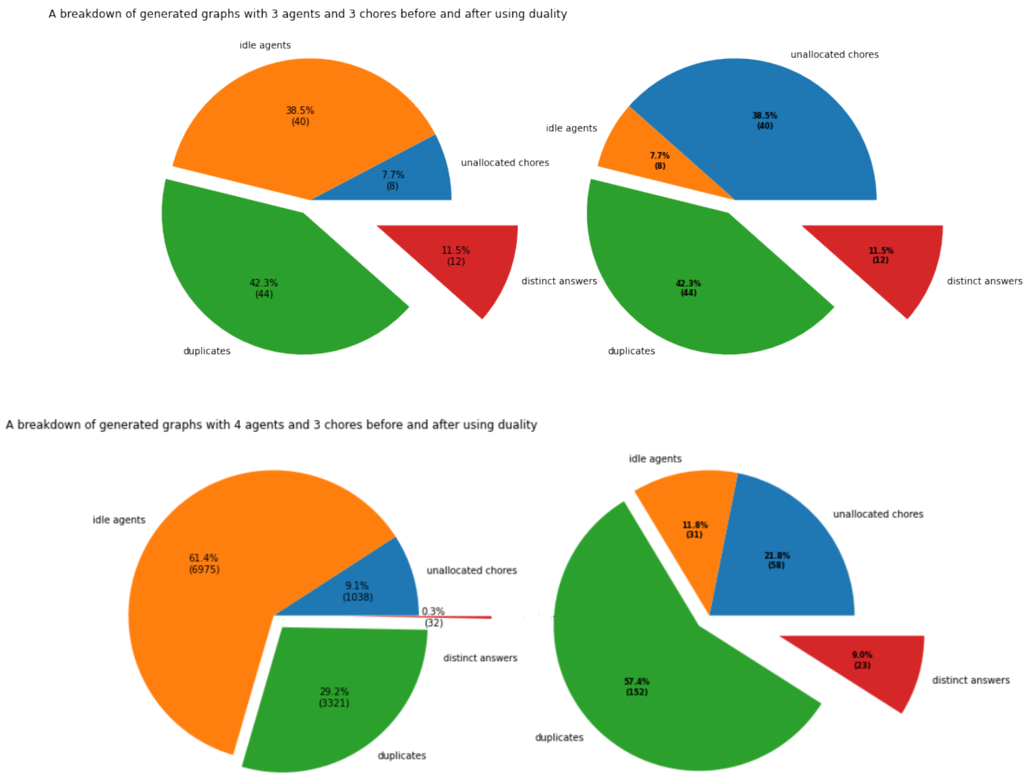

Figure 5.2: A breakdown of generated graphs. Value function $[1,4], n, m = 3,3; 4,3$. (average of ten trials)

From now on all the experiments would be run by the improved algorithm. As we have seen, the two tricks are extremely powerful, without which very limited problems can be solved. Therefore we should only consider this version in practice.

5.2.2 Value function influences the complexity by its expressivity

Motivated by the fact that agents' freedom of expressing their values have to be limited, we carried out experiments by controlling the size of the problems and observed how different value functions impacted on the allocation. The results are obtained by averaging twenty trials. Due to the limit of space, experiment results when n or $m = 4$ are put in the appendix B.4. The graphs here should be enough to tell the story.

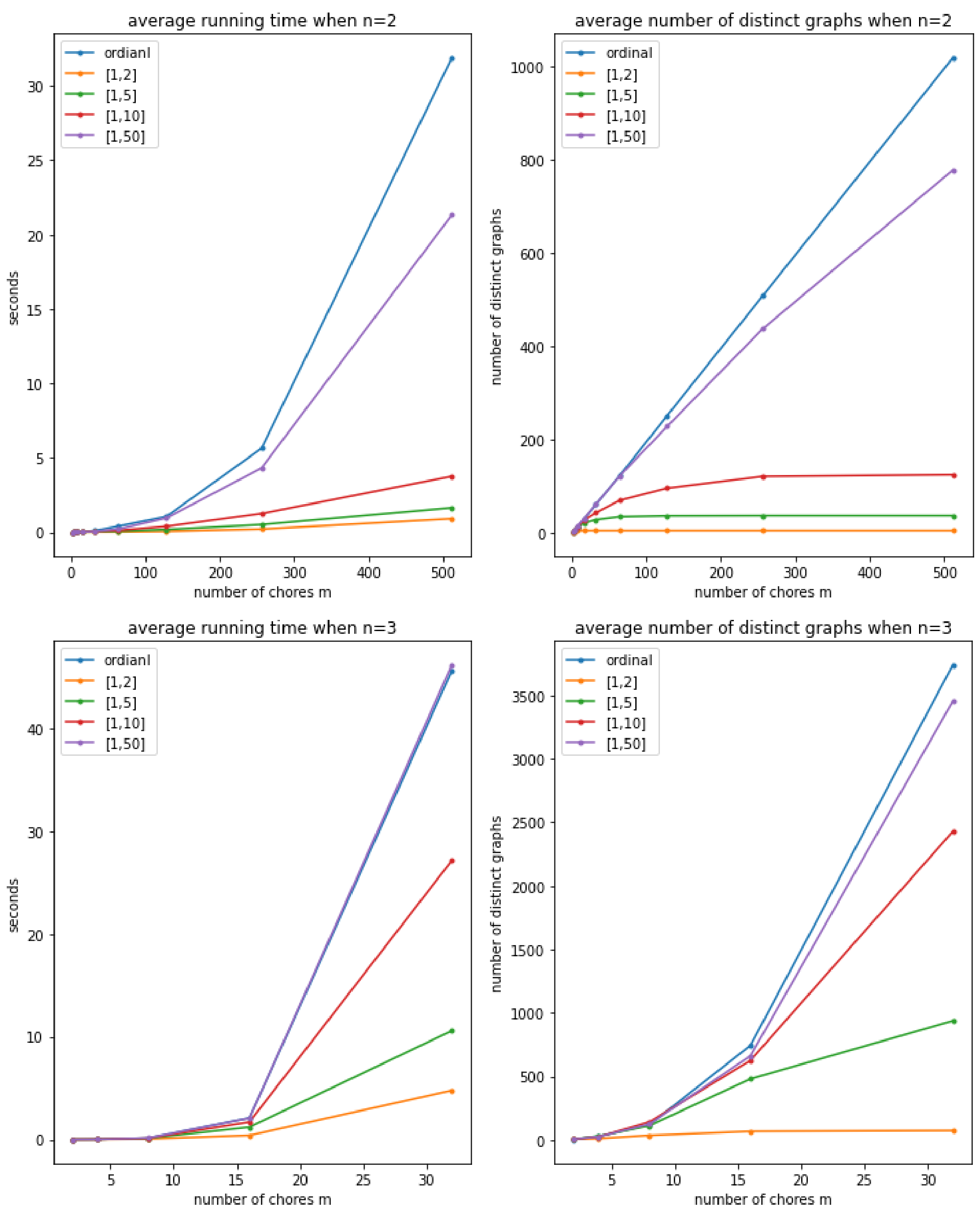

Figure 5.3: Fixing the number of agent, different value functions result in the running time and the number of distinct graphs generated in phase I increasing at very different rates.

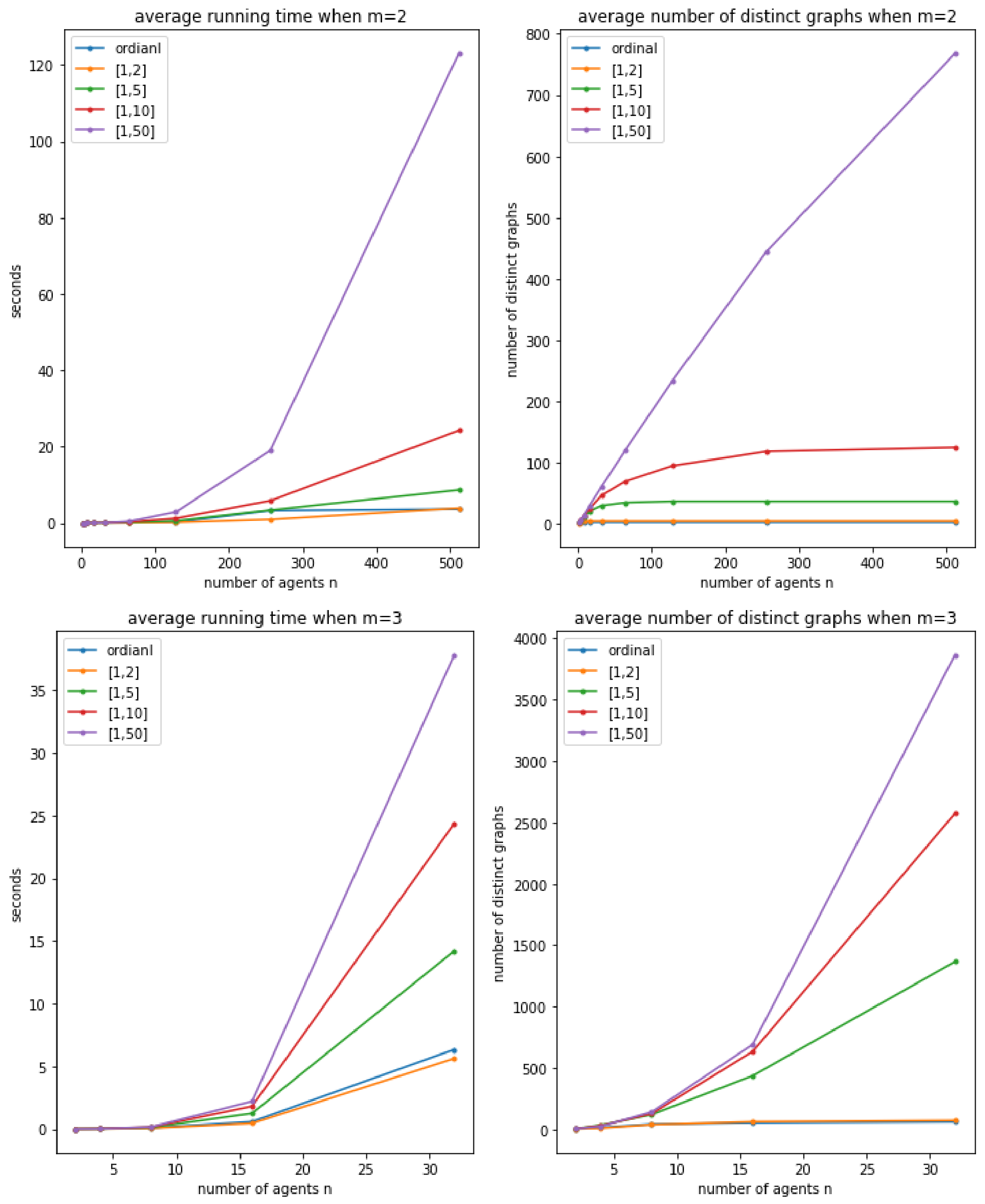

Figure 5.4: Fixing the number of chores, the effect of value functions on running time and the number of distinctly generated graphs is very similar to the previous case.

The first conclusion is that the number of distinct graphs is a determinant of the algorithm running time. Apart from the big difference brought by the value functions, as shown by 5.3, the number of distinct graphs remaining constant while the number of chores are increasing also seems quite counter-intuitive. However, after a little bit analysis, we managed to convince ourselves that this is expected. Recall that the graphs are generated according to the ratios. However, if the value for every chore is either one or two, the ratio of the agents values about one single chore only has three possibilities:1/2, 1 and 2. Therefore, we will only obtain two k-splits at where the ratio is changing from 1/2 to 1 and 1 to 2, and only three k-cuts as the chore with the same ratio will all be connected to the two agents. Similarly, for the cardinal function [1,5], there are 19 different ratios in total and lead us to 18 k-splits plus 19 k-cuts. For [1,10], 55 different ratios result in 109 different graphs...Since our values are generated randomly, only with large enough number of chores all ratios show up and the number of distinct graphs reaches what we calculated. Notice that for ordinal functions, the number of possible ratios is the same as with cardinal [1,m], which increases quadratically as m increases. Hence the blue line corresponding to the ordinal function behaves much like our upper bound estimation: $2m-1$.

Having understood the basic case, we are now able to reason why the algorithm runs in less time when the values are in the smaller range. Since fewer distinct graphs are generated for each pair, fewer distinct graphs are obtained in general. Fixing m and running the algorithm with an increasing number of agents n should give us a similar result since it is only the values who are doing the trick. We should not be surprised to see the ordinal function gives us the fewest graphs as m is now very small.

5.2.3 Scalability is bounded by five, the smaller number of agents and chores

We have found out that both the value function and the problem size play a huge role in deciding the time complexity. Is one factor more dominant than the other? To explore this, we ran the algorithm on problems of different sizes as well as with different value functions. In the graphs below, we plotted what size of the problem could be run within thirty seconds. As we saw previously, with fixed n, the algorithm behaves most differently between the ordinal value function and the small-range cardinal function [1,2]. Comparing their difference shown vertically to the differences between different number of agents or chores, we learn that the problem size is more influential when it comes to deciding the running time.

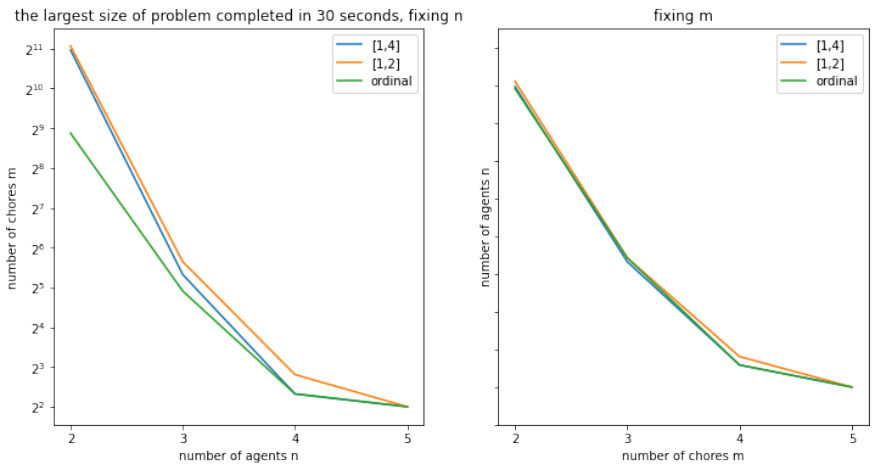

Figure 5.5: The problem size is a more dominant factor on time complexity than the value function's type.

Therefore, to investigate the scalability, we can now control our inputs under the same value function type and focus only on the problem size. Knowing that the number of MWW graphs is bounded by the smaller number of n and m , we can further limit our experiments to the $n = m$ case. We ran one hundred trials for different numbers and plotted the best, average and worst cases. Recall that total complexity is $O(m^{\frac{m(m-1)}{2}})$. Taking the log, we get the quasi-polynomial function $\frac{m(m-1)}{2}O(\log m)$, which looks indeed like the graphs below. Using the dual graph generator gave us a similar effect as expected. We attempted to run the case when $n = m = 5$ but stopped after eight hours without an answer. So we conclude that as soon as both the number of chores and agents reach five, the computation becomes very costly.

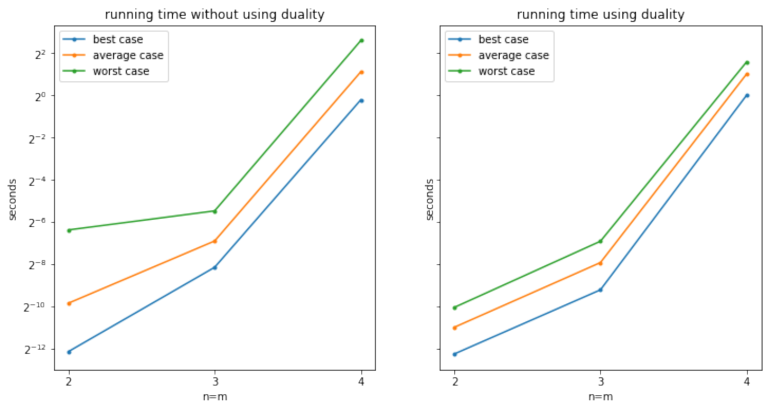

Figure 5.6: Running time when n=m, value function $[1,5]$.

5.3 Incentives for lying

Will people favour a mechanism that encourages dishonest or manipulative behaviour? Back in Chapter 2, we have discussed two kinds of incentive compatibility, namely *Dominant-strategy incentive-compatible* (DSIC) and *Bayesian-Nash incentive-compatibility* (BNIC). We can sense that DSIC is a stronger condition than BNIC since it requires every agent staying honest regardless of others' behaviour.

In this section, we consider two situations where an agent may wish to lie. First, we presumes that an agent only cares to decrease her own disutility. We will start by testing BNIC because it is a weaker condition. By showing that even achieving BNIC is impossible, we argue that DSIC is not satisfied by this algorithm. However, we will still argue that lying in this case is hard.

Next, we study the case where an agent becomes more manipulative and wishes to influence the entire allocation. We focus on how reporting a high value can increase a chore's price and thereby affect the whole allocation. Based on the findings, we propose that decreasing the value function's range or increasing the number of participants can limit manipulative behaviour's influence.

5.3.1 Lying to decrease one's own disutility

It is enough to prove BNIC is not satisfiable with one counterexample. Suppose three agents are dividing two chores. They are only allowed to express the values either by one or two. Agent A and B express their true values [1,1] and [2,1] respectively. And C's true values are [1,2].However, if C has known the other two's values and the access to running the algorithm, C can complete a table of different outcomes resulted from her suggesting different values:

values	resulted allocation	C's actual disutility
[1,1]	A:[0.5, 0] B:[0,1] C:[0.5,0]	0.5
[1,2]	A:[0.5, 0] B:[0,1] C:[0.5,0]	0.5
	A:[0, 0.5] B:[1,0] C:[0,0.5]	1
	A:[1/3, 1/3] B:[0,2/3] C:[2/3,0]	0.67
[2,1]	No result	
[2,2]	A:[0.5, 0] B:[0,1] C:[0.5,0]	0.5

Table 5.1: C is trying out which values will bring her the lowest disutility.

C may find the experiment results a bit surprising — among all the values that do give rise to a competitive allocation, the truthful values actually give C the worst outcome: if C chooses to lie about either the second chore, or the first, C will be allocated a half of the first chore for sure, and end up with disutility 0.5. However, if C stays truthful, since there will be three different allocations to choose from, C will face the risk of increasing its disutility to 0.67 or 1. Therefore, unless C is certain that among the three allocations agents will agree on the one that gives C disutility 0.5 (they won't, for instance, if they stick to the *Egalitarian Equivalent* rule), it is better for C to lie. To put it another way, C actually suffers the most by being honest probabilistically. We have obtained a complete opposite example against incentive compatibility.

Back in Chapter 2, we mentioned that the competitive rule enjoys *Independence of lost bids* (ILB). We can see that this property does take place here: the same allocation A:[0.5, 0] B:[0,1] C:[0.5,0] appears when C's stated values being [1,1] and [1,2] respectively. ILB was favoured by [6] because they thought, since misreporting (on the lost bid at least) would not help much, agents would stop doing so. However, they failed to consider the fact that results are multi-valued. The agent still has an incentive to lie if lying increases the probability of getting a lower disutility.

Although we have seen that lying might decrease one's disutility, we still do not think that the mechanism performs badly on this matter. First of all, lying is not necessarily harmful. As shown in the second table of C.3, everyone except Jack's disutility is lowered if Sarah misreports her value about the second chore as three instead of two. And the sum of disutilities also drops from 3.6 to 3.43. Furthermore, an agent won't know how to lie strategically without having any information about others' values. As C.3 illustrates, the agent gets very different results while mending her values. The uncertainty rising from both the allocation's dependence on the entire value matrix and the solution being multi-valued is the best deterrent preventing agents from lying. This is also why we should keep every agent's values hidden. Finally, we may enforce lying ourselves. When the rule gives us no allocation, what shall we do? Maybe we have to ask someone to distort their values so that we can try again.

5.3.2 Lying to disrupt the allocation

Is there really no strategy for an agent to manipulate the allocation? What if its purpose is to disrupt the allocation rather than to focus on its own interest? For example, by enhancing a chore's price significantly, the adversary might help those who don't mind the chore as now they can fulfill their liabilities (use up its budget) with doing only a little amount of this chore.

We notice that the gap between two chore's prices are in general increasing as Sarah raises her value about the second chore from one to four in C.3. To elaborate on this direct link between a chore's price and the values it has been attached to, we conducted an experiment by retaining almost the whole matrix v and continually increasing Sarah's value on chore 2. Note that due to ILB, we will keep seeing some allocations such as the ones appeared when Sarah lied [1,4]. Focusing only on the newly emerged allocations at each time (this is why there are only few dots left and the line is not very smooth), we get the following pictures:

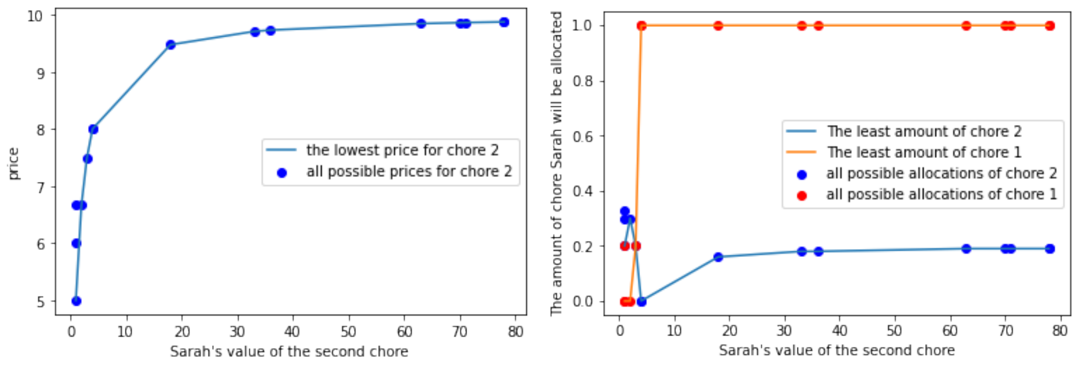

Figure 5.7: The dots represent all the results we get. Since we are getting multi-valued solutions, we draw the line connecting only the lowest price of chore 2 among different allocations on the left, and we draw lines connecting only the lowest amount of chores Sarah needs to do among all allocations on the right.

The first thing we notice is that as chore 2 is being viewed as more and more painful by Sarah, its price soars up until it approximates the entire budget. However, increasing her value for chore 2 does not always help Sarah avoid the chore. Conversely, Sarah is allocated to an increasingly larger partition of it after certain stage (but never reaches 1/5) because as long as the partition is slightly smaller than other's share, her high disutility value on chore 2 guarantees that Sarah won't envy others as she does slightly less, and even together with doing the entire chore 1. This special allocation emerges like [1,0.16],[1,0.17],[1,0.18],[1,0.19] as shown in the right graph of Figure 5.7. And everyone else is allocated an equally small partition of chore 2.

Since a manipulation is so easy to realise, we are motivated to search for solutions to limiting an agent's influence. Our experiment is designed to see how the problem size and the value function play a role. The setting is that the adversary wishes to increase the price of the second chore. So when the value function is cardinal, her strategy is to give chore 2 the highest value she is allowed to put down and set other chores' values the lowest. If only the ordinal value function is allowed, the adversary will tag the chore as the last choice, and randomise her preferences for other chores. We fixed the number of chores m during the experiments.

In the graphs below, lines of different colours match problems with different value functions. Our experiment results were averaged over two hundred cases.

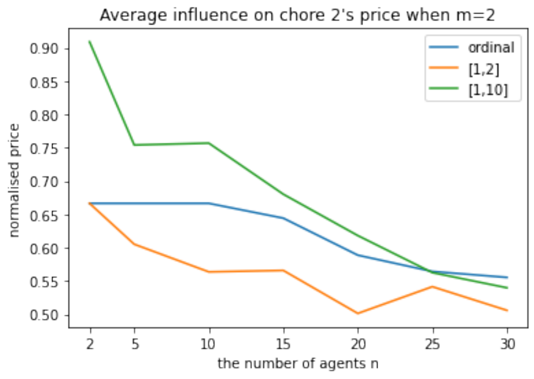

Figure 5.8: An agent's influence diminishes as the problem becomes large.

As we can see, the influence may be contained by us adding more participants in the allocation and choosing a value function with a smaller range. The similar effect was not perceived when we fixing n and increasing m. More experiments should be done to explore that side.

5.4 No solution and multiple solutions

A major problem that arises when chores are allocated competitively is that neither a result is guaranteed nor is it always single-valued. To illustrate the latter case, we listed three allocations with agents' disutilites in the brackets in C.2. Both Amy and Sarah's disutilities vary greatly, from 0.25 to 1 (and one's best is the other's worst) even though Bill won't be affected much. Now let us try to pick a result. To our surprise, no matter whether we reason as a utilitarian (check the total disutilities are 3.5, 3.47, 2.5), or as an egalitarian (check the highest disutility in each case is 1, 1.07,1) or we decided to minimise the Nash product (0.094, 0.097, 0.016), the third allocation actually dominates. Due to this observance, we started to be curious about not only how likely a problem gets no solutions or multiple solutions, but also how likely a multi-valued solution can get a "good" answer.

Our experiment was designed as follows: we fixed the number of agents to three and generated two hundred tests for different number of chores between two and twenty and for different value functions; then we repeated our experiments again with fixed m and

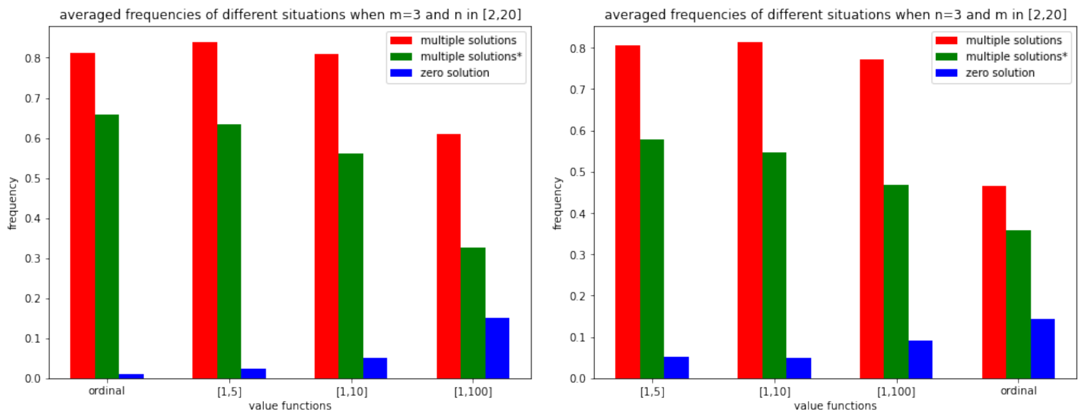

Figure 5.9: We counted the frequencies of different scenarios happened when m or n was fixed as 3 and n or m varied between 2 and 20. The three labels "multiple solutions", "multiple solutions* " , and "zero solution" correspond to three different situations: we got more than one allocation result; we got more than one allocation result and there was a solution satisfying the three conditions among them; we got no allocations.

a increasing number of agents. We were increasing either m or n with the hope that we could perceive any relationship between the problem size and the likelihood of having none or multiple results. We did find that when $n = 2$, increasing the number of chores will lower the likelihood of having solutions (see B.2). However, partly due to much fewer number of chores we were allowed to run for $n = 3$ in the same amount of time as when $n = 2$, we did not see the expected decrease, at least when m was increasing between 2 and 20 (check the original data in the Table B.1). Therefore, we had to add up the answers got with n or m being different numbers between 2 and 20 and present the diagrams below. There are some nice findings, at least for the case when m or n equals to 3 but we are afraid that they cannot be generalised much yet:

- Judging from both graphs, the chance that we can pick out a "nice" solution out of multiple allocations is not low at all, surpassing fifty percent in all cases. And the chance of having no solution is much lower than having multiple solutions for both fixed m and n.
- Since the ordinal function has the smallest range when $m = 3$, the left graph shows that with fixed m, having a value function with a larger range leads to a higher chance with no solution and a lower chance to get a nice answer inside multiple solutions (the gap between the red and green bar is increasing).
- We may need to consider problems with fixed m and n separately as on the right, with fixed n, the likelihood of getting no solution is always higher than what is on the left under the same type of value function. And what happened with ordinal function on the right is also different from the left side— the chance of getting a nice solution out of multiple results seems quite high although the chance of getting no results is also high. As m is always changing, we cannot simply analyse the right graph by different value functions' ranges.

Chapter 6

Conclusion

We have successfully implemented the algorithm and started a comprehensive evaluation about its feasibility. Is this a good application in real life? Our final answer is, like always, "it depends". We warn against attempts on problems with no less than five agents and chores at the same time. But we are clear that if obtaining a result is really important, let patience be the virtue. When asked to put down values as a ranking, agents may complain about the setting being too restrictive, if they are viewing two chores equally or the distinction within one pair of chores is significant larger than one within another pair. Similar complains may be received if the cardinal value function has a small range. However, increasing expressivity may be at cost of an increased running time, a lower chance of getting a solution, and less incentive compatibility. The mechanism designer will have to consider all-roundly.

6.1 Originality and difficulty

Throughout the work, we kept fine-tuning our own understandings. We decided on the implementation details independently so that the algorithm can perform better. We combined a variety of methods to analyse, both idiosyncratically and systematically, the algorithm's performance as well as the solutions' patterns. When exploring the relation between budgets and values, we came up with a formula that links the two together. We initiated the discussion over the incentive for lying out of different motivations, and associated it with normative properties like DSIC or ILB. Our original attention on comparing different value functions is partly due to that we have a concrete program. We discovered how the value function could influence the algorithm's running time.

Partly due to the fact that no similar theoretical discussion or experimental framework on the practical feasibility of competitive allocation of chores is known to us, we found that asking the right questions and designing the right experiments could be challenging. During experiments, finding real patterns was also not easy, for which the limited scalability of the algorithm is to be blamed. While some problems seem fast to solve, repeating them a large number of times is costly. See Figure B.1 for example, after 100000 trials we were still not able to capture the worst case's behaviour because a bad instance happens extremely rarely. Some of our questions' answers might only be attained with a deeper understanding of the algorithm, which we may not be fully equipped with as our effort was tending towards the practical side instead to trying to understand all the theorems.

6.2 Limitations and further work

We believe there is much more space to continue investigating our algorithm's feasibility. Regarding integrity and secrecy, we argued that each agent's values could be held private and publishing the allocation results and budgets would not reveal the information. However, we did not prove that those values would not be recovered if the conditions were more restricted (e.g. we know more about what kind of values each agent is allowed to put down). This may be treated as a separate mathematical question beyond competitive allocation itself. But the problem is crucial if we really want to use the mechanism and protect agents' personal values at the same time. Regarding time complexity, we have found out a relation between the expressivity of the value function and the running time due to the changed number of distinct MWW graphs. While the graphs generated in phase I can be removed greatly, we have found out the yield of truly competitive solutions from these graphs is again really small. Therefore, there might be more space to improve the algorithm. We perceived that the yield may depend on the value function as well, whose plots are included in the appendix B.3. Further work can link it with the multi-solution and no solution problem. As for incentive compatibility, we did not find any strategy of lying that guarantees agents to decrease their disutility. But we should not exclude its possibility especially when more information about the population is provided. We have also not discussed more about the collusive behaviour.And we would like to see if there is a way to resolve the no-solution problem. So far, all our values and budgets are integers. However, perhaps we can use approximations to secure a result, rather than plead some agent to change their values or even budgets. Just as we mentioned in the last section of the background chapter, money is really virtual and should be seen as a tool to give us an ideal solution. We should not hesitate too much to change the initial requirements a little bit.

Beyond all these, problems with unequal budgets have not been tackled. And since all our experiments had an implicit assumption that population is homogeneous, i.e. their values were generated uniformly random, it is also worthwhile to explore the outcomes when population have more traits. For example, the population may be quite grumpy, tending to give high disutility values to chores or they are generally indifferent about which chores to do. Using values under different Gaussian distributions may be a good start point.

Jumping outside of our box, we encourage more implementations of fair allocation algorithms to be accomplished and evaluation centring around more concrete questions to be carried out. The most immediate extension can be the similar algorithms constructed for competitive allocation of the mixed [27]. Additionally, it will be interesting to see this algorithm modified for allocating indivisible chores (see section 7 of [10]). Also, competitive allocation of chores under other utility functions such as the sometimes more realistic *separable, piecewise-linear concave* (SPLC) function is starting to receive solutions, like the simplex-like algorithm proposed in [13].

Bibliography

[1] Kenneth J. Arrow and Gerard Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22, July 1954.

[2] Haris Aziz, Ioannis Caragiannis, and Ayumi Igarashi. Fair allocation of combinations of indivisible goods and chores. 2018.

[3] Xiaohui Bei, Xinhang Lu, Pasin Manurangsi, and Warut Suksompong. The price of fairness for indivisible goods. 2019.

[4] Nawal Benabbou, Mithun Chakraborty, Xuan-Vinh Ho, Jakub Sliwinski, and Yair Zick. The price of quota-based diversity in assignment problems. *ACM Transactions on Economics and Computation*, 8, 09 2020.

[5] Anna Bogomolnaia, Hervé Moulin, Fedor Sandomirskiy, and Elena Yanovskaia. Dividing bads under additive utilities. *Social Choice and Welfare*, 52, 03 2019.

[6] Anna Bogomolnaia, Hervé Moulin, Fedor Sandomirskiy, and Elena Yanovskaya. Dividing goods and bads under additive utilities. 2016.

[7] Anna Bogomolnaia, Hervé Moulin, Fedor Sandomirskiy, and Elena Yanovskaya. Dividing goods or bads under additive utilities. 2016.

[8] Anna Bogomolnaia, Hervé Moulin, Fedor Sandomirskiy, and Elena Yanovskaya. Competitive division of a mixed manna. In *Econometrica*, volume 85, November 2017.

[9] William C. Brainard and Herbert E. Scarf. How to Compute Equilibrium Prices in 1891. *American Journal of Economics and Sociology*, 64(1):57–83, January 2005.

[10] Simina Branzei and Fedor Sandomirskiy. Algorithms for competitive division of chores. 2019.

[11] Eric Budish. The combinatorial assignment problem:approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119, 2011.

[12] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. *ACM Transactions on Economics and Computation*, 7, 2016.

[13] Bhaskar Ray Chaudhury, Jugal Garg, Peter McGlaughlin, and Ruta Mehta. Competitive allocation of a mixed manna. 2020.

[14] Bhaskar Ray Chaudhury, Jugal Garg, Peter McGlaughlin, and Ruta Mehta. Dividing bads is harder than dividing goods on the complexity and of fair and efficient division and of chores. 2020.

[15] Ning Chen, Xiaotie Deng, Xiaoming Sun, and Andrew Chi-Chih Yao. Fisher equilibrium price with a class of concave utility functions. In *Algorithms - ESA 2004, 12th Annual European Symposium Proceedings*, pages 169–179.

[16] Xi Chen and Shang-Hua Teng. Spending is not easier than trading: On the computational equivalence of fisher and arrow-debreu equilibria. In *Algorithms and Computation*, pages 647–656, 2009.

[17] John S Chipman. Homothetic preferences and aggregation. *Journal of Economic Theory*, 8(1):26–38, 1974.

[18] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[19] Nikhil R. Devanur, Jugal Garg, Ruta Mehta, Vijay V. Vazirani, and Sadra Yazdanbod. A new class of combinatorial markets with covering constraints: Algorithms and applications. *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms*.

[20] Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, page 389–395. IEEE Computer Society, 2002.

[21] Huw Dixon. *Equilibrium and Explanation*. Blackwells, 1990.

[22] Ronald Dworkin. What is equality? part 2: Equality of resources. *Philosophy Public Affairs*, 10(4):283–345, 1981.

[23] Edmund Eisenberg and David Gale. Consensus of Subjective Probabilities: The Pari-Mutuel Method. *The Annals of Mathematical Statistics*, 30(1):165 – 168, 1959.

[24] Kousha Etessami and Mihalis Yannakakis. On the complexity of nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

[25] Duncan K Foley. Resource allocation and the public sector. *Yale Economic Essays*, (7):45–98, 1967.

[26] Martin Gardner. *Aha! Aha! insight*. Scientific American, 1978.

[27] Jugal Garg and Peter McGlaughlin. Computing competitive equilibria with mixed manna. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2020*, pages 420–428.

[28] Rahul Garg, Sanjiv Kapoor, and Vijay Vazirani. An auction-based market equilibrium algorithm for the separable gross substitutability case. pages 128–138, 01 2004.

[29] Louis Gevers. *Walrasian social choice: some simple axiomatic approaches*, volume 1, page 97–114. Cambridge University Press, 1986.

[30] Jonathan R. Goldman and Ariel D. Procaccia. Spliddit: unleashing fair division algorithms. *SIGecom Exch.*, 13:41–46, 2015.

[31] Leonid Hurwicz. On informationally decentralized systems. *Decision and organization: A volume in Honor of J. Marschak*, 1972.

[32] Leonid Hurwicz. On allocations attainable through nash equilibria. *Journal of Economic Theory*, 21(1):140–165, 1979.

[33] Ryo ichi Nagahisa. A local independence condition for characterization of walrasian allocations rule. *Journal of Economic Theory*, 54:106–123, 1991.

[34] David Kurokawa and Ariel Procaccia. Leximin allocations in the real world. pages 345–362, 06 2015.

[35] Richard Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM conference on Electronic commerce*, 2004.

[36] Peter McGlaughlin and Jugal Garg. Improving nash social welfare approximations. *Journal of Artificial Intelligence Research*, 68:225–245, may 2020.

[37] Hervé Moulin. *Social welfare orderings*, page 30–60. Econometric Society Monographs. Cambridge University Press, 1988.

[38] Hervé Moulin. Fair division in the internet age. *Annual Review of Economics*, 11(1):407–441, 2019.

[39] Hervé Moulin and William Thomson. Can everyone benefit from growth?: Two difficulties. *Journal of Mathematical Economics*, 17(4):339–345, 1988.

[40] James B. Orlin. Improved algorithms for computing fisher's market clearing prices. *Proceedings of the 42nd ACM symposium on Theory of computing*, 2010.

[41] Elisha A. Pazner and David Schmeidler. Egalitarian Equivalent Allocations: A New Concept of Economic Equity. *The Quarterly Journal of Economics*, 92(4):671–687, 11 1978.

[42] Sara Ramezani and Ulle Endriss. Nash social welfare in multi-agent resource allocation. In *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 117–131. Springer Berlin Heidelberg, 2010.

[43] Aviad Rubinstein. Inapproximability of nash equilibrium. 2014.

[44] Wayne Shafer and Hugo Sonnenschein. Chapter 14 market demand and excess demand functions. volume 2 of *Handbook of Mathematical Economics*, pages 671–693. Elsevier, 1982.

[45] Hugo Steinhaus. The problem of fair division. *Econometrica*, January 1948.

[46] Ankang Sun, Bo Chen, and Xuan Vinh Doan. Connections between fairness criteria and efficiency for allocating indivisible chores. *AAMAS '21: Proceedings of the 20th International Conference on Autonomous Agents and Multi-Agent Systems*.

[47] Martino Traxler. Fair chore division for climate change. *Social Theory and Practice*, 28(1):101–134, 2002.

[48] Hal R Varian. Equity, envy, and efficiency. *Journal of Economic Theory*, 9(1):63–91, 1974.

[49] Léon Walras. Elements d'economie politique pure ou theorie de la richesse sociale. 1874.

[50] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. The effect of false-name bids in combinatorial auctions: new fraud in internet auctions. *Games and Economic Behavior*, 46(1):174–188, 2004.

Appendix A

Some other pseudocode used in implementation

Algorithm 6 Generating another rich family using duality

```
procedure DUALGRAPHGENERATOR(agents)
    $newvalues \leftarrow []$
    $newAgents \leftarrow []$
    for $agent$ in $agents$ do
        $newvalues$.append($agent.values$)
    end for
    for $chore$ in $chores$ do
        $values \leftarrow [row[chore]$ for $row$ in $newvalues]$
        $newAgents$.append($Agent(chorej, values, 0)$)
                                    ▷ We do not need to use budgets here, so they can be 0.
    end for
    $graphs \leftarrow GRAPHGENERATOR(newAgents)$
    return $graphs$
end procedure
```

Algorithm 7 Helper function 1 for MaxFlow

```
1: procedure BFS(C, F, s, t)                              ▷ C is the capacity matrix
2:     $n \leftarrow$ len(C)
3:     $queue \leftarrow []$
4:     $queue$.append($s$)
5:     $level \leftarrow n * [0]$                                        ▷ initialisation
6:     $level[s] \leftarrow 1$
7:     while $queue$ do
8:         $k \leftarrow queue.pop(0)$
9:         for $i$ in range(n) do
10:            if $F[k][i] < C[k][i]$ and $level[i] == 0$ then
11:                $level[i] \leftarrow level[k] + 1$
12:                $queue$.append(i)
13:            end if
14:        end for
15:    end while
16:    return $level[t] > 0$
17: end procedure
```

Algorithm 8 Helper function 2 for MaxFlow

```
1: procedure DFS(C, F, k, cp)
2:     $tmp \leftarrow cp$
3:     if k == len(C)-1 then
4:         return $cp$
5:     end if
6:     for $i$ in range(len(C)) do
7:         if $level[i] == level[k] + 1$ and $F[k][i] < C[k][i]$ then
8:             $f \leftarrow Dfs(C, F, i, min(tmp, C[k][i] - F[k][i]))$
9:             $F[k][i] \leftarrow F[k][i] + f$
10:            $F[i][k] \leftarrow F[i][k] - f$
11:            $tmp \leftarrow tmp - f$
12:        end if
13:    end for
14:    return $cp - tmp$
15: end procedure
```

Algorithm 9 maxflow

```
1: procedure MaxFlow(C,s,t)
2:     $n \leftarrow len(C)$
3:     $F \leftarrow [n * [0]$for i in range(n)]                ▷ F is the flow matrix
4:     $flow \leftarrow 0$
5:     while Bfs(C,F,s,t) do
6:         $flow \leftarrow flow + Dfs(C, F, s, infinity)$
7:     end while
8:     return $flow, F$
9: end procedure
```

Algorithm 10 Checking competitiveness

```
1: procedure ifCompetitive(profile, graph, agents)
2:     $q \leftarrow \{\}$                ▷ should equal the price vector
3:     for $chore$ in chores do
4:         $q$.update($\{chore$:min([$i.budget.disutilities[chore]/profile[i.name]$ for $i$ in $agents$])$\}$)
5:     end for
6:     $condition1 \leftarrow$sum([$i.budget$ for $i$ in $agents$]) == sum($q.values()$)
7:     if $condition1$=True then
8:         $maxflow, network \leftarrow MAXFLOW(GRAPHTONET(graph, q), 0, m+n+1)$
9:         if $sum(q.values()) == maxflow$ then
10:            $bundle \leftarrow$[[] for $i$ in range(n)]
11:            for $i$ in range(n) do:
12:                for $j$ in $chores$ do:
13:                    $bundle[i]$.append ($network[i+1][1+n+j]/q[j]$)
14:                end for
15:            end for
16:        end if
17:    end if
18:    return $bundle, q$
19: end procedure
```

Appendix B

Other experiment results

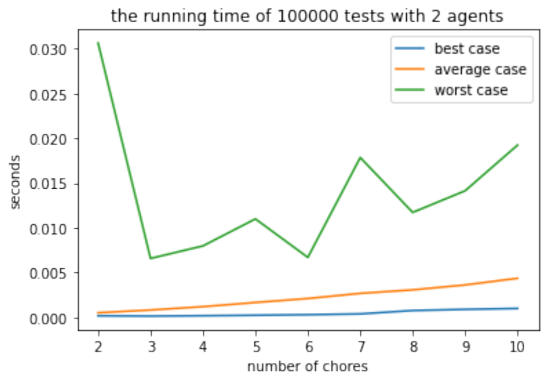

Figure B.1: Worst case are extremely rare.

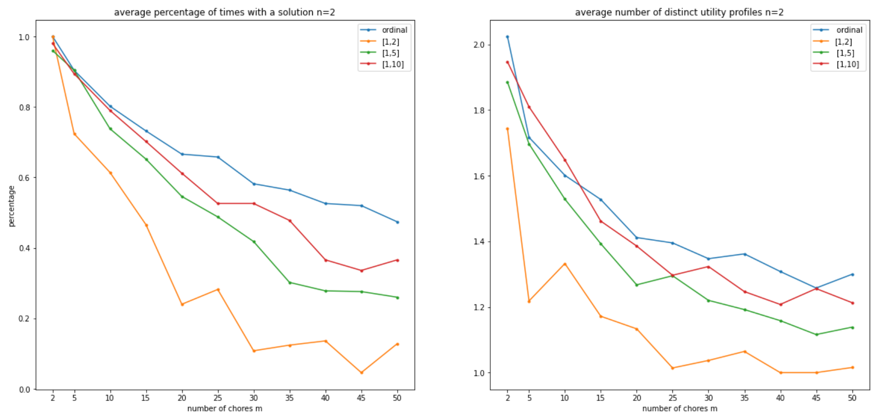

Figure B.2: The likelihood of having a solution is decreasing as m increases. It also depends of the value function.

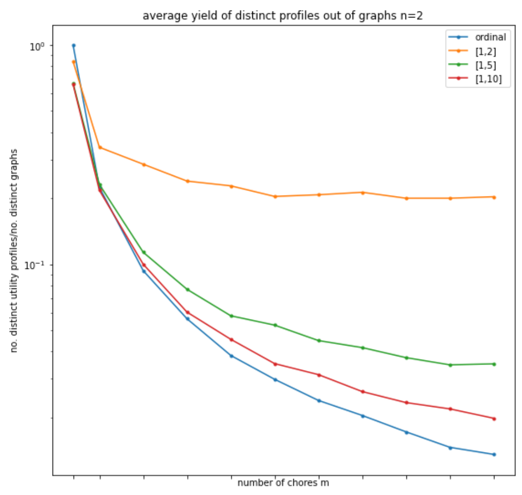

Figure B.3: The yield of distinct utility profiles out of distinct MWW graphs is very low. And the order between different value function is the reverse of how different percentages of times the algorithm gives an solution are ranked.

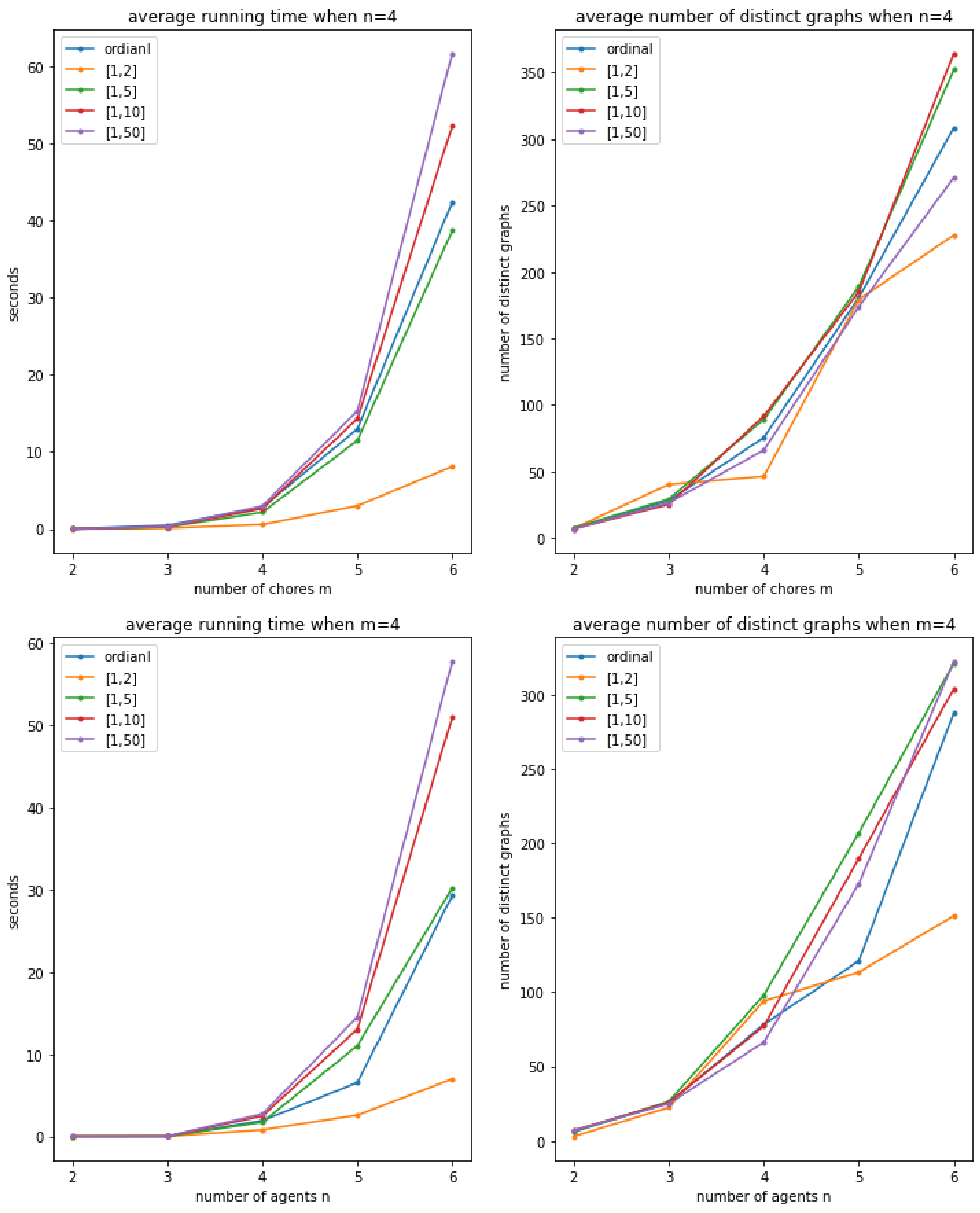

Figure B.4: The order of number of distinct graphs according to different value function is not the same as the order of running time although the bigger range, running time is still longer.

		$m = 3$				$n = 3$			
n/m	$value function$	mu'	mu	z	s	mu'	mu	z	s
2		48	48	3	49	0	0	0	100
3		55	67	0	33	59	72	0	28
4		89	90	1	9	51	54	17	29
5	ordinal	75	89	0	11	54	64	12	24
10		80	96	0	4	35	48	16	36
15		62	89	1	10	22	45	25	30
20		51	90	3	7	30	43	30	27
2		37	42	7	51	36	42	9	49
3		61	75	6	19	47	66	12	22
4	[1,5]	63	89	2	9	67	89	3	8
5		78	92	0	8	65	87	5	8
10		57	97	0	3	61	95	1	4
15		74	97	1	2	64	93	4	3
20		74	95	0	5	64	92	2	6
2		42	53	7	40	34	46	10	44
3		48	71	7	22	53	76	3	21
4		63	80	5	15	59	85	1	14
5	[1,10]	58	86	2	12	58	86	7	7
10		58	95	2	3	54	95	1	4
15		59	94	5	1	64	94	4	2
20		65	88	8	4	60	87	8	5
2		37	65	7	28	42	56	7	37
3		39	73	10	17	47	75	7	18
4		49	79	8	13	48	81	4	15
5	[1,100]	37	68	9	23	36	72	15	13
10		34	57	17	26	50	86	11	3
15		20	46	21	33	49	87	8	5
20		13	39	34	27	55	83	12	5

Table B.1: The original data used to plot the bar charts in section 5.4. Experiments were done in two hundred trial for each pair of (m,n) and a different value function. There are many small patterns in the table worth exploring.

- mu' — percentage of solutions have a singled out answer selected from multiple solutions achieving minimal total sum disutility, minimal disutility of the most suffering agent, minimal nash product;
- mu — percentage of solutions that are multi-valued;
- z— percentage of solutions that have zero solution;
- s—percentage of solution that have single solution.

Appendix C

Allocation examples used for discussion

Agent	Values	Budget	Allocation	Disutility
Bill	[1,1]	4	[1, $\frac{1}{7}$]	$\frac{8}{7}$
Jack	[2,2]	1	[0, $\frac{2}{7}$]	$\frac{4}{7}$
Amy	[3,3]	1	[0, $\frac{2}{7}$]	$\frac{6}{7}$
Lucy	[4,4]	1	[0, $\frac{2}{7}$]	$\frac{8}{7}$

Table C.1: Another illustration of how the budget is treated differently from the values. Bill's disutility (8/7) is four thirds times of Amy's (6/7), in alignment with our equation.

Agent	Values	Budget	Allocation1	Allocation2	Allocation3
Bill	[2,2]	2	[0,0.25] (0.5)	[0,4/15] (0.54)	[0.25,0] (0.5)
Jack	[1,3]	2	[0,0.25] (0,75)	[0.2,0.2] (0.8)	[0.25,0] (0.25)
Amy	[5,1]	2	[0,0.25] (0.25)	[0,4/15] (0.27)	[0,1] (1)
Lucy	[2,4]	2	[0,0.25] (1)	[0,4/15] (1.07)	[0.25,0] (0.5)
Sarah	[1,5]	2	[1,0] (1)	[0.8,0] (0.8)	[0.25,0] (0.25)
			final price: [2,8]	[2.5,7.5]	[8,2]

Table C.2: The problem of being multi-valued. The resulted three allocations are very different from each other.

Agent	Values	Allocation	Disutility
Bill	[2,2]	[0, 0.3]	0.6
Jack	[1,3]	[0.6, 0]	0.6
Amy	[3,2]	[0, 0.3]	0.6
Lucy	[2,4]	[0.4, 0.1]	1.2
Sarah	[1,2]	[0, 0.3]	0.6
			price: [3.3,6.7]

Agent	Values	Allocation	Disutility
Bill	[2, 2]	[0, 4/15]	0.53
Jack	[1, 3]	[0.8, 0]	0.8
Amy	[3, 2]	[0, 4/15]	0.53
Lucy	[2, 4]	[0, 4/15]	1.07
Sarah	[1,3]	[0.2, 0.2]	0.5
			price: [2.5,7.5]

Agent	Values	Allocation1	Disutility1	Allocation2	Disutility2
Bill	[2,2]	[0, 0.25]	0.5	[0, 0.27]	0.53
Jack	[1,3]	[0, 0.25]	0.75	[0.2, 0.2]	0.8
Amy	[3,2]	[0, 0.25]	0.5	[0, 0.27]	0.53
Lucy	[2,4]	[0, 0.25]	1	[0, 0.27]	1.07
Sarah	[1,4]	[1,0]	1	[0.8, 0]	0.8
			price: [2,8]		[2.5,7.5]

Agent	Values	Allocation1	Allocation2	Allocation3
Bill	[2,2]	[0, 0.4]	[0, 1/3]	[0,0.3]
Jack	[1,3]	[0.4, 0]	[0.5, 0]	[0.6, 0]
Amy	[3,2]	[0, 0.4]	[0, 1/3]	[0, 0.3]
Lucy	[2,4]	[0.4, 0]	[0.5, 0]	[0.4, 0.1]
Sarah	[1,1]	[0.2, 0.2](0.5)	[0, 1/3](2/3)	[0, 0.3](0.6)
		price:[5,5]	[4,6]	[3.3, 6.7]

Table C.3: The outcomes after Sarah lying about her value of the second chore. Her real values are [1,2]. If she reports them as [1,3], she will be strictly better off. If she reports [1,1] instead, she may face three different results with her disutility being increased, reduced or unaffected. If she reports [1,4], she will be strictly worse off.