Computationally Efficient Representation Languages for Fairly Dividing Indivisible Goods

MSc Thesis *(Afstudeerscriptie)*

written by

Boas Kluiving

under the supervision of **Dr. Ronald de Haan**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
September 17th, 2020	Dr. Ekaterina Shutova *(chair)*
	Prof. dr. Ulle Endriss
	Prof. dr. Ronald de Wolf
	Dr. Ronald de Haan

Institute for Logic, Language and Computation

Abstract

We consider the scenario of fairly dividing a set of indivisible items among a group of agents with cardinal preferences. Because agents may have different preferences for every combination of items, intermediate representation languages are required in order to compactly represent the agents' preferences. To choose the most fair allocation, we consider various solution concepts, such as utilitarian, egalitarian or Nash welfare, and look for the allocation elected by a particular solution concept.

In order to do this efficiently however, we need to be able to calculate such an elected allocation in polynomial time. In this thesis, we study the computational complexity of finding these allocations for various representation languages. Our aim here is to discover representation languages that have polynomial-time algorithms for finding such allocations, while at the same time being as expressive as possible. We also study intractability results for restricted versions of representation languages in order to discover which properties make the problem computationally intractable.

In particular, the languages that we study include the dichotomous language based on propositional formulas, the additive language and languages based on weighted bids. Based on the computational properties of these languages, we present a new representation language, called the disjunctive partition language, which has interesting tractability properties. Finally, we also study the different representation languages from a parameterized complexity perspective.

Acknowledgements

I would like to start by thanking my thesis supervisor, Ronald de Haan. When I first approached Ronald about the possibilities for doing a thesis in December, he was immediately very open to it and helped me a lot with finding a good topic for my thesis. During meetings, he was always enthusiastic and interested in my ideas and he provided me with valuable feedback and suggestions. Although we had to move our meetings online, he remained very approachable and we found a good way to continue our meetings. I would like to thank Ronald in particular for all the helpful comments and feedback he gave me on the various revisions I sent him of my thesis.

Secondly, I wanted to thank both Simon Rey and Ulle Endriss for taking the time to discuss my preliminary results with them. Discussing my progress with both of them was very helpful to put my preliminary results more in perspective and their comments and suggestions gave me new ideas for what I could still work on.

I would also like to thank the thesis committee for taking the time to read and review my thesis.

Finally, I would like to thank Ulle Endriss for getting me interested in the field of computational social choice in the first place. I found his course on the subject very interesting, which led me to carry out a research project under his supervision, together with Adriaan and Pepijn. Ulle warmly welcomed us to the field of computational social choice and guided us in a very nice way with doing research.

Contents

Chapter 1

Introduction

Consider the scenario of an inheritance of a family member: there is a collection of items left over that need to be divided among the relatives. However, each relative might value the items differently. For example, someone might have a personal connection to a specific item such as a painting, making that item worth a lot to him. It could also be the case that a person does not value the item as much as the others, because he already owns a similar item. For instance, suppose one of the items we need to divide is a car. A relative who already owns a better car himself might not value this item the same as someone who does not have a car and could use one.

In addition to this, the value of an item might depend on which other items someone receives. If a single relative receives a complete dinnerware set, this might be worth more to him than if we were to split up the dinnerware set over multiple people. On the other hand, if two items are very similar, having just one of these items might be enough for someone; also having the second one does not add much value anymore. If we have two cars that we need to divide, it might not make sense to give them both to one person, since one car is already sufficient for him.

With all these preferences of the relatives, we would like to find an allocation of the items such that everyone receives the items that they value most, while at the same time ensuring that everyone receives a fair share. An important assumption that we make here is that the items themselves are indivisible: we cannot split a painting or a car in parts to give it to multiple people. This scenario of fairly dividing a set of indivisible goods is what we focus on in this thesis (Bouveret, Chevaleyre, et al., 2016).

A couple of challenges arise when studying such a fair division problem. The first one is that of preference representation. A natural way to do so would be to ask all the agents participating in the fair division problem to specify a utility value for every combination of items they might receive. Note that we cannot simply ask them to specify a utility value for every item separately and add these together. If we do this, we lose the ability to express relations between items, such as in the case of the dinnerware set or with multiple cars.

But if an agent needs to specify a utility value explicitly for all subsets of the set of items, the number of utility values he needs to specify becomes exponential in the number of items. To solve this problem, we need an intermediate representation language in which an agent can express his preferences compactly. This representation language of course cannot represent all possible utility functions, but the idea is that most interesting functions – those that are likely to correspond to real-world preferences of agents – can be expressed in such a language. This raises the question of which representation language we should choose for a fair division problem.

Once we have found a way to express the agents' preferences compactly, a second problem that arises is how to decide which allocation to choose. On the one hand, we would like to assign the objects to the agents wanting those objects the most, but on the other hand we would also like everyone to get a fair share of the items. For example, we could try to find an allocation such that the sum of the agents' utilities are maximized. However, this does not provide any fairness guarantees: it might happen that one agent ends up receiving all the items if he values everything the most. A different idea is to find an allocation that maximizes the utility of the agent that is worst-off to make everyone obtain a share that is as equal as possible. We call such measures that quantify how good an allocation is solution concepts. There are also other solution concepts which try to find a balance between these two criteria or do so in a different way. Which solution concept to use also depends a lot on the specific application of the fair division problem.

Besides choosing a representation language and a solution concept, a third challenge is that of computational complexity. Namely, for many intuitive representation languages it is intractable to find the allocation elected by some solution concept. As an example, even when all the utilities are additive – i.e., the sum of an agent's bundle is always the sum of the individual utilities for the objects, meaning there are no relations between objects – and there are only two agents, calculating an allocation that maximizes the utility for the worst-off agent is already NP-hard (Lipton et al., 2004). Therefore, it is interesting to look for representation languages that are such that we can calculate the allocations elected by different solution concepts in polynomial time.

Identifying such representation languages will be the main goal of this thesis. We will study various representation languages and determine the computational complexity of finding the allocations elected by different solution concepts for these representation languages. Our aim will be to find representation languages that are on the one hand still expressive enough to represent non-trivial preferences and on the other hand have polynomial-time algorithms to find the allocations elected by the solution concepts. In order to find these tractable representation languages, we will also prove intractability results for various languages. These results allow us to understand under which restrictions these languages remain intractable and which restrictions might help us to find a tractable sub-language.

Note that in this thesis we are specifically considering algorithms that solve these optimization problems exactly. A different approach to obtain tractability results is to look for algorithms that calculate approximately optimal solutions (Markakis, 2017) or to find weaker versions of the solution concepts that allow us

to elect an allocation in polynomial time (Amanatidis et al., 2018; Kyropoulou et al., 2019; Lipton et al., 2004). The approach we will take is to look at restricting the representation languages such that electing the allocations for various solution concepts becomes tractable, while at the same time trying to keep the representation language as expressive as possible.

1.1 Thesis outline

In the following paragraphs, we will give an overview of the contents of the chapters in this thesis.

In Chapter 2, we introduce the model of fair division of indivisible goods formally. We furthermore define various solution concepts that we use later throughout the thesis, such as utilitarian social welfare, egalitarian welfare, Nash welfare, etc., and we define the computational problems related to them. We also provide a more thorough definition of what a representation language entails. Lastly, we discuss the other research related to this thesis.

Subsequently, in Chapters 3 to 7, we study the computational properties of various representation languages in more depth.

We start in Chapter 3 with the dichotomous language, which is a representation language where every agent specifies a propositional formula as his preference. If this formula is true according to the bundle the agent receives, he receives utility 1 and receives utility 0 otherwise. We show that this representation language is NP-complete for all solution concepts we consider, even in a very restrictive setting. Restricting the propositional formula to a single clause does yield tractable algorithms.

In Chapter 4, we study the additive language in more detail. After finding only a (trivial) tractable result for maximizing utilitarian welfare, we consider two restrictions: requiring the preferences to be encoded in unary and only allowing binary-valued preferences. For the first restriction, the problem remains NP-hard, which we also prove, but under the second restriction it becomes possible to optimize for Nash and egalitarian welfare in polynomial time.

In Chapter 5, we look at conjunctive and disjunctive weighted bids. Since these models are a generalization of additive preferences, they are mostly interesting to study in relation to utilitarian welfare. It turns out there is a close link with combinatorial auctions, which we discuss in this chapter. Although the general problem is NP-complete for utilitarian welfare, we find a tractable algorithm for conjunctive weighted bids in the case of two agents. We show that the problem becomes intractable already when there are three (or more) agents. The language of disjunctive weighted bids remains intractable in a very restrictive setting, because of the overlap that is allowed between the bids.

Motivated by the intractability results for disjunctive weighted bids, we introduce in Chapter 6 a variant of disjunctive weighted bids with no overlap between bids called the disjunctive partition language and show that maximizing utilitarian welfare for this language is possible in polynomial time. In the rest of the chapter, we discuss various extensions of this representation language that remain tractable. We discuss the relation with combinatorial auctions and

give intractability results when attempting to apply the same restriction to conjunctive weighted bids.

In Chapter 7, we continue on the variant of disjunctive weighted bids, but now add the extra restriction that the preferences are binary-valued. We provide an algorithm that finds a leximin-optimal allocation (a generalization of egalitarian welfare) in polynomial time and prove its correctness. Furthermore, we show that when using this representation language, the optimal Nash welfare and leximin-optimal allocation coincide. Finally, we look at a few extensions of this representation language and show that these extensions all introduce intractability.

Finally, in Chapter 8, we study the various representation languages from a more detailed perspective: using parameterized complexity. We show various fixed-parameter tractability and intractability results, both for representation languages in general and for the additive and conjunctive weighted bids language in particular. We conclude and provide suggestions for further research in Chapter 9.

Chapter 2

Preliminaries

In this chapter, we introduce the necessary background and definitions that are needed for this thesis. We will follow the notation and definitions from Bouveret, Chevaleyre, et al. (2016) as much as possible.

We start by formally defining the model in Section 2.1. Subsequently, we introduce the various solution concepts in Section 2.2 of which we will study the computational complexity in later chapters and in Section 2.3 we define their corresponding computational problems. We also define the notion of a representation language more formally and illustrate this with an example in Section 2.4. Finally, we discuss related work to this thesis in Section 2.5.

2.1 The model

In the scenario of fairly allocating indivisible goods, we have a set N of agents and a (discrete) set $\mathcal{O}$ of goods that we want to divide over the agents. Every good can only be assigned to at most one agent. Furthermore, every agent specifies for each possible combination of items a positive integer utility value that indicates how much receiving this combination of items is worth to him. We can formally define this in the following way.

Definition 2.1 (Fair division problem)**.** A *fair division problem* consists of a tuple $(N, \mathcal{O}, \boldsymbol{u})$, where

- $N = \{1, \ldots, n\}$ is the set of n agents,
- $\mathcal{O} = \{o_1, \ldots, o_m\}$ is the set of m objects, and
- $\boldsymbol{u} = (u_1, \ldots, u_n)$, where each $u_i : 2^{\mathcal{O}} \to \mathbb{N}$ is the utility function for agent i.

The reason we need agents to specify a utility value for every possible subset of the items $\mathcal{O}$ is that the utility of items need not be additive. For example, an agent i could specify that $u_i(\{o_1\}) = 2$ and $u_i(\{o_2\}) = 2$, but $u_i(\{o_1, o_2\}) = 5 \neq u_i(\{o_1\}) + u_i(\{o_2\})$.

In our notation, every utility value is required to be a natural number. If we wanted to allow (positive) rational numbers as well, we can transform such a fair division problem into an equivalent one where every utility value is a natural number by multiplying all the utility values by the lowest common denominator.

Now that we have defined what a fair division problem is, we can also specify what we mean exactly by an allocation of items. An allocation simply assigns to every agent a set of items such that each item is only assigned to at most one agent.

Definition 2.2 (Allocation)**.** An *allocation* is a function $\pi : N \to 2^{\mathcal{O}}$ such that $i \neq j$ implies that $\pi(i) \cap \pi(j) = \emptyset$.

We call each subset $S \subseteq \mathcal{O}$ a *bundle*. The subset of items $\pi(i)$ thus indicates agent i's bundle according to allocation π. The utility function u_i indicates for each bundle S how much utility agent i obtains when receiving bundle S. Hence, the utility of an agent i given an allocation π is equal to $u_i(\pi(i))$.

Instead of a function from agents to sets of objects, we can also view an allocation as a (partial) function $\mathcal{O} \to N$ from objects to agents where every object is assigned to (at most) one agent. Sometimes this way of looking at allocations makes the mathematical notation easier. If $\pi : N \to 2^{\mathcal{O}}$ is an allocation from agents to sets of objects, we will refer to $\pi^{-1} : \mathcal{O} \to N \cup \{\epsilon\}$ as the same allocation π, but described as a function from objects to agents (where an item is mapped to ϵ if it is not assigned to any agent).

Example 2.3. Consider a fair division problem $(N, \mathcal{O}, \boldsymbol{u})$ with three agents $N = \{1, 2, 3\}$ and two items $\mathcal{O} = \{o_1, o_2\}$. The utility functions for the agents are as follows:

$u_1(\emptyset) = 0$	$u_2(\emptyset) = 0$	$u_3(\emptyset) = 1$
$u_1(\{o_1\}) = 2$	$u_2(\{o_1\}) = 3$	$u_3(\{o_1\}) = 2$
$u_1(\{o_2\}) = 3$	$u_2(\{o_2\}) = 5$	$u_3(\{o_2\}) = 2$
$u_1(\{o_1, o_2\}) = 10$	$u_2(\{o_1, o_2\}) = 5$	$u_3(\{o_1, o_2\}) = 4$

A possible allocation π_1 would be to allocate all items to agent 1. In this situation, the utilities of the agents are respectively $(10, 0, 1)$.

$$\pi_1(1) = \{o_1, o_2\}, \pi_1(2) = \emptyset \text{ and } \pi_1(3) = \emptyset$$

A different allocation π_2 is to allocate o_1 to agent 1 and o_2 to agent 2. This yields the utilities $(2, 5, 1)$.

$$\pi_2(1) = \{o_1\}, \pi_2(2) = \{o_2\} \text{ and } \pi_2(3) = \emptyset$$

Finally, consider the allocation π_3 in which we assign o_1 to agent 2 and o_2 to agent 1. Then the utilities of the agents will be $(3, 3, 1)$.

$$\pi_3(1) = \{o_2\}, \pi_3(2) = \{o_1\} \text{ and } \pi_3(3) = \emptyset$$

△

2.2 Solution concepts

Now that we have defined fair division problems and allocations, we also want to quantify in some way how 'good' an allocation is. For example, we might prefer an allocation in which the agents in total receive more utility over an allocation where the total utility is lower. On the other hand, we might also want to take the fairness of an allocation into account: does a single agent receive all the utility or is the utility divided over the different agents?

We call these measures that quantify how good an allocation is *solution concepts*. One natural idea to compare allocations is to consider the utilitarian social welfare; the sum of utilities of all agents. This is defined in the following way:

Definition 2.4 (Utilitarian social welfare)**.** The *utilitarian social welfare* (USW) of a given allocation $\pi : N \to 2^{\mathcal{O}}$ is defined as

$$\text{USW}(\pi) = \sum_{i \in N} u_i(\pi(i)).$$

We sometimes also refer to this solution concept as just *utilitarian welfare* (leaving out the 'social' part).

We can, given a fair allocation problem $(N, \mathcal{O}, \boldsymbol{u})$, look for the allocation that maximizes the utilitarian social welfare. However, while social welfare maximizes the sum of utilities of all agents, this does not mean that every agent will get an equal share. It might even happen that all items are assigned to a single agent.

A different notion that attempts to capture the fairness of a solution is egalitarian social welfare. In this solution concept, we do not look at the sum of all utilities, but rather at the utility of the agent that is worst off. We then try to find an allocation that maximizes this utility.

Definition 2.5 (Egalitarian social welfare)**.** The *egalitarian social welfare* (ESW) of a certain allocation $\pi : N \to 2^{\mathcal{O}}$ is defined as

$$\text{ESW}(\pi) = \min_{i \in N} u_i(\pi(i)).$$

One of the disadvantages of this solution concept is that it only measures the utility of the worst-off agent. Any two allocations where the worst-off agent has the same utility will have the same egalitarian welfare, regardless of the utility that the other agents receive. A refinement of this solution concept is the leximin comparison.

Definition 2.6 (Leximin comparison)**.** Given an allocation π, we define the utility vector of this allocation $\boldsymbol{u}_\pi$ to be equal to $(u_1(\pi(1)), \ldots, u_n(\pi(n)))$. We furthermore define $\boldsymbol{u}^*_\pi$ as the utility vector $\boldsymbol{u}_\pi$ sorted in ascending order.
We say that an allocation π is *leximin-better* than π', or $\pi \succ_{\text{lex}} \pi'$, if given the two sorted utility vectors $\boldsymbol{u}^*_\pi = (u^*_{\pi,1}, \ldots u^*_{\pi,n})$ and $\boldsymbol{u}^*_{\pi'} = (u^*_{\pi',1}, \ldots u^*_{\pi',n})$, there exists a k such that $u^*_{\pi,k} > u^*_{\pi',k}$ and $u^*_{\pi,j} = u^*_{\pi',j}$ for all $j < k$.

Intuitively, we compare two different allocations by calculating the utility values for every agent for both allocations, sorting them and comparing these sorted vectors lexicographically. An allocation is then leximin-optimal if it is, among all other possible allocations, one of the leximin-best.

Definition 2.7 (Leximin-optimal allocation)**.** A *leximin-optimal allocation* is an allocation π such that there is no other allocation π' with $\pi' \succ_{\text{lex}} \pi$.

A different way to think about a leximin-optimal allocation is that it first maximizes the utility of the worst-off agent. Then, subject to that, it maximizes the utility of the second worst-off agent and again subject to that the utility of the third worst-off agent and so on. Because of this, note that a leximin-optimal allocation also maximizes the egalitarian welfare.

Although egalitarian welfare and leximin-optimal allocations both consider the fairness of an allocation, they do optimize for this fairness at all costs. As an example, if one allocation yields the utility vector $(10, 11)$, while the other allocation yields $(9, 20)$, then according to egalitarian welfare and leximin $(10, 11)$ is better than $(9, 20)$. These two solution concepts – unlike utilitarian welfare – do not take into account that the sum of utilities in $(9, 20)$ is a lot higher than $(10, 11)$. We would like to have a solution concept that takes both fairness into account, but at the same time also prefers allocations with higher total utility.

Nash welfare is a solution concept that attempts to combine both of these criteria. With Nash welfare, instead of considering the sum of utilities, we look at the product of the utilities. We then search for the allocation that maximizes this product of utilities.

Definition 2.8 (Nash social welfare)**.** The *Nash social welfare* (NSW) of a given allocation $\pi : N \to 2^{\mathcal{O}}$ is defined as

$$\text{NSW}(\pi) = \prod_{i \in N} u_i(\pi(i)).$$

By considering the product of utilities, we favour redistributions of utility that are inequality-reducing (e.g. $2 \cdot 4 < 3 \cdot 3$), but at the same time we also favour overall increases in utility (e.g. $10 \cdot 11 < 9 \cdot 20$).

Lastly, a slightly different way to achieve fairness in allocations is by requiring envy-freeness. An allocation is envy-free if every agent likes his bundle at least as much as any of the other agents' bundles.

Definition 2.9 (Envy-freeness)**.** We call an allocation π *envy-free* if for every two agents $i, j \in N$ it holds that $u_i(\pi(i)) \geq u_i(\pi(j))$.

We can always trivially construct an allocation that is envy-free by just assigning no items to every agent. But of course, such an allocation is not very interesting to study. Therefore, we add an extra constraint called Pareto efficiency, which ensures that there is no other allocation that is strictly better for some agent and no worse for all the other agents.

Definition 2.10 (Pareto efficiency)**.** An allocation π is *Pareto efficient* if there exists no other allocation π' such that $u_i(\pi'(i)) \geq u_i(\pi(i))$ for all agents $i \in N$ and $u_j(\pi'(j)) > u_j(\pi(j))$ for some agent $j \in N$.

An interesting solution concept is then to find an allocation π that is both envy-free and Pareto efficient. Note that such an allocation does not always exist, as we will see in the example below.

Now that we have defined a number of different solution concepts, we will give an example showing the allocations elected by the different solution concepts and how they compare to each other.

Example 2.11. Consider the fair division problem from Example 2.3. If we wish to maximize the utilitarian welfare, this is obtained by assigning all items to agent 1 (allocation π_1):

$$\max_{\pi} \mathrm{USW}(\pi) = \mathrm{USW}(\pi_1) = 11$$

For egalitarian welfare, note that we must always give one agent an empty bundle, since we have three agents and only two items. Since agent 3 still receives utility 1 for the empty bundle (and the others 0), any allocation maximizing egalitarian welfare will need to give agent 3 the empty bundle and a single item to agent 1 and 2. Since we cannot do better, the maximum egalitarian welfare will be 1 and allocations π_2 and π_3 both achieve this value.

$$\max_{\pi} \mathrm{ESW}(\pi) = \mathrm{ESW}(\pi_2) = \mathrm{ESW}(\pi_3) = 1$$

A leximin-optimal allocation can be seen as a refinement of the maximum egalitarian welfare allocation and does make a choice between π_2 and π_3. Note that $\pi_3 \succ_{\mathrm{lex}} \pi_2$, since $(1, 3, 3)$ is lexicographically better than $(1, 2, 5)$. Hence, π_3 is a leximin-optimal allocation.

If we consider Nash welfare on the other hand, allocation π_2 is preferred over π_3, since $\mathrm{NSW}(\pi_2) = 1 \cdot 2 \cdot 5 = 10$, while $\mathrm{NSW}(\pi_3) = 1 \cdot 3 \cdot 3 = 9$. So,

$$\max_{\pi} \mathrm{NSW}(\pi) = \mathrm{NSW}(\pi_2) = 10.$$

Lastly, note that in this fair division problem, there exists no allocation that is both envy-free and Pareto efficient. If we decide not to allocate all three items, we can always find a Pareto-better allocation by also allocating that item to an agent. If on the other hand we do allocate all the items, one of the agents receives the empty bundle. He will always envy an agent that does receive (at least one) item. △

2.3 Computational aspects

A major part of this thesis will consist of finding efficient algorithms and intractibility results for computing the allocations elected by the solution concepts we defined in the previous section. For that, we first need to define the computational problems of finding the allocations elected by the solution concepts in more detail.

In this thesis, we will assume the reader is familiar with basic complexity theory and polynomial-time reductions. For an introduction and an overview of this field, we refer to textbooks on this topic (e.g. Arora and Barak, 2009).

Let us start with utilitarian welfare. The scenario we will study is when we are given a certain fair division problem instance and we want to find an allocation elected by the utilitarian welfare, i.e., we want to find an allocation that maximizes the utilitarian welfare. We call this optimization problem MAX USW.

MAX USW
Input: A fair division problem instance $(N, \mathcal{O}, \boldsymbol{u})$.
Output: An allocation π' such that $\mathrm{USW}(\pi') = \max_\pi \mathrm{USW}(\pi)$.

However, when studying the computational complexity of such a problem, it often makes more sense to analyze the decision variant of a computational problem. In the decision variant of this problem, we ask ourselves whether there exists an allocation that has utilitarian welfare of at least k. We call this decision problem USW.

USW
Input: A fair division problem instance $(N, \mathcal{O}, \boldsymbol{u})$ and an integer k.
Question: Does there exist an allocation π' such that $\mathrm{USW}(\pi') \geq k$?

Any algorithm that solves the optimization problem MAX USW can also be used to solve the decision variant USW. Namely, if we can find the allocation with maximum USW, we can simply compute its utilitarian welfare and check whether this is greater than k. Conversely, this means that if we can prove that the decision variant is intractable, then the optimization problem must also be intractable.

We will often use the decision variant when we are proving that finding the allocation elected by a solution concept is intractable. On the other hand, when we prove that a certain problem is efficiently solvable, we will most of the times use the optimization variant to obtain the strongest possible result.

Analogously to USW and MAX USW, we can also define ESW, MAX ESW, NSW and MAX NSW. Because of the similarity, we omit the definitions.

For leximin-optimal allocations, we do include an explicit definition of the computational problems. For the optimization variant, we simply ask for a leximin allocation.

MAX LEX
Input: A fair division problem instance $(N, \mathcal{O}, \boldsymbol{u})$.
Output: A leximin-optimal allocation π for this fair division instance.

For the decision variant, instead of an integer k, we specify a vector of n integers and check whether there exists an allocation that is lexicographically better than this vector.

LEX
Input: A fair division problem instance $(N, \mathcal{O}, \boldsymbol{u})$ and a non-decreasing vector $\boldsymbol{v} \in \mathbb{N}^n$.
Question: Does there exist an allocation π such that $\boldsymbol{u}^*_\pi$ is lexicographically better than $\boldsymbol{v}$?

Proposition 2.12. ESW $\leq_p$ LEX

Proof sketch. Suppose we are given a fair division problem instance $(N, \mathcal{O}, \boldsymbol{u})$ plus an integer k and we want to decide whether there exists an allocation π with $\mathrm{ESW}(\pi) \geq k$. We transform this into a LEX problem by choosing

$$\boldsymbol{v} = (\underbrace{k, k, \ldots, k}_{n \text{ times}})$$

and asking whether there exists an allocation π such that $\boldsymbol{u}^*_\pi$ is lexicographically better than $\boldsymbol{v}$. This is the case if and only if there exists an allocation π with egalitarian welfare of at least k. □

Here, $\leq_p$ denotes polynomial-time (many-one) reducibility. Formally, $A \leq_p B$ for decision problems $A, B \subseteq \{0,1\}^*$ if there exists a polynomial-time computable function f such that $x \in A$ if and only if $f(x) \in B$.

Usually, when showing intractability, we will prove that ESW is intractable. By Proposition 2.12 this then automatically implies that LEX is intractable as well, so we will not prove this separately anymore in later chapters.

Finally, we will also define the computational problem for envy-freeness combined with Pareto efficiency. This solution concept is different from the others in the sense that we are not maximizing a quantity. Since an allocation that is envy-free and Pareto efficient does not always exist, we will study the decision problem of whether, given a fair division problem instance, such an allocation exists.

> EEF EXISTENCE
> *Input:* A fair division problem instance $(N, \mathcal{O}, \boldsymbol{u})$.
> *Question:* Does there exist an allocation π' that is both envy-free and Pareto efficient?

Studying envy-freeness plus Pareto efficiency computationally via this decision variant is also done in other research, such as by Bouveret and Lang (2008) and Bliem et al. (2016). In this thesis, we will mostly focus on obtaining tractability and intractability results for the other solution concepts described in this section. We however also study this solution concept to be able to compare the results better with other research.

2.4 Representation languages

The problem with analyzing the computational complexity of fair division problems directly is that even just specifying a utility function $u_i : 2^{\mathcal{O}} \to \mathbb{N}$ explicitly already requires exponential size. This is because we are specifying an explicit integer value for every of the 2^m subsets of the items.

The solution is to not let agents specify their preferences explicitly, but instead use an intermediate representation language in which the agent can express his preferences. This representation language will then of course in most cases not be able to represent all possible utility functions, but a good representation language allows us to represent agents' preferences as closely as possible, while at the same time keeping the language as compact as possible.

Formally, we can define representation languages in the following way.

Definition 2.13 (Representation language)**.** Fix a set of objects $\mathcal{O}$. A *representation language* is defined by a pair (L, I), where L is the set of expressions the language consists of and I is a function that maps any language expression in L to a utility function $u : 2^{\mathcal{O}} \to \mathbb{N}$ – i.e., the type of I is $L \to (2^{\mathcal{O}} \to \mathbb{N})$.

We give an example of a simple representation language to illustrate the definition.

Example 2.14. Let us consider a representation language in which each agent indicates a subset of the items $\mathcal{O}$ that he likes. For every item of this subset that this agent receives in his bundle, he will receive a utility of 1; for all other items he receives a utility of 0.

We can formally represent this as follows. We define the set of expressions L such that $L = 2^{\mathcal{O}}$, meaning that the expression an agent can represent in the language is a subset of the items. Then, given an expression $S_{\text{expr}} \in L$, we define $u(S_{\text{expr}})$ to be:

$$u(S_{\text{expr}})(S_{\text{bundle}}) = |S_{\text{bundle}} \cap S_{\text{expr}}|$$

By choosing an expression from L, an agent has completely specified his utility function.

For a concrete example, suppose that $\mathcal{O} = \{o_1, o_2, o_3, o_4\}$ and agent i chooses expression $S_{\text{expr}} = \{o_1, o_3\} \in L$, then we have that $u_i = u(S_{\text{expr}})$. Here are a few examples of the value of the utility function u_i for different bundles:

$$\begin{aligned} u_i(\{o_1\}) &= 1 & u_i(\{o_2\}) &= 0 & u_i(\{o_1, o_2\}) &= 1 \\ u_i(\{o_1, o_3\}) &= 2 & u_i(\{o_2, o_4\}) &= 0 & u_i(\{o_1, o_2, o_3, o_4\} &= 2 \end{aligned}$$

We will study this representation language, albeit slightly differently defined, in more detail in Section 4.3. △

When we study specific representation languages in later chapters, we will often only define them intuitively and not define them formally according to Definition 2.13. It can sometimes however still be useful to think about representation languages in this formal way.

The goal in this thesis will be to find representation languages that on the one hand can express intuitive properties about agents' preferences, but that on the other hand also allow us to find allocations elected by solution concepts for a fair division instance efficiently. At the very least, the representation languages that we study should allow us to compute the utility of agents easily when given an allocation. If this is already hard to compute, it will most likely be even more difficult to find allocations elected by solution concepts.

Definition 2.15. A representation language (L, I) has *polynomial-time computable utility functions* if there exists a polynomial-time computable procedure F such that for any bundle $S \subseteq \mathcal{O}$ and expression $l \in L$, $F(S, l) = I(l)(S)$.

Intuitively, a representation language has polynomial-time computable utility functions if there exists an algorithm that can compute the utility of a bundle of an agent, given the agent's preferences in the representation language, in polynomial time.

A first observation is that as long as the representation language has polynomial-time computable utility functions, electing allocations for most solution concepts is in the complexity class NP (i.e., possible in polynomial time by a non-deterministic Turing machine). This provides us a first upper bound on the computational complexity for these solution concepts.

Proposition 2.16. USW, ESW, NSW and LEX for any representation language with polynomial-time computable utility functions are in NP.

Proof sketch. To show membership in NP, we give a polynomial-size certificate and a polynomial-time verifier algorithm. The certificate will be (an encoding of) an allocation. The verifier algorithm will then, given a fair division instance $(N, \mathcal{O}, \boldsymbol{u})$, an integer k and a certificate allocation π, compute the USW of π and check whether $\text{USW}(\pi) \geq k$. For ESW, NSW and LEX, we use the same approach, but now calculate the ESW, NSW or lexicographic vector.

As the representation language has polynomial-time computable utility functions, there exists a polynomial-time computable procedure F to calculate the bundle of an agent. Computing the allocations elected by any of these solution concepts requires n queries to this procedure F. Thus, in total, the verifier algorithm runs in polynomial time.

If there exists an allocation with $\text{USW}(\pi) \geq k$, there will be a certificate (namely this allocation) such that the verifier algorithm outputs true. Otherwise, the verifier algorithm will output false for all possible certificate allocations. □

In later chapters, when we study specific representation languages, we will often prove that these decision problems are NP-complete for a certain representation language. Because we have already shown here that these decision problems are in NP for any representation language with polynomial-time computable utility functions, we will only prove NP-hardness there and omit the NP membership proof.

2.5 Related work

Now that we have defined the model and the other preliminaries we need in this thesis, we will briefly discuss on a high level the other research that has been carried out concerning the computational complexity of the fair division of indivisible goods.

An important aspect in fairly dividing indivisible goods is to avoid an agent envying another agent's bundle. Lipton et al. (2004) studied this problem from an algorithmic perspective and found that, even in a very restricted setting with only two agents having identical preferences, computing an allocation that minimizes envy is NP-hard. On the other hand, they did show that there always exists an allocation where the envy is bounded by the maximum marginal utility – i.e., the largest difference in utility that a single item can make – and they described a simple algorithm to find such an allocation.

Bouveret and Lang (2008) analyzed the computational complexity of finding whether an envy-free and Pareto efficient allocation exists (EEF EXISTENCE) in more depth for various representation languages. In particular, their focus was on dichotomous languages, where every agent specifies a propositional formula: if a formula holds for the bundle that an agent receives, his utility is 1; otherwise his utility is 0.

They found that EEF EXISTENCE is Σ_2^p-complete for dichotomous languages and remains NP-hard even after restricting the number of agents to two and requiring that they have identical preferences. Regarding additive languages,

they proved that EEF EXISTENCE is NP-complete when agents only have binary-valued preferences, i.e., every item is worth 1 or 0 utility, even under some very strong restrictions. De Keijzer et al. (2009) later extended these results by showing that the EEF EXISTENCE problem for general additive preferences is Σ_2^p-complete.

These results indicate that EEF EXISTENCE becomes intractable even in very restricted representation languages. In an attempt to find tractability results for small parameters, Bliem et al. (2016) performed a parameterized complexity analysis for additive preferences. They found tractable algorithms both for the case when the number of resources is very small and for the case when the number of agents together with the number of different values that occur in the utility functions is very small.

A different criterion for finding fair allocations is maximizing egalitarian welfare. This amounts to finding an allocation such that the utility of the agent that is worst off is maximized. This problem is also referred to in the literature as the 'Santa Claus problem': Santa Claus has m gifts that he wants to assign to n children such that the unhappiest child is as happy as possible (Bansal and Sviridenko, 2006).

Golovin (2005) shows that maximizing egalitarian welfare is NP-complete and this problem is even NP-hard to approximate under various assumptions. He does provide a polynomial-time algorithm for maximizing egalitarian welfare under binary-valued preferences, but proves that allowing $\{0, 1, 2\}$ preferences already makes the problem NP-hard.

Yet another solution concept that can play a role in fair allocation is utilitarian social welfare, where we simply add the utilities of all agents together to find the allocation with the largest sum of utilities. As Chevaleyre et al. (2008) note, there is a close connection between this setting and the winner determination problem in combinatorial auctions.

In a combinatorial auction, we are given a set of agents plus a set of items and each agent can place a bid on any combination of the items. The winner determination problem then corresponds to finding the assignment of items such that the total profit is maximized (Cramton et al., 2004). Such an assignment then also corresponds to the assignment that maximizes utilitarian welfare, so we can apply algorithmic results from the combinatorial auctions literature also to the setting of fair division.

Chevaleyre et al. (2008) in particular study the expressivity, succinctness and computational complexity of two types of representation languages: bundle form and k-additive form. In the bundle form, an agent lists all bundles of items to which he assigns a non-zero value, which is similar to how bidding languages in combinatorial auctions are usually represented. In the k-additive form on the other hand, an agent assigns utilities to all subsets of the items of size $\leq k$ and the utility of a bundle S is then described as the sum of the utilities of those subsets contained in S.

Both for the bundle form and k-additive form with $k \geq 2$, Chevaleyre et al. (2008) find that maximizing utilitarian welfare is NP-complete. Roos and Rothe (2010) and Nguyen et al. (2014) extend these results by showing NP-completeness

for egalitarian welfare and Nash product welfare as well in the case of both the bundle form and the 1-additive form.

Baumeister et al. (2013) look at representation languages for maximizing egalitarian welfare from a different perspective, namely by choosing positional scoring rules as representation language. The agents specify their preferences as an ordinal ranking over the items and these are then converted to cardinal utilities by a specific scoring vector, such as Borda scoring, lexicographic scoring or k-approval. They find NP-hardness results for most scoring vectors, but a tractability result for k-approval. Note that the k-approval setting here is equivalent to the binary-valued additive setting, for which Golovin (2005) also described a polynomial-time algorithm.

Darmann and Schauer (2015) extend these results on positional scoring rules for Nash welfare. Similar to Baumeister et al. (2013), they obtain NP-hardness results for Borda scoring and Lexicographic scoring, but a tractable algorithm for k-approval. Their approach is based on reducing the problem to a minimum-cost flow problem. Note again that this algorithm also implies an efficient algorithm for maximizing Nash welfare in the binary-valued additive case.

This result was also discovered by Barman et al. (2018). Apart from describing an approximation algorithm for when agents have identical valuations, they also present an exact algorithm in the additive binary-valued setting for finding a Nash-optimal solution. This algorithm does not directly depend on the minimum-cost flow problem, but iteratively finds greedy improvements using augmenting paths.

This scenario of binary-valued additive valuations and Nash welfare was further investigated by Halpern et al. (2020). Their findings include a proof that under binary-valued additive valuations, an allocation maximizes Nash welfare if and only if it is a leximin-optimal allocation. This establishes a close connection between maximizing Nash welfare and egalitarian welfare in this binary-valued setting. In particular, any Nash-optimal allocation thus also maximizes the egalitarian welfare. This result was independently also discovered by Aziz and Rey (2020).

Chapter 3

The dichotomous language

For finding efficient representation languages, both tractability and intractability results can help us. Intractability results can first of all show us which approaches do not work. Secondly, these results can also help to give more insight in what makes the problem difficult. This again can aid us in discovering how we need to restrict the general problem so that these difficult situations do not arise.

One intuitive approach for representing an agent's preferences is using propositional formulas, where the variables consist of the items $\mathcal{O}$ in the fair division problem. For example, such a formula could consist of an expression such as $\varphi = \neg c \wedge (a \vee b)$, meaning that, if we want φ to be satisfied, the agent should not be assigned item c and should be assigned either item a or b (or both). Representing preferences by a propositional formula makes it possible to express relations between objects (e.g. "I wish to receive all of these items" or "receiving one of these items already suffices for me as long as I also receive this other item"). More formally, we can define propositional formulas on items in a fair division problem in the following way.

Definition 3.1 (Propositional formulas on items)**.** Let $(N, \mathcal{O}, \boldsymbol{u})$ be a fair division instance, let φ be a propositional formula such that $\mathrm{Var}(\varphi) \subseteq \mathcal{O}$ and let $S \subseteq \mathcal{O}$ be a bundle of items.
We say that *bundle S satisfies formula φ*, or $S \vDash \varphi$, if φ becomes true under the following interpretation $M : \mathrm{Var}(\varphi) \to \{\mathrm{true}, \mathrm{false}\}$

$$M(v) = \begin{cases} \mathrm{true} & \text{if } v \in S \\ \mathrm{false} & \text{if } v \notin S \end{cases}$$

There are different ways to define representation languages based on propositional formulas. For example, we could let each agent specify a set of propositional formulas along with a weight for each formula. The utility of an agent's bundle is then equal to the sum of the weights of all the formulas that are satisfied according to that bundle. Uckelman et al. (2009) study representation languages of this form in more depth.

Because we are mainly interested in finding representation languages that are computationally tractable, we start by analyzing a very basic version of a

representation language based on propositional formulas called dichotomous preferences. Bouveret and Lang (2008) also study this representation language. In this language, every agent can specify a single propositional formula and the utility of an agent's bundle is 1 if this formula is satisfied according to this bundle and 0 otherwise. We can define this representation language in the following way.

Definition 3.2 (Dichotomous language)**.** An expression in the *dichotomous language* consists of a vector $\boldsymbol{\varphi} = (\varphi_1, \ldots, \varphi_n)$ of propositional formulas, where each φ_i is a propositional formula (representing agent i's preferences) that contains propositional symbols v_o for each object $o \in \mathcal{O}$.

Definition 3.3 (Utility of dichotomous language)**.** Given an expression in the dichotomous language $\boldsymbol{\varphi}$, we define the utility of each agent given a bundle $S \subseteq \mathcal{O}$ in the following way:

$$u_i(S) = \begin{cases} 1 & \text{if } S \vDash \varphi_i \\ 0 & \text{if } S \nvDash \varphi_i \end{cases}$$

Note that this language is still fairly restrictive, since every agent can only specify a single propositional formula and his preferences are either completely satisfied or unsatisfied. Still, this language can be interesting to analyze the complexity of as a starting point. Namely, if this language already is NP-hard to compute an optimal solution for, then most other languages based on propositional formulas will be NP-hard as well. This is because more complicated representation languages based on propositional formulas (such as allowing multiple weighted formulas) will at the very least also allow us to express dichotomous preferences in them, making the problem of finding an optimal solution at least as hard as in the dichotomous case.

3.1 Hardness results

In this section, we will show that, for all solution concepts we consider in this thesis, it is NP-hard to find the allocations elected by these solution concepts for the dichotomous language. The hardness result continues to hold even when we restrict the dichotomous language further. To show this, we first prove NP-hardness of a related intermediate problem OPT EXISTENCE. We then prove that we can reduce OPT EXISTENCE to any of the decision problems USW, ESW, NSW and EEF EXISTENCE, making these problems NP-hard as well.

Let OPT EXISTENCE be the following decision problem:

$$\text{OPT EXISTENCE} = \{\langle N, \mathcal{O}, \boldsymbol{\varphi}\rangle \mid \exists \pi \text{ such that } \pi(i) \vDash \varphi_i \text{ for all } i \in N\}$$

Intuitively, this decision problem checks whether there exists an allocation such that every agent is completely satisfied.

To prove NP-hardness, we follow a similar approach as taken for proving NP-hardness of EEF EXISTENCE in dichotomous languages in Bouveret and Lang (2008) (which we derive here as a corollary).

Theorem 3.4. OPT EXISTENCE in the dichotomous language with identical preferences, positive formulas and 2 agents is NP-hard.

Proof. To show hardness, we will reduce from SET SPLITTING, which is known to be NP-hard (Gary and Johnson, 1979, p. 221).

> SET SPLITTING
> *Input:* A set S and a collection $C = \{S_1, \ldots, S_c\}$ of subsets of S.
> *Question:* Does there exist a partition of S into S_a and S_b such that no subset in C is completely in either S_a or S_b?

Given an input instance $\langle S, C\rangle$ for SET SPLITTING, we show how to construct (in polynomial time) a dichotomous fair division problem G with identical preferences, positive formulas and 2 agents, such that $\langle S, C\rangle \in$ SET SPLITTING if and only if $G \in$ OPT EXISTENCE.

Let $G = \langle \{1, 2\}, S, (\varphi_1, \varphi_2)\rangle$, where $\varphi = \varphi_1 = \varphi_2 = \bigwedge_{S_i \in C} \bigvee_{s \in S_i} v_s$. Note that φ contains no negative literals. This formula φ amounts to requiring that the agents have at least one item of every subset S_i in the collection

Now suppose that there exists a set splitting in S_a and S_b. Then choose the allocation π with $\pi(1) := S_a$ and $\pi(2) := S_b$. We have that $S_a \vDash \varphi$ and $S_b \vDash \varphi$, since no subset in C is completely in S_a or S_b, so every subset S_i of C must have some of its items in S_a and the others in S_b.

Conversely, suppose that there exists an allocation π such that $\pi(1) \vDash \varphi$ and $\pi(2) \vDash \varphi$. Then we claim that choosing $S_a := \pi(1)$ and $S_b = \pi(2)$ is a set splitting. Suppose not, then there is a subset S_i of C that is completely contained in S_a (w.l.o.g.), meaning that $S_i \cap S_b = \emptyset$ and thus $S_a \nvDash \varphi$, contradicting our assumption. Hence, we have proved our claim. □

Corollary 3.5. USW in the dichotomous language with identical preferences, positive formulas and 2 agents is NP-complete.

Proof. Note that a two-player dichotomous fair allocation problem has maximum utilitarian social welfare 2 if and only if there is an allocation that satisfies φ_i for both agents. Hence, we can reduce OPT EXISTENCE to USW. □

Corollary 3.6. ESW in the dichotomous language with identical preferences, positive formulas and 2 agents is NP-complete.

Proof. Note here that a dichotomous fair allocation problem has maximum egalitarian social welfare 1 if and only if the worst-off agent has utility 1. This is precisely the case if there exists an allocation such that φ_i is satisfied for every agent. Hence, we can reduce OPT EXISTENCE to ESW. □

Corollary 3.7. NSW in the dichotomous language with identical preferences, positive formulas and 2 agents is NP-complete.

Proof. A dichotomous fair allocation problem has maximum Nash welfare 1 exactly if every agent receives utility 1. Note that if some agent receives utility 0 instead of 1, then the product of the utilities will also be 0. So the maximum

Nash welfare is 1 if and only if there exists an allocation such that φ_i is satisfied for each agent. Thus, we can reduce OPT EXISTENCE to USW. □

Corollary 3.8 (Bouveret and Lang (2008))**.** EEF EXISTENCE in the dichotomous language with identical preferences, positive formulas and 2 agents is NP-hard.

Proof. First of all, as the preferences are identical and dichotomous, an allocation is envy-free if and only if it satisfies either all agents or none. Secondly, note that we can always satisfy at least one agent's formula, since all formulas are positive. So satisfying none of the agents is not Pareto-efficient. This means that a dichotomous and identical fair allocation problem with positive formulas is envy-free and efficient if and only if all agents are satisfied. Thus, we can also reduce OPT EXISTENCE to MAX ESW. □

3.2 Restricting the formulas to clauses

One way to obtain positive results is to constrain the class of allowed propositional formulas in a different way. A possible restriction is to assume that the agents' preferences are represented by a single clause, i.e., only disjunctions of literals. We will show in this section that this restriction makes the problem tractable for the different solution concepts we consider.

We do this by first proving that a different problem, SATISFY AGENTS, is polynomial-time solvable and subsequently showing how we can use this to find the allocations elected by the different solution concepts. This problem is defined as follows:

SATISFY AGENTS $= \{\langle N, \mathcal{O}, \boldsymbol{\varphi}, k\rangle \mid \exists \pi$ such that $\pi(i) \vDash \varphi_i$ for at least k agents $i \in N\}$

Proposition 3.9. SATISFY AGENTS for the dichotomous language is solvable in polynomial time when each agent specifies his preferences as a single positive clause.

Proof. Let $\langle N, \mathcal{O}, \boldsymbol{\varphi}\rangle$ be a fair allocation, where the formulas in $\boldsymbol{\varphi} = (\varphi_1, \ldots, \varphi_n)$ are positive clauses and let k be arbitrary. We let $\mathcal{O}_{\varphi_i}$ denote the objects occurring as disjunctions in formula φ_i. Now define the bipartite graph $\langle V, E\rangle$ with $V = N \cup \mathcal{O}$ (the agents in one partition and objects in the other) and $E = \{(i, o) \mid o \in \mathcal{O}_{\varphi_i}\}$.

We now claim that $\langle V, E\rangle$ contains a matching of size k if and only if $\langle N, \mathcal{O}, \boldsymbol{\varphi}\rangle$ can satisfy at least k agents. Suppose that we have a maximum matching of size k. For every (i, o) edge in the matching, we assign item o to agent i. As this is a matching, every item is assigned only once. Furthermore, $o \in \mathcal{O}_{\varphi_i}$, so assigning o to agent i will satisfy φ_i for him regardless of the rest of the rest of his bundle. Conversely, if there exists an assignment π that satisfies k agents, this means that every of those k agents are assigned at least one item o_i^* occurring in $\mathcal{O}_{\varphi_i}$. For each of those k agents, include an edge from that item o_i^* to the agent to construct a matching of size k in $\langle V, E\rangle$.

We can find a maximum bipartite matching in polynomial time using for example the Hopcroft-Karp algorithm and then check if its size is larger or equal than k. Hence, SATISFY AGENTS is solvable in polynomial time. □

Now, we will use this algorithm for SATISFY AGENTS to solve the problem for the various solution concepts. Contrary to Proposition 3.9, in the following theorems the single clause that each agent specifies does not need to be positive. That is, the clause could also contain negative literals.

Theorem 3.10. MAX USW for the dichotomous language restricted to single clauses is solvable in polynomial time, even if we require the allocation to be complete.

Proof. We start by showing that the problem with general clauses (including possibly negative literals) can be reduced to the problem with only positive clauses. Suppose agent i has a clause φ_i that contains literal $\neg o$ for some item o. Then there are two cases: (1) either all other agents also have $\neg o$ in their formula or (2) there exists an agent j that does not have such a literal. In the latter case, we can remove agent i from the problem (since by not assigning o to him, we completely satisfy his formula). If it turns out in the final allocation that o is not yet divided, we can always assign it to agent j.

In the former case, all agents have $\neg o$ as literal in their formula. We check whether there exists an agent that has another literal besides $\neg o$. If such an agent exists, we can always satisfy one of his other literals by assigning (or not assigning him) that item. If we then give that agent also o, his clause is still satisfied and all other agents' clauses are also satisfied (as they are indeed all not assigned o). So we have found an allocation that has the maximum possible social welfare. If such an agent does not exist, we can satisfy $n - 1$ agents.

To solve the problem with positive clauses, note that every agent receives utility 1 if and only if his formula is satisfied. Thus, finding the maximum sum of utilities is equivalent to finding an allocation that satisfies the maximum number of agents. This can be done in polynomial time using the algorithm for SATISFY AGENTS. □

Theorem 3.11. MAX ESW for the dichotomous language restricted to single clauses is solvable in polynomial time, even if we require the allocation to be complete.

Proof. The proof of this is very similar to the proof for MAX USW. Here, note that we have either an egalitarian social welfare of 1 if we can satisfy every agent, or a welfare of 0 if this is not the case. So calculating the maximum egalitarian welfare is equivalent to checking whether we can satisfy every agent.

We can handle negative literals in clauses in the same way as in the proof for MAX USW. In case (1) we remove agent i from the problem and in case (2) we either have egalitarian welfare 1 if there exists an agent with another literal and egalitarian welfare 0 otherwise. So the problem that remains is checking whether we can satisfy all n agents' formulas, which can be done in polynomial time using the algorithm for SATISFY AGENTS. □

Theorem 3.12. MAX NSW for the dichotomous language restricted to single clauses is solvable in polynomial time, even if we require the allocation to be complete.

Proof. Note that in the dichotomous language, an allocation only has Nash welfare 1 if every agent receives utility 1. If a single agent receives utility 0, then the Nash welfare will be 0 as well (since then we multiply by 0). So there exists an allocation with Nash welfare 1 if and only if there exists an allocation with egalitarian welfare 1. Since MAX ESW is solvable in polynomial time (see Theorem 3.11), MAX NSW is also solvable in polynomial time. □

Theorem 3.13 (Bouveret and Lang (2008))**.** EEF EXISTENCE for dichotomous preferences restricted to single clauses is solvable in polynomial time, even if we require the allocation to be complete.

Proof. We claim that egalitarian social welfare 1 is achievable if and only if there exists an envy-free and efficient allocation. Suppose there exists an allocation with egalitarian social welfare 1. Then every agent is completely satisfied, so there is no envy and this is the best situation possible.

For the other direction, suppose we have an allocation that is envy-free and Pareto-efficient. If every agents' formula is not satisfied, then the allocation is not Pareto-efficient, since we can always satisfy at least one agent. On the other hand, if we have an allocation where there exists an agent that is not satisfied, then one the items in his formula must have been assigned to another agent, so he envies that agent[1].

Hence, we can solve EEF EXISTENCE using our algorithm for MAX ESW. □

[1]Note in the case where the non-satisfied agent has only negative clauses that there exists an agent that does not have all these items, which he envies.

Chapter 4

The additive language

In the additive language, the value of a bundle is simply equal to the sum of the value of the individual items. This does mean that there is no interaction in utility possible between different items. However, if the values of the items are mostly independent from each other, then this can be a good model to work with. Although this model looks restrictive, some solution concepts are already computationally intractable to find the elected allocation for in this case.

4.1 Basic results

If we assume that the preferences of the objects are additive, then it becomes easy to represent the preferences efficiently. For the representation language, we only need to keep track for every agent of the utility he assigns to all the items individually.

Definition 4.1 (Additive language)**.** An expression in the *additive language* consists of a set of utility functions $\boldsymbol{u} = (u_1, \ldots, u_n)$, with utility function $u_i : \mathcal{O} \to \mathbb{N}$ indicating the utility value for all the different objects in $\mathcal{O}$ according to agent i.

Definition 4.2 (Utility of additive language)**.** Given an expression in the additive language $\boldsymbol{u}$, we define the utility of each agent given a bundle $S \subseteq \mathcal{O}$ in the following way:

$$u_i(S) = \sum_{o \in S} u_i(o)$$

In case we are interested in maximizing utiltarian social welfare, we can do this easily under the assumption that the preferences are additive.

Proposition 4.3. MAX USW for the additive language is computable in polynomial time.

Proof sketch. There exists a simple algorithm that calculates a maximum utilitarian allocation. Namely, find for every item $o \in \mathcal{O}$ the agent i that assigns the highest utility to that item (i.e., $\text{argmax}_{i \in N} u_i(o)$) and assign the item to that agent. □

However, for the other solution concepts, such as ESW, NSW and EEF, even the additive setting already becomes NP-hard to optimize for. We refer to the results in the following papers.

Theorem 4.4 (Lipton et al., 2004)**.** ESW for the additive language is NP-complete, even for two agents.

Theorem 4.5 (Roos and Rothe, 2010)**.** NSW for the additive language is NP-complete, even for two agents.

Theorem 4.6 (De Keijzer et al., 2009)**.** EEF EXISTENCE for the additive language is Σ_2^p-complete.

Note here that Σ_2^p denotes the second level of the polynomial hierarchy. Any Σ_2^p-hard language is also NP-hard.

Theorem 4.7 (Bouveret and Lang, 2008)**.** EEF EXISTENCE for the additive language with only two agents is NP-hard.

4.2 Unary encoding

We can conclude that allowing full additive preferences in the representation language makes the language already too expressive to be able to find the allocations elected by solution concepts (except in the case of USW). In order to discover more tractable results, we need to make further restrictions to our model. One such restriction to the additive language is to furthermore require that the utilities of the objects that are specified are encoded in unary.

However, even under this restriction, the problem remains NP-hard for all three solution concepts, as we prove below. The general construction of the hardness proof is based on the hardness proof in Bliem et al. (2016), where NP-completeness is proved for EEF EXISTENCE for unary additive preferences. We generalize the idea of this proof such that the reduction also works for ESW and NSW.

Theorem 4.8. ESW for additive languages with preferences encoded in unary is NP-hard, even if all agents have the same preferences.

Proof. We will reduce from the NP-complete problem UNARY BIN PACKING (Jansen et al., 2013).

> UNARY BIN PACKING
> *Input:* A set I of items, a (positive) weight s_i for each $i \in I$, the bin capacity B and an integer K all encoded in unary.
> *Question:* Is there a partition of the items $I_1, \ldots, I_K$ such that $\sum_{i \in I_j} s_i \leq B$ for all $j \in \{1, \ldots, K\}$?

Suppose we are given a set of items I, weights for these items s_i and a bin capacity B and integer K. We will define the following polynomial-time computable map f with $f(\langle I, \langle s_i \mid i \in I\rangle, B, K\rangle) = F$. Here, $F = \langle N, \mathcal{O}, \boldsymbol{u}\rangle$ is an additive fair division problem such that F has an allocation with egalitarian welfare B if and only if we can partition the items in K partitions such that none of them exceed capacity B.

To do so, we let $N = \{1, \dots K\}$ and $\mathcal{O} = I \cup \{d_1, \dots, d_q\}$, with $q = B \cdot K - \sum_{i \in I} s_i$. We define $u = u_1 = u_2 = \dots = u_K$ in the following way: for $i \in I$, $u(i) = s_i$ and $u(d_i) = 1$. The intuition behind this reduction is to try to divide all items equally over the K agents such that they all have utility exactly B (and thus egalitarian welfare B). We use the d_i items of size 1 to fill up the remaining space that is left over after dividing the items.

To prove correctness, suppose that there exists a bin packing for items I, capacity B in K bins. Then there must exist a partition $I_1, \dots I_K$ such that $\sum_{i \in I_j} s_i \leq B$ for all $j \in \{1, \dots, K\}$. We can now create an allocation where we assign all items occurring in I_j to agent j. Furthermore, we also assign $K - \sum_{i \in I_j} s_i$ dummy items of size 1 to agent j, so that his total utility is exactly K (note that we have exactly enough dummy items to do so by construction). Then every agent has utility exactly K and we thus have an egalitarian social welfare of K.

For the converse, suppose that there exists an allocation π such that the egalitarian welfare is K. Then every agent must have utility exactly K, meaning that we have total utility $B \cdot K$. Note that any allocation will have at most $\sum_{i \in I} s_i + B \cdot K - \sum_{i \in I} s_i = B \cdot K$ total utility (with equality if and only if the allocation is complete). So this allocation is complete and moreover assigns precisely K utility to every agent (if there was an agent with $K + 1$ utility, the total sum would be larger than $B \cdot K$). Now define $I_j = \pi(j) \cap I$ for each bin $j \in \{1, \dots, K\}$, then we have that $\sum_{i \in I_j} s_i \leq \sum_{o \in \pi(j)} u(o) \leq K$, so this indeed forms a valid bin packing. □

Theorem 4.9. NSW for additive languages with preferences encoded in unary is NP-complete, even if all agents have the same preferences.

Proof. To show NP-hardness, we reduce from the same NP-complete problem UNARY BIN PACKING. Given an input instance $\langle I, \langle s_i \mid i \in I \rangle, B, K \rangle$, we use the same map f as defined in the proof of Theorem 4.8 to construct fair division problem $F = f(\langle I, \langle s_i \mid i \in I \rangle, B, K \rangle)$.

We already know by Theorem 4.8 that we can partition the items in K partitions without exceeding B if and only if F has an allocation with egalitarian welfare B. We now claim that an allocation π of F has egalitarian welfare at least B if and only if π has Nash welfare at least $B^{|N|}$.

First of all, suppose that F has an allocation π with egalitarian welfare at least B. Then for all agents $i \in N$ it holds that $u_i(\pi(i)) \geq B$, meaning that $\text{NSW}(\pi) = \prod_{i \in N} u_i(\pi(i)) \geq B^{|N|}$.

For the converse, suppose that F has an allocation π with Nash welfare at least $B^{|N|}$. Note that for any allocation π' of F it holds that $\text{USW}(\pi') \leq B \cdot K$ (with equality happening precisely if the allocation is complete), so this also holds for allocation π. From this, we can conclude the following:

$$\sqrt[|N|]{\text{NSW}(\pi)} = \sqrt[|N|]{\prod_{i \in N} u_i(\pi(i))} \leq^{(*)} \frac{\sum_{i \in N} u_i(\pi(i))}{|N|} = \frac{\text{USW}(\pi)}{|N|} \leq \frac{B \cdot K}{K} = B$$

where $(*)$ holds because of the inequality of arithmetic and geometric mean. $(*)$ is an equality if and only if $i, j \in N$ implies that $u_i(\pi(i)) = u_j(\pi(j))$.

But by our assumption $\sqrt[|N|]{\text{NSW}(\pi)} = B$, so this must mean that $(*)$ is an equality. This implies that for any two agents $i, j \in N$ it holds that $u_i(\pi(i)) = u_j(\pi(j))$, so all agents have the same utility and this utility is equal to B. Hence, $\text{ESW}(\pi) = B$. □

Theorem 4.10. EEF EXISTENCE for additive languages with preferences encoded in unary is NP-complete, even if all agents have the same preferences.

Proof. We again use the same reduction f as in the proof of Theorem 4.8. We know for this reduction that F has an allocation with egalitarian welfare B if and only if it is possible to partition the items in K partitions without exceeding capacity B. We now claim furthermore that that an allocation π has egalitarian welfare B if and only if π is envy-free and Pareto efficient.

To see this, suppose first of all that π is an allocation with egalitarian welfare B. Since the total utility possible is at most $B \cdot K$, this must mean that every agent receives exactly B utility and that the allocation is complete. Because every agent has the exact same utility function and they all receive the same utility, no agent envies another agent. Moreover, since the allocation is complete and all agents assign the same utilities to all items, it is impossible to increase the utility of an agent without decreasing the utility of another agent. Hence, π is also Pareto efficient.

Conversely, suppose that π is an allocation that is envy-free and Pareto efficient. First of all, the allocation must be complete, for if not we could construct an allocation π' where we assign an unallocated item to some agent i, which strictly increases the utility of agent i while keeping the utility of the other agents the same. Then π' would Pareto dominate π, contradicting that π is Pareto efficient. Hence, the sum of all utilities must be exactly $B \cdot K$.

Secondly, every agent must receive the exact same utility. Suppose not, then some agent i receives a bundle such that $u_i(\pi(i)) > u_j(\pi(j))$ for another agent j. But since every agent has the same utility function $u_i(\pi(i)) = u_j(\pi(j))$, which would mean that agent j envies i's bundle. Combined with the fact that all utilities must sum to $B \cdot K$, it must be the case that all agents have utility exactly B, meaning that $\text{ESW}(\pi) = K$. □

4.3 Binary-valued preferences

Since even additive languages encoded in unary are NP-hard for egalitarian welfare, Nash welfare and envy-freeness, we need to look at a further restriction. To this end, we restrict ourselves to binary-valued preferences: instead of allowing an agent to specify an arbitrary positive utility value for every item, the agent can only choose either 0 or 1 as value for an item.

Definition 4.11 (Binary-valued preferences)**.** An expression in the additive language $\boldsymbol{u} = (u_1, \ldots, u_n)$ has *binary-valued preferences* if for all agents $i \in N$ and all items $o \in \mathcal{O}$ it is the case that $u_i(o) \in \{0, 1\}$.

Note that this is a different kind of restriction than the unary encoding that we discussed in the previous section. The unary restriction still allows agents to

specify any utility value for an item, but the value should be encoded in unary, meaning that the utility value is bounded by a polynomial of the input size. On the other hand, with binary-valued preferences the only two utility values that an agent can specify for an item are 0 and 1. So this is a much greater restriction.

Under this extra restriction, the problem now becomes polynomial-time solvable for egalitarian social welfare, as is also shown by Golovin (2005). Since the proof of this statement is omitted in Golovin (2005) and for the sake of completeness, we include the proof here.

Theorem 4.12. MAX ESW for the binary-valued additive language is computable in polynomial time.

Proof sketch. We reduce this problem to a maximum flow instance. We first solve a slightly different problem: given an integer k, does there exist an allocation with ESW greater or equal than k? If we have an efficient algorithm for this problem, we can also solve the original problem of finding an allocation with optimal ESW by performing a binary search on k.

We construct the following graph with capacities. We define a source node s, a sink node t, a node v_o for every item $o \in \mathcal{O}$ and a node v_i for every agent $i \in N$. We add edges from s to every item node v_o with capacity 1 and edges from every agent node v_i to t with capacity k. Lastly, we add an edge between any item node v_o and agent node v_i if and only if agent i assigns utility 1 to item o. Formally, $G = (V, E)$ with:

$$V = \{s, t\} \cup \{v_o \mid o \in \mathcal{O}\} \cup \{v_i \mid i \in N\}$$
$$E = \{(\langle s, v_o\rangle, 1) \mid o \in \mathcal{O}\} \cup \{(\langle v_i, t\rangle, k) \mid t \in N\} \cup \{(\langle v_o, v_i\rangle, 1) \mid u_i(o) = 1\}$$

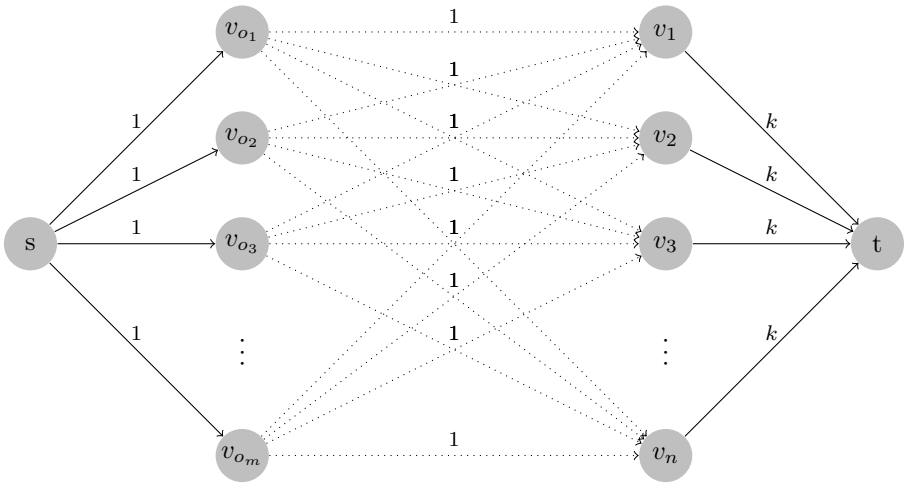


Figure 4.1: The max flow graph G as described in the proof of Theorem 4.12. Note that a dotted edge between a node v_{o_j} and v_i exists if and only if $u_i(o_j) = 1$.

This graph is also illustrated in Figure 4.1. G has a maximum flow of size $n \cdot k$

if and only if there exists an allocation π with $\text{ESW}(\pi) \geq k$. Using for example the Ford-Fulkerson algorithm, we can check for this in polynomial time. □

Regarding Nash welfare, it turns out that finding an optimal solution for the binary-valued additive language is also possible in polynomial time.

Theorem 4.13 (Darmann and Schauer (2015) and Barman et al. (2018))**.** MAX NSW for the binary-valued additive language is computable in polynomial time.

Darmann and Schauer (2015) present an algorithm for this approach that relies on a minimum cost flow. Barman et al. (2018) on the other hand provide a greedy algorithm that works by iteratively finding augmenting paths to find an allocation that improves the current Nash welfare.

Aziz and Rey (2020) and Halpern et al. (2020) later found that in the binary-valued additive case, Nash-optimal and leximin-optimal solutions coincide.

Theorem 4.14 (Aziz and Rey (2020) and Halpern et al. (2020))**.** In the binary-valued additive language, an allocation has maximum Nash welfare if and only if that allocation is leximin-optimal.

In Chapter 7, we will present an algorithm for finding optimal leximin and Nash welfare for a generalization of the binary-valued additive language in more detail and prove its correctness. In addition, we investigate and prove the link between leximin and Nash welfare for a more general setting.

Lastly, envy-freeness however remains NP-hard even when restricted to binary-valued additive preferences, as is shown by Bouveret and Lang (2008).

Theorem 4.15 (Bouveret and Lang, 2008)**.** EEF EXISTENCE with binary-valued additive preferences is NP-complete.

Chapter 5

The weighted bids language

5.1 Definitions

In the previous chapter, we have looked at the additive language. This language however does not allow you to express utility relations between objects, such as specifying that two goods are complementary or mutually exclusive.

A natural way to express these kind of preferences is using weighted bids. A bid consists of a subset of the items $\mathcal{O}$ and an integer representing the weight. Each agent can specify multiple bids. There are two ways to define the utility of a bundle given these bids: conjunctively or disjunctively. With conjunctive weighted bids, an agent receives the utility u' belonging to a bid B' if his bundle contains all items in B'. With disjunctive weighted bids on the other hand, an agent receives utility u' of a bid B' if his bundle contains at least one item in B'.

More formally, we can define conjunctive weighted bids in the following way:

Definition 5.1 (Conjunctive weighted bids)**.** An expression in the *conjunctive weighted bids language* consists of a vector $\boldsymbol{B} = (B_1, \ldots, B_n)$, where each $B_i = \{(B_{i,1}, u_{i,1}), \ldots, (B_{i,r}, u_{i,r})\}$ represents the bids of agent i with $B_{i,j} \subseteq \mathcal{O}$ and $u_{i,j} \in \mathbb{N}$.

Definition 5.2 (Utility of conjunctive weighted bids)**.** Given an expression in the conjunctive weighted bids language $\boldsymbol{B}$, we define the utility of an agent i given a bundle $S \subseteq \mathcal{O}$ as follows:

$$u_i(S) = \sum_{\substack{(B_{i,j}, u_{i,j}) \in B_i \\ \text{s.t. } B_{i,j} \subseteq S}} u_{i,j}$$

We will now give an example of how we can specify complementary preferences – items becoming worth more when combined – using conjunctive weighted bids.

Example 5.3 (Complementary preferences)**.** Suppose we have three items o_1, o_2, o_3, where o_1 and o_2 are worth 2 utility separately and o_3 is worth 1 utility separately. We want to specify furthermore that if we are assigned all three of

them, we obtain 10 extra utility. So the three items together are worth more than they are separately. We would express this with the bids

$$B_i = \{(\{o_1\}, 2), (\{o_2\}, 2), (\{o_3\}, 1), (\{o_1, o_2, o_3\}, 10)\}.$$

Following Definition 5.2, this would yield us the following utilities for this agent:

$$\begin{aligned} u_i(\emptyset) &= 0 & u_i(\{o_1, o_2\}) &= 4 \\ u_i(\{o_1\}) &= 2 & u_i(\{o_1, o_3\}) &= 3 \\ u_i(\{o_2\}) &= 2 & u_i(\{o_2, o_3\}) &= 3 \\ u_i(\{o_3\}) &= 1 & u_i(\{o_1, o_2, o_3\}) &= 15 \end{aligned}$$

△

For disjunctive bids, we use the same representation as for conjunctive bids. So Definition 5.1 is identical to Definition 5.4. The difference is in the utility of a bundle: when does the weight of a bid contribute to the utility of a bundle? This leads to the following definitions:

Definition 5.4 (Disjunctive weighted bids)**.** An expression in the *disjunctive weighted bids language* consists of a vector $\boldsymbol{B} = (B_1, \ldots, B_n)$, where each $B_i = \{(B_{i,1}, u_{i,1}), \ldots, (B_{i,r}, u_{i,r})\}$ represents the bids of agent i with $B_{i,j} \subseteq \mathcal{O}$ and $u_{i,r} \in \mathbb{N}$.

Definition 5.5 (Utility of disjunctive weighted bids)**.** Given an expression in the disjunctive weighted bids language $\boldsymbol{B}$, we define the utility of an agent i given a bundle $S \subseteq \mathcal{O}$ as follows:

$$u_i(S) = \sum_{\substack{(B_{i,j}, u_{i,j}) \in B_i \\ \text{s.t. } B_{i,j} \cap S \neq \emptyset}} u_{i,j}$$

Disjunctive bids make it easy to express mutually exclusive preferences – i.e., goods that are worth less than they are individually when multiple of them are assigned to the same agent.

Example 5.6 (Mutually exclusive preferences)**.** Consider the scenario of event tickets. There are two different events that you would like to go to. If you can go to event A, you experience a utility of 5, while going to event B gives you utility 10. There are three different tickets possible: o_a, a ticket for event A, o_b, a ticket for event B, or o_c, a combination ticket for both events. Having more than one ticket for an event does not yield you any extra utility.

We can express these kind of preferences using the following bids:

$$B_i = \{(\{o_a, o_c\}, 5), (\{o_b, o_c\}, 10)\}\}$$

Following Definition 5.5, we would then get the following utilities:

$$\begin{aligned} u_i(\emptyset) &= 0 & u_i(\{o_a, o_b\}) &= 15 \\ u_i(\{o_a\}) &= 5 & u_i(\{o_a, o_c\}) &= 5 \\ u_i(\{o_b\}) &= 10 & u_i(\{o_b, o_c\}) &= 10 \\ u_i(\{o_c\}) &= 15 & u_i(\{o_a, o_b, o_c\}) &= 15 \end{aligned}$$

△

Now that we have some intuition in what kind of preferences we can represent using disjunctive and conjunctive weighted bids, we will look into the computational complexity of computing allocation elected by solution concepts for these languages.

In Section 5.2, we investigate the complexity of this for conjunctive weighted bids. For envy-freeness, Nash welfare and egalitarian welfare, this will turn out to be intractable already for the unweighted variant. For utilitarian welfare, we first of all discuss the similarities with combinatorial auctions and subsequently we find a close connection between maximizing utilitarian welfare and finding the maximum weighted independent set of a graph.

In Section 5.3, we take a closer look at the complexity results of disjunctive weighted bids. In particular, we will see that a very limited model of these types of bids is already intractable.

5.2 Conjunctive weighted bids

The complexity of finding an allocation elected by a solution concept depends on the specific solution concept used. As we have seen in the previous chapter, Nash welfare, egalitarian welfare and envy-freeness were already NP-hard for the additive language. This unfortunately makes the same problem intractable for conjunctive weighted bids languages as well, as we will show in Section 5.2.1. We study the complexity properties of utilitarian welfare separately in Section 5.2.2

5.2.1 ESW and NSW

As we have shown in Section 4.1, finding an optimal allocation with respect to Nash welfare or egalitarian welfare for the additive language is NP-hard. We can express the additive language using conjunctive weighted bids. Given a function $u_i : \mathcal{O} \to \mathbb{N}$ representing the utility of every item, we simply create a singleton bid for every item. That is, $B_i = \{(\{o\}, u_i(o)) \mid o \in \mathcal{O}\}$.

This already shows that computing allocations elected by these solution concepts must be NP-hard for conjunctive weighted bids as well. However, this might just be due to the fact that we allow weights for our bids. Therefore, it is interesting to analyze whether the problem remains NP-hard for unweighted conjunctive bids, as we cannot reduce the additive language to this in the same way.

We will show that even with unweighted bids, we are able to express additive preferences in this language, albeit only additive preferences where the utilities are encoded in unary. We can then use this to show that the solution concepts ESW and NSW are NP-hard, since the unary-encoded additive language is already NP-hard for these solution concepts.

Theorem 5.7. Let F be a fair division problem in the additive language where the utilities are encoded in unary. Then there exists an equivalent fair division problem F' in the conjunctive weighted bids language where all weights are equal to 1.
That is, for any $\boldsymbol{k} \in \mathbb{N}^n$, F has an allocation π with $u_i(\pi(i)) \geq k_i$ for all agents $i \in N$ if and only if F' has an allocation π' with $u_i(\pi'(i)) \geq k_i$ for all $i \in N$.

Proof. Let $F = (N, \mathcal{O}, \boldsymbol{u})$ be an expression in the additive language where the utilities are encoded in unary. We convert this into a conjunctive weighted bids language expression with weights 1 by adding, for every object $o' \in \mathcal{O}$ and every agent $i \in N$, $u_i(o')$ extra dummy items $o_{o',i,j}$. Since all utilities are encoded in unary, we will still have a polynomial number of items in $\mathcal{O}'$.

For every dummy item, we create a bid $(\{o', o_{o',i,j}\}, 1)$. The idea is that every agent has his own dummy items, so we can assign all dummy items always to the respective agents. This means that if an item o' is assigned to an agent, we have also satisfied all $(\{o', o_{o',i,j}\}, 1)$ bids, so we have $u_i(o')$ satisfied bids in total yielding $u_i(o')$ utility together. But we cannot satisfy any of the bids without receiving item o', so the agent receives 0 utility in case he does not get assigned o'. More formally, we construct a fair division problem in the conjunctive weighted bids language $F' = (N', \mathcal{O}', \boldsymbol{B}')$ with

$$
\begin{aligned}
N' &= N, \\
\mathcal{O}' &= \mathcal{O} \cup \bigcup_{o' \in \mathcal{O}, i \in N} \{o_{o',i,j} \mid 1 \leq j \leq u_i(o')\}, \\
B'_i &= \bigcup_{o' \in \mathcal{O}} \bigcup_{0 \leq j \leq u_i(o')} \{(\{o', o_{o',i,j}\}, 1)\}.
\end{aligned}
$$

Let $\boldsymbol{k} \in \mathbb{N}^n$. We claim that F has an allocation π such that each agent $i \in N$ has $u_i(\pi(i)) \geq k_i$ if and only if F' has an allocation π' such that each agent $i \in N$ has $u_i(\pi'(i)) \geq k_i$.

Suppose that F has an allocation π such that each agent $i \in N$ has $u_i(\pi(i)) \geq k_i$. We then define an allocation π' for F' with $\pi'(i) = \pi(i) \cup \{o_{o',i,j} \in \mathcal{O}' \mid o' \in \pi(i)\}$. It can be checked that indeed each agent i then also has utility at least k_i in F'.

Conversely, suppose that F' has an allocation π' with $u_i(\pi'(i)) \geq k_i$ for each $i \in N$. We define allocation π for F with $\pi(i) = \pi'(i) \cap \mathcal{O}$. To see that allocation π has at least as much utility as π' for an agent $i \in N$, consider any object $o' \in \mathcal{O}$. If $o' \notin \pi'(i)$, then none of the bids $(\{o', o_{o',i,j}\}, 1)$ are satisfied. If however $o' \in \pi'(i)$, note that the number of bids of the form $(\{o', o_{o',i,j}\}, 1)$ that are satisfied is at most $u_i(o')$, yielding a total utility of $u_i(o')$ in F'. But $o' \in \pi(i)$ means that we also receive utility $u_i(o')$ in F. Hence, the utility of agent i according to π is at least as high as according to π', meaning that $u_i(\pi(i)) \geq k_i$. □

Corollary 5.8. ESW and NSW for conjunctive weighted bids are NP-complete, even if the agents' preferences are identical and all weights are 1.

Proof. By Theorems 4.8 and 4.9, ESW and NSW are NP-hard for the unary additive language where each agent has identical preferences.

Let F be a unary additive fair division problem with identical preferences. By Theorem 5.7, we can encode F into a conjunctive weighted bids problem with weights 1 and identical preferences F'. F' then has an allocation where all agents i have utility larger or equal than k_i if and only if F has an allocation where all agents i have utility larger or equal than k_i. So, F has an allocation π with $\text{ESW}(\pi) \geq x$ if and only if F' has an allocation π' with $\text{ESW}(\pi') \geq x$.

Hence, we can solve the problem of checking whether, given a unary additive fair division problem with identical preferences, there exists an allocation with egalitarian welfare larger or equal than k efficiently if we can solve ESW for conjunctive weighted bids with weights 1 and identical preferences efficiently, implying that this problem is also NP-hard.

Very similarly, we can prove NP-hardness for NSW. F has an allocation π with $\text{NSW}(\pi) \geq x$ precisely when F' has an allocation π' with $\text{NSW}(\pi') \geq x$. Thus, if we can solve NSW for conjunctive weighted bids with weights 1 and identical preferences efficiently, we can also use this to solve NSW for the unary additive language with identical preferences. □

5.2.2 USW

The problem of optimizing utilitarian welfare for conjunctive weighted bids is closely related to two other well-studied problems: winner determination in combinatorial auctions and the weighted independent set problem. In this section, we will study the relation between these problems and USW for conjunctive weighted bids more closely and utilize this to obtain complexity results for the USW problem.

5.2.2.1 Link with combinatorial auctions

When optimizing for utilitarian welfare, the conjunctive weighted bids language is closely related to the winner determination problem in combinatorial auctions. In this section, we take a closer look at the similarities and differences of these two concepts. We follow the definitions from Cramton et al. (2004).

Definition 5.9 (Combinatorial auctions)**.** A *combinatorial auction* consists of a tuple $(N, G, \boldsymbol{v})$, where

- $N = \{1, \ldots, n\}$ is the set of agents;
- $G = \{1, \ldots, m\}$ is the set of items;
- $\boldsymbol{v} = (v_1, \ldots, v_n)$ are the valuation functions of each agent with $v_i : 2^G \to \mathbb{N}$.

The idea is that an agent can submit bids by reporting a valuation function representing for each combination of items the price he is willing to pay for receiving those items. After all agents have submitted their bids, the auctioneer then chooses an allocation such that his total revenue is maximized. We call the problem of determining the optimal outcome, given a combinatorial auction instance and the agents' valuations, the *winner determination* problem.

Definition 5.10 (Winner determination)**.** The *winner determination* problem finds an allocation π^* for which it holds that

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_{i \in N} v_i(\pi(i)),$$

where π ranges over all possible allocations.

Note that the setting of combinatorial auctions is almost identical to the setting of fair division problems (compare with how we defined a fair division problem in Definition 2.1). The difference is mostly in which part we are trying to optimize.

In fair division problems, the objective is to find an allocation such that the utility is fairly divided among the agents. In combinatorial auctions on the other hand, we are only interested in maximizing the revenue of the auctioneer and we do not need to take fairness into account.

However, if we are concerned with finding an allocation optimizing utilitarian welfare in a fair division problem, this problem becomes the same as the winner determination problem. Results from the combinatorial auction literature can thus also be relevant for optimizing utilitarian welfare in fair division settings.

Since expressing a valuation function $v_i : 2^G \to \mathbb{N}$ explicitly would require exponential size, bidding languages are used in combinatorial auctions to represent preferences more compactly. Representation languages for valuation functions for combinatorial auctions are usually defined using atomic bids combined by XOR and OR constructs. Following Nisan (2000), this leads to these definitions:

Definition 5.11 (Atomic bids)**.** An agent specifies an atomic bid (S, p), with $p \in \mathbb{N}$ indicating the price the agent is willing to pay for the items $S \subseteq G$. The valuation function $v_{(S,p)}$ of this atomic bid for a given bundle T is then defined as

$$v_{(S,p)}(T) = \begin{cases} p & \text{if } S \subseteq T, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 5.12 (OR/XOR formulas)**.** Let $v, u : 2^G \to \mathbb{N}$ be valuation functions. Given a bundle T, we define valuation functions (v XOR u) and (v OR u) as

$$\begin{aligned} (v \text{ XOR } u)(T) &= \max(v(T), u(T)), \\ (v \text{ OR } u)(T) &= \max_{\substack{R,S \subseteq T \\ \text{s.t. } R \cap S = \emptyset}} v(R) + u(S). \end{aligned}$$

An atomic bid (S, p) simply indicates that an agent is willing to pay a price p as long as he receives all items in S. We can combine these atomic bids using OR and XOR operators. The XOR operator expresses that an agent can obtain at most one of the bids. The OR operator on the other hand indicates that an agent can obtain multiple bids for the sum of their respective prices, as long as the bids are disjoint. Thus, any two bids with overlap in the items cannot simultaneously be satisfied.

Example 5.13. Suppose we are given valuation functions u and v with

$$\begin{aligned} v &= (\{a\}, 3) \text{ XOR } (\{b\}, 4) \text{ XOR } (\{a, b\}, 5), \\ u &= (\{a\}, 3) \text{ OR } (\{b\}, 4) \text{ OR } (\{a, b\}, 5). \end{aligned}$$

In the XOR case, we can only satisfy at most one bid, so $v(\{a, b\}) = 5$ (the third bid is satisfied). In the OR case, we can satisfy multiple bids, but they must be disjoint. This means that we can either satisfy both the first and second bid (giving a value of 7), or just the third bid (giving a value of 5). So $u(\{a, b\}) = 7$. △

Based on these operators, we can define representation languages. The following two languages have similarities to the conjunctive weighted bids language.

Definition 5.14 (XOR bids)**.** An agent i can specify any number of atomic bids, combining them by XOR operators. In other words, his valuation function v_i is of the form

$$v_i = (S_1, p_1) \text{ XOR } (S_2, p_2) \text{ XOR } \ldots \text{ XOR } (S_r, p_r).$$

Definition 5.15 (OR bids)**.** An agent i can specify any number of atomic bids, combining them by OR operators. In other words, his valuation function v_i is of the form

$$v_i = (S_1, p_1) \text{ OR } (S_2, p_2) \text{ OR } \ldots \text{ OR } (S_r, p_r).$$

If we compare these two languages with the conjunctive weighted bid language as defined in this chapter (see Definition 5.1 and Definition 5.2), we do find that the languages are slightly different. The XOR bids language differs in the sense that at most one bid can be accepted per agent, while in the conjunctive weighted bids language multiple bids per agent can be accepted. The OR bids language on the other hand does allow multiple bids per agent to be accepted, but still differs from the conjunctive weighted bids language, since there can be no overlap between the accepted bids. We illustrate this with the following example.

Example 5.16. Suppose an agent i has specified the bids $(\{a, b\}, 5)$, $(\{b, c\}, 10)$ and $(\{c, d\}, 12)$. According to the XOR bids language, the valuation function would correspond to

$$v_{i,\text{XOR}} = (\{a, b\}, 5) \text{ XOR } (\{b, c\}, 10) \text{ XOR } (\{c, d\}, 12).$$

At most one bid of the agent can be accepted, so $v_{i,\text{XOR}}(\{a, b, c, d\}) = 12$.

In the OR bids language, we would have the following valuation:

$$v_{i,\text{OR}} = (\{a, b\}, 5) \text{ OR } (\{b, c\}, 10) \text{ OR } (\{c, d\}, 12)$$

Here, the agent could be assigned both the $\{a, b\}$ and the $\{c, d\}$ bid, but not also the $\{b, c\}$, since there may not be any overlap in the accepted bids. So the total price he would be willing to pay for everything is $v_{i,\text{OR}}(\{a, b, c, d\}) = 17$.

Lastly, in the conjunctive weighted bids language, his preferences would be expressed as

$$B_i = \{(\{a, b\}, 5), (\{b, c\}, 10), (\{c, d\}, 12)\}.$$

In this language, it is allowed for bids to contain overlap, meaning that all bids can be accepted if the agent gets assigned $\{a, b, c, d\}$. Hence, his total utility according would then be $u_i(\{a, b, c, d\}) = 27$. △

5.2.2.2 Link with weighted independent set

In this section, we show that there is a close intuitive connection between finding the maximum USW in a conjunctive weighted bids fair allocation problem and finding the maximum weighted independent set of a graph. Because this problem is NP-hard, this will show that finding maximum USW in this scenario is intractable as well. The weighted independent set decision problem is given as follows:

WEIGHTED INDEPENDENT SET
Input: An undirected graph $G = (V, E)$ plus a weight for every vertex $w : V \to \mathbb{N}$ and a number k.
Question: Does there exist an independent set $V' \subseteq V$ – i.e., a set of nodes where $\{v, w\} \in E$ implies $v \notin V'$ or $w \notin V'$ – such that $\sum_{v \in V'} w(v) \geq k$?

In order to show the connection between these two problems, we first need to introduce a new concept called a *bid graph*. This is a graph that represents the different bids that the agents specify and their compatibility.

Definition 5.17 (Bid graph)**.** Given a fair division problem in the conjunctive weighted bids language $F = (N, \mathcal{O}, \boldsymbol{B})$, we define the *bid graph of* F as $G_F = (V, E)$ with weights $w : V \to \mathbb{N}$ in the following way:

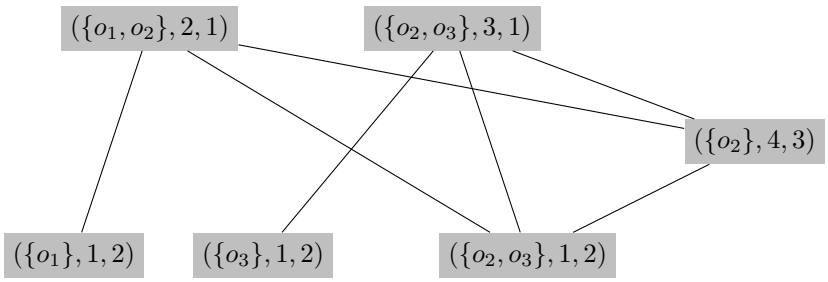
$$V = \{(B_{i,j}, u_{i,j}, i) \mid i \in N \text{ and } (B_{i,j}, u_{i,j}) \in B_i\}$$
$$E = \{\{(B_{i,j}, u_{i,j}, i), (B_{i',j'}, u_{i',j'}, i')\} \mid i \neq i' \text{ and } B_{i,j} \cap B_{i',j'} \neq \emptyset\}$$
$$w((B_{i,j}, u_{i,j}, i)) = u_{i,j}$$

Intuitively, this bid graph has a vertex for every bid that an agent specifies. Regarding the edges, we connect two vertices if their bids belong to different agents and contain some overlap in the items. The idea is that two bids are connected by an edge if they cannot both be satisfied. This is the case if the bids overlap in the items they contain and belong to different agents. Finally, every vertex gets the weight of the utility of the bid it represents.

Example 5.18. Consider a fair division problem in the conjunctive weighted bids language with three agents $N = \{1, 2, 3\}$ and three items $\mathcal{O} = \{o_1, o_2, o_3\}$. The agents' bids are as follows:

$$B_1 = \{(\{o_1, o_2\}, 2), (\{o_2, o_3\}, 3)\}$$
$$B_2 = \{(\{o_1\}, 1), (\{o_3\}, 1), \{o_2, o_3\}, 1)\}$$
$$B_3 = \{(\{o_2\}, 4)\}$$

The bid graph corresponding to these preferences is then the following:



△

We claim that there is a close correspondence between the maximum utilitarian welfare in a fair division problem and finding the maximum weighted independent set in the corresponding bid graph.

Lemma 5.19. A weighted bids fair division problem F has an allocation with utilitarian welfare k if and only if bid graph G_F has a weighted independent set with sum of weights k.

Proof. ($\Rightarrow$) Suppose G_F has a weighted independent set with sum of weights k. This means that there exists $V' \subseteq V$ with $\sum_{v \in V'} w(v) = k$. We now define an allocation π where we allocate exactly those bids to the agents. Thus,

$$\pi(i) = \bigcup_{\substack{(B_{i',j'}, u_{i',j'}, i') \in V' \\ \text{s.t. } i = i'}} B_{i',j'}.$$

We never assign an item to multiple different agents. This is because for any two different bids (of different agents) that are incompatible – i.e., contain overlap in the items – there exists an edge in G_F. As V' is an independent set of this graph, only one of these two bids can be in V'.

Furthermore, we have by construction that

$$u_i(\pi(i)) = \sum_{\substack{(B_{i',j'}, u_{i',j'}, i') \in V' \\ \text{s.t. } i = i'}} u_{i',j'}.$$

Hence,

$$\text{USW}(\pi) = \sum_{i' \in N} \left(\sum_{\substack{(B_{i',j'}, u_{i',j'}, i') \in V' \\ \text{s.t. } i = i'}} u_{i',j'} \right) = \sum_{(B_{i',j'}, u_{i',j'}, i') \in V'} u_{i',j'} = \sum_{v \in V'} w(v) = k,$$

as desired.

($\Leftarrow$) Suppose π is an allocation for fair division problem $(N, \mathcal{O}, \boldsymbol{B})$ with $USW(\pi) = \sum_{i \in N} u_i(\pi(i)) = k$. We define

$$V' = \{(B_{i,j}, u_{i,j}, i) \in V \mid B_{i,j} \subseteq \pi(i)\}.$$

We claim first of all that V' is an independent set of G. Suppose $(B_{i,j}, u_{i,j}, i)$ and $(B_{i',j'}, u_{i',j'}, i')$ are connected by an edge in G, then $i \neq i'$ and $B_{i,j} \cap B_{i',j'} \neq \emptyset$. If both were in V', then $B_{i,j} \subseteq \pi(i)$ and $B_{i',j} \subseteq \pi(i')$, implying that $\pi(i) \cap \pi(i') \neq \emptyset$. This is in contradiction with the definition of an allocation.

Secondly, we have that

$$u_i(\pi(i)) = \sum_{\substack{(B_{i,j}, u_{i,j}) \in B_i \\ \text{s.t. } B_{i,j} \subseteq \pi(i)}} u_{i,j} = \sum_{\substack{(B_{i',j'}, u_{i',j'}, i') \in V' \\ \text{s.t. } i = i'}} u_{i',j'}.$$

Hence,

$$\sum_{v \in V'} w(v) = \sum_{(B_{i,j}, u_{i,j}, i) \in V'} u_{i,j} = \sum_{i \in N} u_i(\pi(i)) = k.$$

□

We can now use the correspondence that we proved in Lemma 5.19 to reduce these two problems to each other.

Theorem 5.20. USW $\leq_p$ WEIGHTED INDEPENDENT SET

Proof. Let $F = (N, \mathcal{O}, \boldsymbol{B})$ be a conjunctive weighted bids fair division problem. Suppose we want to decide whether there exists an allocation with a utilitarian welfare of k.

Given F, we construct the bid graph G_F as in Definition 5.17. This can be done in polynomial time, since we can simply find the edges by looping over all pairs of bids and checking whether they have overlap in their items.

Because of Lemma 5.19, F has an allocation with utilitarian welfare k if and only if G_F has a weighted independent set of k. □

Theorem 5.21. WEIGHTED INDEPENDENT SET $\leq_p$ USW

Proof. For this reduction, we make use of an idea from the hardness proof in Loker and Larson (2010) (more specifically Proposition 3.0.5), where they consider a similar but slightly different representation language for combinatorial auctions.

Suppose that we are given a graph $G = (V, E)$ and a weight function $w : V \to \mathbb{N}$. We construct the following fair division problem $F = (N, \mathcal{O}, \boldsymbol{B})$. The idea is that we create an agent for every vertex in the graph and an item for every edge in the graph. Each agent then has a single bid containing the edges connected to the vertex of that agent.

We let $N = V$, $\mathcal{O} = E$ and define for each $v \in V (= N)$ a single bid

$$B_v = \{(N(v), w(v))\},$$

where $N(v) = \{e \in E \mid v \in e\}$ are the edges incident to v.

We claim that this fair division problem F contains an allocation with USW k if and only if G has a weighted independent set with sum of weights k. We do this by proving that the bid graph G_F of this fair division problem is isomorphic to G. Combining this with the correspondence in Lemma 5.19, this then proves our claim.

Let $G = (V, E)$ be the original graph and $G_F = (V_F, E_F)$ be the bid graph of F as defined in Definition 5.17. Following this definition we have that $V_F = \{(N(v), w(v), v) \mid v \in V\}$. For our isomorphism, we define $f : V \to V_F$ by $f(v) = (N(v), w(v), v)$. It is easy to see that this map is a bijection.

Now suppose that edge $\{v_1, v_2\} \in E$. This means that $\{v_1, v_2\} \in N(v_1)$ and $\{v_1, v_2\} \in N(v_2)$ and thus $N(v_1) \cap N(v_2) \neq \emptyset$ with $v_1 \neq v_2$. This again implies by definition that $\{f(v_1), f(v_2)\} = \{(N(v_1), w(v_1), v_1), N(v_2), w(v_2), v_2\} \in E_F$.

Conversely, suppose that $\{f(v_1), f(v_2)\} = \{(N(v_1), w(v_1), v_1), N(v_2), w(v_2), v_2\} \in E_F$. Then we must have that $v_1 \neq v_2$ and $N(v_1) \cap N(v_2) \neq \emptyset$. The only way for this to occur is if $\{v_1, v_2\} \in N(v_1) \cap N(v_2)$, so we have that $\{v_1, v_2\} \in E$. Hence, f is indeed an isomorphism.

So given a graph G we construct fair division problem F. Because of Lemma 5.19, F has an allocation with utilitarian welfare k if and only if G_F has a weighted independent set of sum k. As we have shown that G is isomorphic with

G_F, this again is the case if and only if G has a weighted independent set of sum k, which concludes our proof. □

Note that Theorem 5.20 is technically not needed if we have already proved Theorem 5.21. Combining Theorem 5.20 with the fact that WEIGHTED INDEPENDENT SET is NP-hard, we can conclude that USW is NP-hard, meaning that every problem in NP can be reduced to USW. Now, because WEIGHTED INDEPENDENT SET is also in NP, this means that there must exist a polynomial-time reduction from USW to WEIGHTED INDEPENDENT SET.

However, the reason we proved Theorem 5.20 separately is that it becomes apparent from the proof of this theorem that there actually is an intuitive and easy to understand reduction between these two problems.

Corollary 5.22. USW for conjunctive weighted bids is computationally equivalent under polynomial time reductions to WEIGHTED INDEPENDENT SET

This correspondence also allows us to show that maximizing utilitarian welfare for conjunctive weighted bids is intractable.

Corollary 5.23. USW for conjunctive weighted bids is NP-complete, even if all weights are equal to 1.

Proof. INDEPENDENT SET (the unweighted variant of WEIGHTED INDEPENDENT SET, i.e., all weights are 1) is known to be NP-hard (Gary and Johnson, 1979, p. 194). Theorem 5.21 gives us a polynomial-time reduction to transform any weighted independent set problem of a graph G into a conjunctive weighted bids fair division problem F.

Note that this reduction is such that if all weights of G are 1, then all of the conjunctive bids in F also have weight 1. Hence, USW for conjunctive weighted bids with all weights equal to 1 is NP-hard. □

5.2.2.3 Restricting the number of agents

As shown in the previous section, the general problem of maximizing utilitarian welfare is intractable. However, in the reduction of Theorem 5.21, the number of agents scales linearly with the number of vertices of the graph we want to find a weighted independent set in. Perhaps the problem becomes easier to solve if we have a restricted number of agents. Our first result shows that this is indeed the case for the scenario of exactly two agents.

Theorem 5.24. MAX USW for the conjunctive weighted bids language with two agents can be solved in polynomial time.

Proof. First of all, because of Lemma 5.19, in order to find the maximum utilitarian social welfare of a fair division problem $F = (N, \mathcal{O}, \boldsymbol{B})$, it suffices to construct the bid graph G_F as defined in Definition 5.17 and find the maximum weighted independent set of this graph.

In general, finding the maximum weighted independent set of a graph is NP-complete. However, we will show that in the case of two agents, G_F is a bipartite graph. For these classes of graphs it is possible to find the maximum weighted

independent set in polynomial time, so this allows us to construct a polynomial algorithm to find the maximum utilitarian welfare (Zhang et al., 2018; Kumar, 2003).

What is left to prove is that G_F is bipartite. If we call the two agents i_1 and i_2, then $V_{i_1} := \{(B_{i,j}, u_{i,j}) \in V \mid i = i_1\}$ and $V_{i_2} := \{(B_{i,j}, u_{i,j}) \in V \mid i = i_2\}$ form a partition of the graph. Moreover, there are no edges between vertices in V_{i_1} (and similarly for V_{i_2}), since there can only be an edge between two vertices in the bid graph if the bids are from different agents. Thus, G is bipartite. □

Thus, for two agents, calculating the maximum utilitarian welfare is possible in polynomial time. This of course raises the question of whether we can generalize this any further. Is it also possible to solve this fair division problem for a higher number of agents n if we keep n fixed? As the following result will show us, this is not the case. Starting from $n = 3$, the USW decision problem already becomes intractable.

Theorem 5.25. When the number of agents is fixed to three ($n = 3$), USW for the conjunctive weighted bids language is NP-complete.

To prove this theorem, we will reduce from the following problem.

> INDEPENDENT SET BOUNDED DEGREE 3
> *Input:* An undirected graph $G = (V, E)$ such that every node has degree at most 3 – i.e., $|N(v)| \leq 3$ - and an integer k.
> *Question:* Does there exist an independent set $V' \subseteq V$ – i.e., a set of nodes such that $\{v, w\} \in E$ implies $v \notin V'$ or $w \notin V'$ – of size at least k?

The decision problem INDEPENDENT SET BOUNDED DEGREE 3 is known to be NP-complete (Garey et al., 1976)[1]. Before we can reduce this problem to USW with $n = 3$, we first need the following graph-theoretic result.

Lemma 5.26. If G is a graph of bounded degree 3, then either we can find a 3-coloring of G or a maximum independent set of G in polynomial time.

Proof. Let G be a graph with bounded degree 3. We make use of the following theorem from the literature.

Theorem 5.27 (Brooks (1941))**.** For any graph G with degree Δ, there exists a Δ-coloring unless G is a complete graph or an odd cycle graph.

The proof of this theorem also gives us a polynomial-time procedure to find this Δ-coloring in case it exists (see also Zając, 2018 for a linear time algorithm). So we can conclude from this theorem that if G is not an odd cycle graph or a complete graph, then we can find a 3-coloring in polynomial time.

Suppose that G is an odd cycle graph. We can also find a 3-coloring in this case. Assume that the nodes are in the order of the cycle $v_1, v_2, \ldots, v_{2n+1}$. We color every node v_i with i even blue and every node v_i with i odd red, except for the

[1]Garey et al. (1976) show that the vertex cover problem with bounded degree 3 is NP-complete. It is however very easy to reduce the vertex cover to the independent set problem by taking the same graph G and setting $k' := |V| - k$.

last node v_{2n+1}. Neighbor v_1 is colored red and neighbor v_{2n} blue, so we color v_{2n+1} green. This defines a valid 3-coloring.

Finally, consider the case where G is a complete graph. If G is a complete graph with 3 or fewer nodes, it is easy to see that we can find a 3-coloring in polynomial time. The only other complete graph where every node has bounded degree 3 is the complete graph with 4 nodes. In this case, there is no 3-coloring possible. However, the only independent sets of this graph are the four sets consisting of a single node, so it is easy to compute a maximum independent set in this case. □

Now we are ready to define the reduction and prove its correctness.

Proof of Theorem 5.25. Let G be a graph with bounded degree 3 and k an integer. We want to construct a fair division problem F in the conjunctive bids language with $n = 3$ such that F has an allocation with utilitarian welfare k if and only if G has an independent set of size k. If we have such a construction, then we can solve an instance of INDEPENDENT SET BOUNDED DEGREE 3 using an algorithm to solve USW for conjunctive weighted bids and 3 agents, meaning that this problem is NP-hard.

Our reduction works in the following way. Given a graph $G = (V, E)$ with bounded degree 3 and integer k, we apply Lemma 5.26. Then, either we have found a 3-coloring of G or we have found the maximum independent set of G. In the latter case, we are already done.

So assume we have a 3-coloring of G and let $c : V \to \{1, 2, 3\}$ be the mapping of nodes to their respective color. Now split the nodes in three partitions depending on their color V_1, V_2, V_3 – i.e., for every $v \in V$, $v \in V_{c(i)}$. For any two nodes $v, v' \in V_i$ for $i \in \{1, 2, 3\}$ we have that $\{v, v'\} \notin E$. We now define the following fair division problem $F = (N, \mathcal{O}, \boldsymbol{B})$:

$$\begin{aligned} N &= \{1, 2, 3\} \\ \mathcal{O} &= E \\ B_{V_i} &= \{(N(v), 1) \mid v \in V_i\} \text{ for } i \in \{1, 2, 3\} \end{aligned}$$

We claim that the bid graph G_F is isomorphic to G. To show this, consider the map $f : V \to V_F$ with $f(v) = (N(v), 1, c(v))$. It is clear that this map is a bijection.

To show that this map is an isomorphism, let $\{v, v'\} \in E$. Because two nodes of the same color cannot be connected, we must have that $c(v) \neq c(v')$. As $\{v, v'\} \in E$, we have that both $\{v, v'\} \in N(v)$ and $\{v, v'\} \in N(v')$, which shows that $N(v) \cap N(v') \neq \emptyset$. Hence, $(f(v), f(v')) = ((N(v), 1, c(v)), (N(v'), 1, c(v')) \in E_F$.

Conversely, suppose that $(f(v), f(v')) = ((N(v), 1, c(v)), (N(v'), 1, c(v')) \in E_F$. Then this implies that $N(v) \cap N(v') \neq \emptyset$, which can only be the case if $\{v, v'\} \in N(v) \cap N(v')$. Hence, $\{v, v'\} \in E$. So f is indeed an isomorphism.

This means that to find out whether G has an independent set of size k, it suffices to check whether G_F has an independent set of size k. By Lemma 5.19,

this is the case precisely when F contains an allocation with utilitarian welfare k. This concludes the proof of the reduction. □

Corollary 5.28. Let n be the number of agents. For any fixed $n \geq 3$, USW for the conjunctive weighted bids language is NP-complete.

5.3 Disjunctive weighted bids

Disjunctive bids are easier to satisfy than conjunctive bids. With conjunctive bids, all items of the bid need to be assigned to the agent for the bid to become satisfied, while with disjunctive bids, a bid is already satisfied if at least one of the items of the bid is inside the agents' bundle.

We can ask ourselves the same questions about the computational complexity of computing optimal solutions as we did for conjunctive bids. However, it turns out that for all solution concepts (USW, ESW, NSW) we consider, finding the allocation elected by this solution concept is NP-hard, even in a very restricted setting already.

To show, this we will first introduce an auxiliary decision problem OPT EXISTENCE. A disjunctive weighted bids problem $\langle N, \mathcal{O}, \boldsymbol{B} \rangle$ is in this set if there exists an allocation such that all bids are satisfied for every agent. More formally,

$$\text{OPT EXISTENCE} = \{\langle N, \mathcal{O}, \boldsymbol{B} \rangle \mid \exists \pi \text{ such that } u_i(\pi(i)) = \sum_{(B_{i,j}, u_{i,j}) \in B_i} u_{i,j} \ \forall i \in N\}.$$

We will first of all prove that this decision problem is already NP-hard, even in the unweighted case with just two agents and identical bids. Secondly, we show that we can decide OPT EXISTENCE efficiently given an algorithm for USW, ESW, NSW or EEF, which proves NP-hardness for these problems as well.

Lemma 5.29. OPT EXISTENCE is NP-hard, even if all weights are equal to 1, there are two agents and the agents' bids are identical.

Proof. To show hardness, we will reduce from SET SPLITTING, which is known to be NP-hard (Gary and Johnson, 1979, p. 221). We use a similar technique as in Theorem 3.4.

> SET SPLITTING
> *Input:* A set S and a collection $C = \{S_1, \ldots, S_c\}$ of subsets of S.
> *Question:* Does there exist a partition of S into S_a and S_b such that no subset in C is completely in either S_a or S_b?

Given an input instance $\langle S, C \rangle$ for SET SPLITTING, we show a polynomial-time computable construction for a disjunctive weighted bids fair division problem F such that $\langle S, C \rangle \in$ SET SPLITTING if and only if $F \in$ OPT EXISTENCE.

Define $F = \langle \{1, 2\}, S, (B_1, B_2) \rangle$, with

$$B_1 = B_2 = \{(S_i, 1) \mid S_i \in C\}.$$

That is, we create a fair division problem with two agents and specify a bid of weight 1 for every subset $S_i \in C$. We claim that $\langle S, C \rangle \in$ SET SPLITTING if and only if $F \in$ OPT EXISTENCE

Suppose there exists a set splitting in S_a and S_b. Then choose the allocation π with $\pi(1) := S_a$ and $\pi(2) := S_b$. As no subset in C is completely in S_a or S_b, for every $S_i \in C$ at least one of these items is in S_a and one of the items is in S_b. Hence, every bid $(S_i, 1)$ is satisfied for both agents, which implies that $u_i(\pi(i)) = |B_i|$.

Conversely, suppose there exists an allocation π with $u_i(\pi(1)) = |B_1|$ and $u_i(\pi(2)) = |B_2|$. We claim that choosing $S_a := \pi(1)$ and $S_b = \pi(2)$ is a set splitting. Let $S_i \in C$ be arbitrary. We know that every bid in B_1 and B_2 is satisfied, so in particular the bid $(S_i, 1)$ must be satisfied for both agent 1 and 2. This can only be the case if $S_i \cap \pi(1) \neq \emptyset$ and $S_i \cap \pi(2) \neq \emptyset$. Thus, there must be both an item of S_i in S_a and in S_b. □

Now that we have proved that OPT EXISTENCE is NP-hard, we can show that computing the allocations elected by the solution concepts USW, ESW or NSW is NP-hard as well, by reducing OPT EXISTENCE to these decision problems.

Corollary 5.30. USW for weighted disjunctive bids with all weights equal to 1, two agents and identical bids is NP-complete.

Proof. Let n be the number of bids of a single agent in a two-agent weighted disjunctive bids problem F with two agents and identical bids having utility 1. Because the bids of the agents are identical, both agents have n bids. Now note that an allocation satisfies all bids of both agents precisely if the utilitarian welfare of this allocation is $2n$. Hence, we can reduce OPT EXISTENCE to USW. □

Corollary 5.31. ESW for weighted disjunctive bids with all weights equal to 1, two agents and identical bids is NP-hard.

Proof. Let n be the number of bids of a single agent, then the other agent also has n bids (all having weight 1). So an allocation here satisfies all bids if and only if the egalitarian welfare of this allocation is n. Hence, we can reduce OPT EXISTENCE to ESW. □

Corollary 5.32. NSW for weighted disjunctive bids with all weights equal to 1, two agents and identical bids is NP-hard.

Proof. If n is again the number of bids of a single agent, we claim that an allocation satisfies all bids if and only if the allocation has Nash welfare $n \cdot n$.

First of all, if an allocation π satisfies all bids, then both agents have utility n and so the Nash welfare is $n \cdot n$. Conversely, suppose that π' is an allocation with Nash welfare $n \cdot n$. We have by definition that $u_1(\pi(1)) \cdot u_2(\pi(2)) = n \cdot n$. Note that this can only hold if either both agents have utility n or one of the two agents has utility strictly larger than n. The latter cannot be the case, since at best all bids of an agent are satisfied, yielding him a utility of n. More utility is not possible. Hence, both agents must have utility n and thus all their bids are satisfied. This allows us to reduce OPT EXISTENCE to NSW. □

We can conclude that for weighted disjunctive bids fair division problems in this version there does not exist an efficient algorithm that finds an optimal allocation (unless P=NP). As the hardness result already holds for two agents and unweighted bids, restricting the number of agents or the possible weights will not help us in finding a tractable subclass.

5.3.1 Connection to the dichotomous language

Although it might not seem clear at first, there is a close relation between the disjunctive weighted bids language and the dichotomous language. If the propositional formulas of the agents are positive (i.e., contain no negative literals) and are given in conjunctive normal form (CNF), then we can encode these preferences in the disjunctive weighted bids language.

The idea for this is the following. A formula in CNF consists of conjunctions of clauses. For each clause in the formula, we specify a disjunctive bid where the items in the bid are precisely the items occurring in that clause. Using this approach, we can reduce the OPT EXISTENCE problem – i.e., does there exist an allocation such that the formula of every agent is satisfied – to a similar problem for disjunctive bids.

Theorem 5.33. Let F be a fair division problem in the dichotomous language, where the propositional formulas are positive and in CNF. Then we can construct an 'equivalent' problem F' in the disjunctive weighted bids language, with all bids equal to 1.
That is, every agent in F can be satisfied completely if and only if every agent in F' can be satisfied completely and all agents have the same utility.

Proof sketch. Let $F = (N, \mathcal{O}, \boldsymbol{u})$ be such a fair division problem in the dichotomous language, with φ_i indicating the formula of agent i for each $i \in N$.

Because φ_i is in CNF and positive, φ_i is of the form

$$(o_{1,1} \vee \cdots \vee o_{1,l_1}) \wedge (o_{2,1} \vee \cdots \vee o_{2,l_2}) \cdots \wedge (o_{c,1} \vee \cdots \vee o_{c,l_c}).$$

where c is the total number of clauses and l_j is the number of literals in the jth clause. Furthermore, let $c_{\max}$ be the maximum number of clauses of a formula among all φ_i.

We construct a fair division problem $F' = (N', \mathcal{O}', \boldsymbol{u}')$, where $N = N'$, $\mathcal{O}' = \mathcal{O} \cup \mathcal{O}_{\text{dummy}}$ and agent i has the following disjunctive bids:

$$B_i = \{(\{o_{1,1}, \ldots, o_{1,l_1}\}, 1), (\{o_{2,1}, \ldots, o_{2,l_2}\}, 1), \ldots, (\{o_{c,1}, \ldots, o_{c,l_c}\}, 1)\} \cup B_{i,\text{dummy}}$$

Here, $B_{i,\text{dummy}}$ indicates a set of $c_{\max} - c$ dummy bids all having weight 1 that can always trivially be satisfied. The idea behind this is that we pad all the bids with extra dummy bids so that all agents have exactly $c_{\max}$ utility if all their bids are satisfied. We can do this by adding $c_{\max} - c$ extra items to $\mathcal{O}_{\text{dummy}}$ that only agent i wants and adding $c_{\max} - c$ bids of the form $(\{o_{\text{dummy},j}\}, 1)$.

Now suppose there exists an allocation π of F such that every formula φ_i is satisfied. For an agent i, this means that he received at least one item of every clause in φ_i, so if we let allocation $\pi' = \pi$ for F', then every of the c 'real' bids

in B_i are satisfied. We can moreover easily satisfy all the dummy bids $B_{i,\text{dummy}}$. So in total all the bids of all agents are satisfied and they all have utility $c_{\max}$.

Conversely, if there exists an allocation π' in F' such that all bids of all agents are satisfied, then this implies that every agent i receives at least one item of every clause in φ_i. Hence, if we set $\pi = \pi'$ (and get rid of the dummy items), we have an allocation π that satisfies all formulas of all agents. □

We can use this relation to prove that we can solve the OPT EXISTENCE problem for (a subset of) the dichotomous language using various solution concepts for the disjunctive weighted bids language.

Corollary 5.34. If there exists a polynomial-time algorithm for USW, ESW or NSW for the disjunctive weighted bids language with all weights equal to 1, there also exists a polynomial-time algorithm for OPT EXISTENCE for the dichotomous language with formulas that are positive and in CNF.

Proof. Let F be a fair division problem in the dichotomous language, where the propositional formulas are positive and in CNF. By Theorem 5.33, we can construct a weighted disjunctive bids fair division problem F' such that it suffices to check whether there exists an allocation in F' that satisfies every agent completely to solve OPT EXISTENCE for F.

Note that an agent is completely satisfied in F' if and only if he has utility $c_{\max}$ and note furthermore that an agent can never receive more utility than this. We can thus check whether every agent is satisfied completely by one of the following:

- Checking whether there exists an allocation π with $\text{USW}(\pi) \geq n \cdot c_{\max}$;
- Checking whether there exists an allocation π with $\text{ESW}(\pi) \geq c_{\max}$;
- Checking whether there exists an allocation π with $\text{NSW}(\pi) \geq c_{\max}^n$. [2]

□

[2] To obtain a Nash welfare of at least $c_{\max}^n$, either every agent should receive utility $c_{\max}$, or there should be at least one agent with utility $> c_{\max}$. The latter case is not possible, so the former case must hold.

Chapter 6

The partition language

We have seen in the previous chapter that allowing agents to specify disjunctive bids – the case where an agent receives utility if his allocation contains at least one item from the bid – quickly makes the language NP-hard. Both maximizing utilitarian social welfare and maximizing egalitarian social welfare is NP-hard, even if there are only two agents, the bids are not weighted and we use only positive literals. In order to find a language in which we can express such disjunctive preferences such that this language is tractable for these solution concepts, we need to look at different restrictions.

The restriction that we will look at in this chapter is requiring that the disjunctive bids from the same agent have no overlap in the items occurring in them. We will show that such a restriction does allow us to calculate an allocation elected by a solution concept such us utilitarian welfare efficiently.

6.1 Definitions

First of all, we need to define this language formally. Instead of letting agents specify bids and requiring that they do not overlap, we define a slightly different preference language, which we will show is actually more general. We let agents specify a partition of all the items and let them furthermore specify their utility for every item. The idea is that agents will only receive the utility of one item from each equivalence class.

Definition 6.1 (Disjunctive partition language)**.** A *disjunctive partition language* consists of a set of a set of utility functions $\boldsymbol{u} = (u_1, \ldots, u_n)$ and a set of equivalence relations $\sim = (\sim_1, \ldots, \sim_n)$. Each $u_i : \mathcal{O} \to \mathbb{N}$ indicates the utility of every item according to agent i. Additionally, $\sim_i$ is an equivalence relation on the set of items $\mathcal{O}$, representing which items are similar for agent i.

When calculating the utility of an agent's bundle, for every equivalence class we only count the utility of one item – the one with highest utility. This leads to the following definition of the utility of an agent's bundle. Note that here (and later throughout this chapter), we use the notation $\mathcal{O}/\sim_i$ to denote the set of all equivalence classes according to equivalence relation $\sim_i$. We furthermore use

the notation $[o]_{\sim_i}$, where $o \in \mathcal{O}$, to refer to the equivalence class according to $\sim_i$ that contains o.

Definition 6.2 (Utility of disjunctive partition language)**.** Given a disjunctive partition language $(\boldsymbol{u}, \boldsymbol{\sim})$, we define the utility of an agent i given a bundle $S \subseteq \mathcal{O}$ as

$$u_i(S) = \sum_{O \in \mathcal{O}/\sim_i} \max_{o \in O \cap S} u_i(o).$$

A different way in which we might also specify the agent's preferences equivalently is by restricting the possible allocations. This leads to the following alternative definition of the utility of an agent's bundle.

Definition 6.3 (Utility of disjunctive partition language (alternative))**.** Given a disjunctive partition language $(\boldsymbol{u}, \boldsymbol{\sim})$, we define the utility of an agent i given a bundle $S \subseteq \mathcal{O}$ as

$$u_i(S) = \sum_{o \in S} u_i(o),$$

where S can contain at most one object out of each equivalence class of $\sim_i$. More formally, we require that S is such that

$$\forall o \in \mathcal{O} : |[o]_i \cap S| \leq 1.$$

We will now show that these two ways to define the utility for the disjunctive partition language are actually equivalent.

Proposition 6.4. There exists an allocation π with utilities $u_1, \ldots, u_n$ for the agents respectively according to Definition 6.2 if and only if there exists an allocation π' with utilities $u_1, \ldots, u_n$ for the agents according to Definition 6.3.

Proof sketch. Direction ($\Leftarrow$) is simple: if there exists an allocation π' such that $\forall o \in \mathcal{O} : |[o]_i \cap S| \leq 1$, every agent receives at most a single item from each equivalence class. This implies that the utility of every item is also counted according to Definition 6.2, so the utilities are the same.

For the ($\Rightarrow$) direction, suppose we are given an allocation π. We then create a new allocation π' where we only let him keep, for each equivalence class, the item from that class that his maximum utility for him and unassign all other items from that equivalence class (these items will not be assigned to anyone). Doing so will yield us an allocation π' that satisfies $\forall o \in \mathcal{O} : |[o]_i \cap S| \leq 1$. Since unassigning these items does not change the utility according to Definition 6.2 and all the leftover items now contribute to the utility of an agent's bundle, the utilities are the same according to both definitions. □

There is a close connection between this preference language and the disjunctive weighted bids language (as defined in Definition 5.4) that has the restriction that bids within a single agent are not allowed to overlap. Note that the language presented here is slightly more general: it also allows to express different utilities for items within the same bid.

Proposition 6.5. Any set of preferences in the disjunctive weighted bids language with no overlap in bids can also be expressed in the disjunctive partition language

Proof sketch. Suppose we are given a set of preferences in the disjunctive weighted bids language. Then we are given $\boldsymbol{B} = (B_1, \ldots, B_n)$, witch $B_i = \{(B_{i,1}, u_{i,1}), \ldots, (B_{i,r}, u_{i,r})\}$ representing the set of bids of agent i. Because of the no overlap condition, we have that $j \neq k$ implies that $B_{i,j} \cap B_{i,k} = \emptyset$ for all agents i. Since the bids contain no overlap, we can just define an equivalence relation where every element from the same bid belongs to the same equivalence class. We give every item in the same equivalence class the same utility, namely the weight of the corresponding bid. We put the items that are not in any bid all in the same equivalence class and define the utility of all those items to be 0.

It can be checked that this encoding of the problem in the disjunctive partition language is indeed equivalent to the original preferences as represented in the disjunctive weighted bids language. □

6.2 Basic algorithm

We will now give an algorithm with which we can efficiently find an allocation with maximum utilitarian welfare given a fair division problem in the disjunctive partition language. To this end, we first define the notion of a preference graph.

Definition 6.6 (Preference graph)**.** Given a fair division problem in the disjunctive partition language $(N, \mathcal{O}, \boldsymbol{u}, \sim)$, we define the bipartite *preference graph* $G_p = (V, E)$ with weights $w : E \to \mathbb{N}$ in the following way:

$$\begin{aligned} V &= A \cup B \text{ with } A = \{(i, [o]_i) \mid o \in \mathcal{O}, i \in N\} \text{ and } B = \mathcal{O} \\ E &= \{\langle (i, O), o' \rangle \mid o' \in O\} \\ w(\langle (i, O), o' \rangle) &= u_i(o') \end{aligned}$$

In words, we define a bipartite graph with on one side a vertex for every item $\mathcal{O}$ and on the other side a vertex for every equivalence class of an agent. We connect two of these vertices with an edge if the item occurs in this equivalence class and use the utility of that item as the weight of this edge.

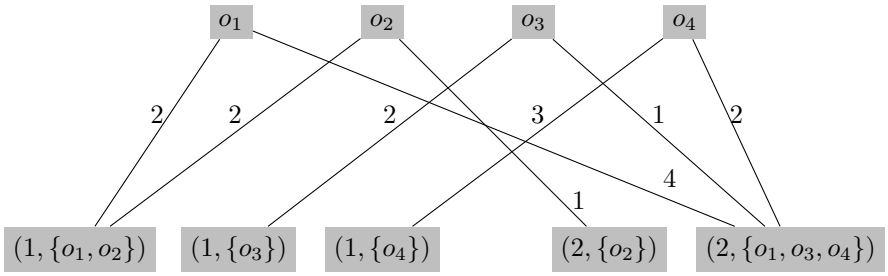
Example 6.7. Consider a fair division problem in the disjunctive partition language with two agents $N = \{1, 2\}$ and four items $\mathcal{O} = \{o_1, o_2, o_3, o_4\}$.
The utility functions of the agents are given as follows:

$$\begin{array}{llll} u_1(o_1) = 2 & u_1(o_2) = 2 & u_1(o_3) = 2 & u_1(o_4) = 3 \\ u_2(o_1) = 4 & u_2(o_2) = 1 & u_2(o_3) = 1 & u_2(o_4) = 2 \end{array}$$

Furthermore, the agents have the following equivalence relations on the items:

$$\begin{aligned} \mathcal{O}/\!\sim_1 &= \{\{o_1, o_2\}, \{o_3\}, \{o_4\}\} \\ \mathcal{O}/\!\sim_2 &= \{\{o_2\}, \{o_1, o_3, o_4\}\} \end{aligned}$$

Then the corresponding preference graph G_p looks like this:



△

Matchings in this preference graph can be interpreted as allocations in the fair division problem. There is a close connection between the weight of such a matching and the utilitarian welfare of the corresponding allocation, which we can use to compute an allocation maximizing utilitarian welfare in polynomial time.

Theorem 6.8. There exists a weighted bipartite matching in G_p of size k if and only if there exists an allocation with utilitarian social welfare k.

Proof. Suppose there exists a weighted bipartite matching $E' \subseteq E$ with $\sum_{e \in E'} w(e) \geq k$. We now define allocation π in the following way:

$$\pi(i) = \{o \in \mathcal{O} \mid \langle (i, [o]_i), o \rangle \in E'\}$$

First of all, each item $o \in \mathcal{O}$ is allocated to only one agent. Since E' is a matching, there can be at most one edge to any node $o \in \mathcal{O}$. Moreover, every agent i has at most one item of each of his equivalence classes $[o]_i$, again since E' is a matching, which makes π a valid allocation. Finally, $\sum_{i \in N} u_i(\pi(i)) = \sum_{e \in E'} w(e) = k$. Conversely, suppose we are given an allocation π with $\sum_{i \in N} u_i(\pi(i)) = k$. We define the following matching:

$$E' = \{\langle (i, O), o' \rangle \in E \mid o' \in \pi(i)\}$$

E' is a valid matching, since (1) every item is allocated to only one agent and hence there can be no two different edges in E' ending in the same node $o \in \mathcal{O}$ and (2) every agent only receives at most one item from each equivalence class, meaning that there cannot be two edges ending in the same node in $\{(i, [o]_i) \mid o \in \mathcal{O}, i \in N\}$. Note that $\sum_{e \in E'} w(e) = \sum_{i \in N} u_i(\pi(i)) = k$. □

Corollary 6.9. MAX USW for the disjunctive partition language is computable in polynomial time.

Proof sketch. By Theorem 6.8, we can reduce the problem of finding the maximum utilitarian welfare of such a fair allocation problem to finding a maximum weight bipartite matching in the corresponding preference graph. Calculating such a matching can be done in polynomial time using the Hungarian algorithm (Munkres, 1957). □

We have now shown that we can compute the maximum utilitarian welfare of the disjunctive partition language in polynomial time. One criticism of using the solution concept utilitarian welfare is that this solution concept does not ensure

fairness in the elected allocation in any way (e.g. it might assign all items to a single agent).

However, optimizing for utilitarian welfare in the disjunctive partition language will most likely not lead to a single agent receiving all the items. This is because the preferences that agents express in this language are subadditive. If an agent receives multiple items from the same partition, only one of these items contributes to his utility. If we instead were to split up these items among different agents, then all these items might contribute to the utility of the agent they are assigned to and thus doing this often leads to a higher utilitarian welfare.

So, in the disjunctive partition language, by optimizing utilitarian welfare we in a way already implicitly incentivize to not assign all items to one agent, but instead divide them over multiple agents. This of course does not mean that every agent will always receive a fair share; there is still no guarantee in fairness when optimizing for utilitarian welfare and unfair situations like this might still happen. But this at least makes it a little bit more interesting to study utilitarian welfare optimization algorithms for the disjunctive partition language.

In Chapter 7, we will study the solution concepts leximin and Nash welfare, which provide more fairness guarantees, with respect to the disjunctive partition language. But to find tractable algorithms for these solution concepts, we need to restrict the disjunctive partition language further.

6.3 The extended language

We know from the previous section that maximizing utilitarian welfare for the disjunctive partition language can be done in polynomial time. This raises the question of whether we can generalize this model even further, so that more complicated preferences can also be expressed. In this section, we investigate such extensions.

We first define the extended language and prove that we can indeed calculate the maximum utilitarian welfare in polynomial time. It might however not be easy to see what kind of desirable properties can (or cannot) be expressed in this extended language. Therefore, we also provide a few examples of complicated forms of preferences that can be encoded in the extended disjunctive partition language.

Definition 6.10 (Extended disjunctive partition language)**.** An expression in the *extended disjunctive partition language* consists of a tuple $(u_i, \sim_i, c_i, m_i)$ for each agent $i \in N$ with:

- a function $u_i : \mathcal{O} \to \mathbb{N}$ indicating the utility of every item,
- an equivalence relation $\sim_i$ on the set of items $\mathcal{O}$, representing which items are similar,
- a function $c_i : \mathcal{O}/\sim_i \to \mathcal{N}$ on the equivalence classes indicating how many items from the same equivalence class we can pick,
- a function $m_i : \mathcal{O}/\sim_i \times \mathcal{O}/\sim_i \to \mathbb{N}$, where $m_i(O, O')$ represents the extra utility for agent i if no item from O or O' is allocated to him, with the

restriction that $m_i(O, O') \neq 0$ implies that:

- $c_i(O) = c_i(O') = 1$,
- there is no other $O'' \in \mathcal{O}/\sim_i$ such that $m_i(O', O'') \neq 0$ or $m_i(O, O'') \neq 0$.

Definition 6.11 (Utility of extended disjunctive partition language)**.** Given a disjunctive partition language $(\boldsymbol{u}, \boldsymbol{\sim}, \boldsymbol{c}, \boldsymbol{m})$, we define the utility of an agent i given a bundle $S \subseteq \mathcal{O}$ as

$$u_i(S) = \sum_{o \in S} u_i(o) + \sum_{\substack{O, O' \in \mathcal{O}/\sim_i \\ \text{s.t. } c_i(O)=c_i(O')=1 \\ \text{and } S \cap O = S \cap O' = \emptyset}} m_i(O, O'),$$

where S can contain at most one object out of each equivalence class of $\sim_i$. More formally, we require that S is such that

$$\forall o \in \mathcal{O} : |[o]_i \cap S| \leq c_i([o]_i).$$

This definition can be seen as an extension of Definition 6.3. Similarly, we also define a preference graph for this language.

Definition 6.12 (Extended preference graph)**.** Given a fair division problem with an extended disjunctive partition language $(N, \mathcal{O}, \boldsymbol{u}, \boldsymbol{\sim}, \boldsymbol{c}, \boldsymbol{m})$, we define the *preference graph* $G_p = (V, E)$ with weights $w : E \to \mathbb{N}$ in the following way:

$$V = A \cup B \text{ with } A = \{(i, j, [o]_i) \mid o \in \mathcal{O}, i \in N, 1 \leq j \leq c_i([o]_i)\} \text{ and } B = \mathcal{O}$$
$$E = \{\langle (i, j, O), o' \rangle \mid o' \in O\} \cup \{\langle (i, 1, O), (i, 1, O') \rangle \mid m_i(O, O') > 0\}$$
$$w(\langle (i, j, O), o' \rangle) = u_i(o')$$
$$w(\langle (i, 1, O), (i, 1, O') \rangle) = m_i(O, O')$$

Intuitively, this graph is very similar to the preference graph for the original partition model. The difference is that there are now $c_i([o]_i)$ nodes per equivalence class of an agent instead of just 1. Furthermore, for any two equivalence classes such that $m_i(O, O') > 0$, there is an extra edge between these two nodes.

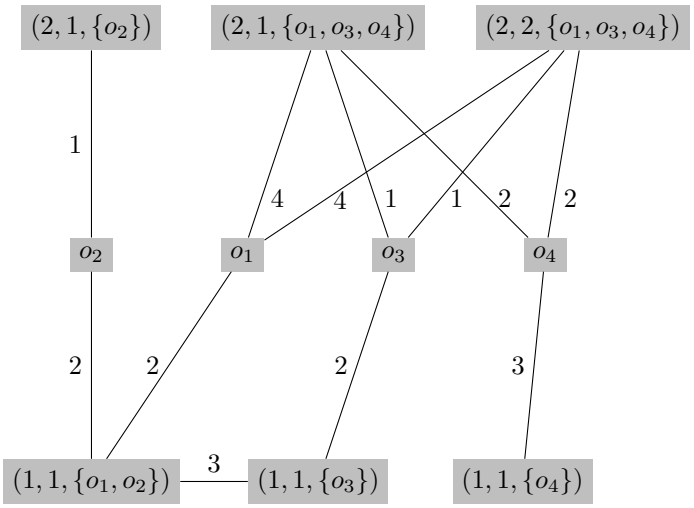
Example 6.13. Consider the same fair division problem as in Example 6.7, but now in the extended language. So we let u_1, u_2, $\mathcal{O}/\sim_1$ and $\mathcal{O}/\sim_2$ be the same as in Example 6.7, but we now also define the following capacities:

$$c_1(\{o_1, o_2\}) = 1 \qquad c_1(\{o_3\}) = 1 \qquad c_1(\{o_4\}) = 1$$
$$c_2(\{o_2\}) = 1 \qquad c_2(\{o_1, o_3, o_4\}) = 2$$

Furthermore, we define $m_i(O, O') = 0$ for all $O, O \in \mathcal{O}/\sim_i$, except we define

$$m_1(\{o_1, o_2\}, \{o_3\}) = 3.$$

Then the corresponding extended preference graph G_p is the following:



△

Theorem 6.14. There exists a weighted matching in G_p of size k for a fair division problem in the extended disjunctive partition language if and only if there exists an allocation with utilitarian social welfare k.

Proof sketch. The general idea of the proof is still the same as in Theorem 6.8. Given a weighted matching $E' \subseteq E$, we define the allocation π with

$$\pi(i) = \{o \in \mathcal{O} \mid \langle (i, j, [o]_i), o\rangle \in E' \text{ for some } j\}.$$

Note that each agent now has at most $c_i([o]_i)$ objects of an equivalence class $[o]_i$ assigned to him, since there are $c_i([o]_i)$ nodes for this equivalence class in G_p.

Furthermore, if an edge $\langle (i, 1, O), (i, 1, O')\rangle$ is part of the matching, then it must be the case that no object from O and no object from O' is assigned to agent i, so this agent receives the bonus utility of $m_i(O, O')$. In this way, we can show that allocation π has the same utilitarian welfare as the weight of matching E'

Conversely, given an allocation π, we define a matching $E' \subseteq E$ consisting of the following:

- For every $o' \in \pi(i)$, we add the edge $\langle (i, j, O), o'\rangle$ to E', where j is the least j for which (i, j, O) is still a free node.
- For every two equivalence classes $O, O' \in O/\sim_i$ with $S \cap O = S \cap O' = \emptyset$ and $c_i(O) = c_i(O') = 1$, we add edge $\langle (i, 1, O), (i, 1, O')\rangle$ to E'.

The weight of matching E' is the same as the utilitarian welfare of π. □

Corollary 6.15. MAX USW for the extended disjunctive partition language is computable in polynomial time.

Proof sketch. By the theorem that we have proved, finding the maximum utilitarian welfare is equivalent to finding a maximum weighted (not necessarily bipartite) matching in G_p. Since we can construct G_p in polynomial time and finding the maximum weighted matching of a general graph is also computable

in polynomial time (see Plummer and Lovász, 1986), computing maximum utilitarian welfare can be done in polynomial time. □

We will now show a few interesting forms of preferences that we can encode with this model.

6.3.1 Allowing multiple items from an equivalence class

The original preference language only allows us to choose a single item from an equivalence class. The idea behind this is that the items within one equivalence class are mutually exclusive. Two (or more) of these 'similar' items will not yield more utility than a single item does. However, we sometimes might want to allow multiple items to be chosen from the same equivalence class, as long as the total number of items does not exceed a certain threshold k. In this case the mutual exclusivity property occurs only from $k+1$ items (instead of from two items already).

The extended preference language allows us to express such preferences. For each equivalence class O of a certain agent i, we can specify that at most k items can be chosen from this equivalence class by setting $c_i(O) = k$.

The way in which utility is defined in Definition 6.11, this upper bound on items within an equivalence class is achieved by putting a restriction on the bundles that are allowed: we require that each bundle contains at most k items from each equivalence class. While defining bundles this way makes it easier to prove the connection with weighted matchings, it might not be the conceptually easiest way to think about these kind of preferences.

A better way to think about these type of preferences is to allow all bundles (also those that have more than k items from an equivalence class), but only count the utilities of the k best items of an equivalence class when calculating the utility of a bundle. To define this properly, we need to introduce a bit more notation. First of all, let us define $\max^{(k)}(X)$ as the sum of the k largest items in X. That is,

$$\max^{(k)}(X) = \max_{\substack{X' \subseteq X \\ |X'|=k}} \sum_{x \in X'} x.$$

Then, given a bundle S, we utility of that bundle for agent i is defined as

$$u_i(S) = \sum_{O \in \mathcal{O}/\sim_i} \max^{(c_i(O))}(O \cap S).$$

This definition is more in line with Definition 6.2 for the (non-extended) disjunctive partition language. Although this is a different definition of utility, note that these two notions of utility (restricting the bundles vs. count only the k-best items) are actually equivalent. That is, there exists an allocation π according to the first definition that achieves a utility vector of $(u_1, u_2, \ldots, u_n)$ for all the agents if and only if there exists an allocation π' according to the second definition that achieves the same utility vector of $(u_1, u_2, \ldots, u_n)$. Given an allocation according to the second definition, we only keep the $c_i(O)$ items with the highest utility for each equivalence class O of an agent i. This will not

give us any less utility, while this new allocation does satisfy the requirement for the first allocation.

6.3.2 2-conjunctive preferences

The extended preference language allows us to encode a 'bonus' for not choosing any of two items. We would like to also be able to encode conjunctive preferences for two objects. That is, for two objects o_1 and o_2 we want to model preferences in such a way that receiving one (or zero) of them does not give any utility, but receiving both of them does give some non-zero utility x.

We can actually reduce this case to the extended preference language in the following way. For an agent i that wants to represent this, we specify his equivalence relation $\sim_i$ such that we have two separate equivalence classes $\{o_1\}$ and $\{o_2\}$. We let $u_i(o_1) = u_i(o_2) = x$, $c_i(\{o_1\}) = c_i(\{o_2\}) = 1$ and specify that $m_i(\{o_1\}, \{o_2\}) = x$. We can now use the algorithm described above for finding the maximum utilitarian social welfare in the extended preference language. At the end, we subtract x utility from the final result.

To see why doing this is correct, note that any allocation that chooses neither o_1 nor o_2 receives x utility (because $m_i(\{o_1\}, \{o_2\}) = x$). If exactly one of these two items is chosen, the total utility is still x (now the bonus m_i is not received anymore), while in the case both items are chosen the total utility equals $2x$.

6.3.3 Allowing negative preferences

Another possible extension is to allow agents to encode negative preferences for items. Agents can specify whether they have a negative or positive preference for an item. If they have a positive preference for an item, they receive the associated utility if the bundle they receive contains that item (and not too many other items from that equivalence class are satisfied already) just as normal. However, if an agent indicates a negative preference for an item, they receive the associated utility if the bundle they receive does *not* contain that item (and again only if not too many other items from that equivalence class are satisfied).

In this way, we can represent 'unwanted' items in our representation language; an agent receives utility if his bundle does not contain that item. Note that the notion of negative preferences only makes sense if we require that the allocation is complete (i.e., all items must be divided). If not, we can always satisfy these negative preferences by simply not assigning these items to anyone if no agent has a positive preference for this item.

We will now define these preferences and how we calculate utility more formally. An agent can express negative preferences by splitting all items in a set of positive items $\mathcal{O}_{\text{pos},i}$ and a set of negative items $\mathcal{O}_{\text{neg},i}$. Of course, we require that $\mathcal{O}_{\text{pos},i} \cap \mathcal{O}_{\text{neg},i} = \emptyset$ and $\mathcal{O}_{\text{pos},i} \cup \mathcal{O}_{\text{neg},i} = \mathcal{O}$. Given a bundle $S \subseteq \mathcal{O}$ of an agent, we define the set of satisfied items for agent i, i.e., the set of items in the bundle that are in accordance with i's preferences, as follows:

$$S_{\text{sat},i} = (\mathcal{O}_{\text{pos},i} \cap S) \cup (\mathcal{O}_{\text{neg},i} \cap (\mathcal{O}/S))$$

Now we define the utility of a bundle in a similar way as we did in Section 6.3.1, namely by only counting the utility of the k best items that you have satisfied

from an equivalence class. This leads to the following definition of utility:

$$u_i(S) = \sum_{O \in \mathcal{O}/\sim_i} \max^{(c_i(O))}(O \cap S_{\text{sat},i})$$

We will for now restrict ourselves to the case where we only specify negative preferences for equivalence classes with $c_i(O) = 1$. We show that we can reduce the problem of finding maximum utilitarian welfare for this case to the extended preference language (with only positive preferences).

Suppose that we are given a fair division problem F with negative preferences specified as above. We construct the following fair division problem F' (with only positive preferences). For each item $o \in \mathcal{O}$, we create $n-1$ dummy items $o_{\text{dum},1}, \ldots, o_{\text{dum},n-1}$. Now, for every agent, we add these $n-1$ dummy items to the equivalence class O in which item o occurs, i.e., the new equivalence class becomes $O \cup \{o_{\text{dum},1}, \ldots, o_{\text{dum},n-1}\}$. If agent i values o positively, then we keep $u'_i(o) := u_i(o)$ and set all $u'_i(o_{\text{dum},j}) := 0$. If agent i values o negatively, then we set $u'_i(o) := 0$ and set all $u'_i(o_{\text{dum},j}) := u_i(o)$.

Claim 6.16. Let F and F' be as defined in the construction above. The maximum utilitarian welfare in F is equivalent to the maximum utilitarian welfare in F'

Proof sketch. Suppose we are given an allocation π for F, then we create a new allocation π' for F'. For all objects $o \in \mathcal{O}$ and agents $i \in N$, we specify that $o \in \pi'(i)$ if $o \in \pi(i)$ and otherwise give agent i one of the $n-1$ dummy items $o_{\text{dum},j}$. It is easy to see that allocation π' yields at least as much utility as allocation π.

Conversely, if π' is an allocation for F', then we define allocation $\pi(i) = \pi(i)/\{o_{\text{dum},1}, \ldots, o_{\text{dum},n-1}\}$ for F. That is, we choose the same allocation of objects as in π', but we get rid of all the dummy items. Since $c_i(O) = 1$ for all items with negative preferences, every agent can only have received either a single dummy item $o_{\text{dum},j}$ or the item o. In case the agent received o in π' and receives utility for this, he will still receive o in allocation π, so he receives the same utility. In case the agent received $o_{\text{dum},j}$ in π' and gets utility for this, he will thus not receive o in π. By construction of F' this agent can only have non-zero utility for $o_{\text{dum},j}$ if he values o negatively in F, so he will also receive the same utility for not having this item in π. This implies that the sum of utilities of allocation π is at least as large as the sum of utilities of π'. □

6.3.4 (Super)linear penalty

The preference language allows us to give an upper bound u on how many items an agent can receive from one equivalence class. This allows us to represent subadditive preferences: receiving all the items is worth less than the sum of receiving each of these items individually. However, the cutoff is quite strict. As long as an agent receives fewer than u items, the utility from this equivalence class is still additive. But starting from the $u+1$st item, the agent receives zero utility for these extra items.

We might like this cutoff between additivity and mutual exclusiveness to be more gradual. That is, starting from some number of items l, receiving more than l items will incur a penalty in utility that increases as more items are chosen from that equivalence class. We show that we can express such preferences using the extended preference language.

More specifically, we can specify the following preferences. For an equivalence class $O \subseteq \mathcal{O}$, let us specify an upper bound u and a lower bound l with $l \leq u \leq |O|$ and a penalty vector $(p_1, \ldots, p_{u-l}) \in (\mathbb{N})^{u-l}$ such that $p_i \leq p_{i+1}$. The idea is that an agent cannot receive more than u items of an equivalence class (just like the c_i function in the language) and that for every item of this equivalence class more than l the agent receives penalty p_i. More formally, for a bundle S that an agent receive, we require that $|O \cap S| \leq u$ and calculate the utility in the following way:

$$u_i(S) = \sum_{o \in S} u_i(o) - \sum_{i=1}^{|O \cap S|-l} p_i$$

We claim that these type of preferences can already be encoded in the extended preference language. Suppose we are given such a fair division problem F with a (super)linear penalty: for some equivalence class O for an agent j, we are given a lower and upper bound and a penalty vector. We transform this into an equivalent fair division problem F' in the extended disjunctive partition language in the following way.

We add $u - l$ extra items to the equivalence class with the ith item o_i having utility $u_j(o_i) = p_i$ for agent j and utility 0 for all other agents and set $c_j(O) = u$. We now use the algorithm described above to find the maximum utilitarian social welfare and subtract $\sum_{i=1}^{u-l} p_i$ from the resulting total utility.

Claim 6.17. Let F and F' be defined as in the construction above. There exists an allocation π for F with utilitarian welfare x if and only if there exists an allocation π' for F' with utilitarian welfare $x + \sum_{i=1}^{u-l} p_i$.

Proof. Suppose that π is an allocation for F with utilitarian welfare x and let O be the equivalence class with (super)linear penalties. First consider the case where an allocation chooses l or fewer items from O. Then we can define π' to pick all items from O that π picked and still also pick all the $u - l$ items o_i, yielding a total utility of $x + \sum_{i=1}^{u-l} p_i$ in allocation π'.

Suppose now that the allocation π chooses $l + k$ items from O, then we can only pick $u - l - k$ of the extra items in π'. We would like to maximize utility, so we will pick the items with the largest utilities (that is, the items $o_k, o_{k+1}, \ldots, o_{u-l}$). So we define π' to pick all the items that π picks plus the $u - l - k$ extra items with the largest utilities. This yields us a total extra utility of $\sum_{i=k}^{u-l} p_i$ and thus our net extra utility is $\sum_{i=k}^{u-l} p_i - \sum_{i=1}^{u-l} p_i = -\sum_{i=1}^{k} p_i$. This penalty of $\sum_{i=1}^{k} p_i$ is the same as we receive according to π in F.

Conversely, if π' is an allocation for F' with utility $x + \sum_{i=1}^{u-l} p_i$, we can create an allocation $\pi(i) = \pi'(i)/\{o_1, \ldots o_{u-l}\}$. We simply keep the same allocation of π', but get rid of all the extra items. Now, if all extra items $o_1, \ldots o_{u-l}$ were

assigned in π', this means that there are l or fewer items chosen from O, thus π will not get any penalty and thus receive utility x.

If on the other hand there were $l + k$ items in O assigned according to π' with $k > 0$, this means that at least k of the extra items were not assigned in π', so the utility that we receive from the extra items is at most $\sum_{i=1}^{u-l} p_i - \sum_{i=1}^{k} p_i$. The penalty for receiving $l + k$ items in π is $\sum_{i=1}^{k} p_i$, so π indeed has utilitarian welfare of at least x. □

6.3.5 Restricting the items that can be divided

Until now, we have mostly looked at fair division problems where in the end every item will be allocated to someone. However, a possible scenario could be that we want to restrict the total number of items that are allocated. For example, in a fair division problem with 10 objects $\mathcal{O}$, you might want to require that at most 8 of those objects are allocated. It does not matter to you which of those objects are the ones that are allocated or the ones not allocated, as long as at most 8 of them are allocated.

We might even take this further by only requiring this for a subset of objects. As an example, if you have 10 items $\mathcal{O} = \{o_1, \ldots, o_{10}\}$, you might require that at most two of the objects from $\{o_3, o_4, o_5, o_6\}$ are assigned to an agent, while there is no such restriction for the other items. We will show that we can encode such preferences in the extended disjunctive partition language.

First of all, let us define such preferences more formally. We specify an equivalence relation $\sim_{\text{lim}}$ on the set of objects $\mathcal{O}$ and a function $c_{\text{lim}} : \mathcal{O}/\sim_{\text{lim}} \to \mathbb{N}$ on the equivalence classes. For any allocation π we require that

$$\forall_{O \in \mathcal{O}/\sim_{\text{lim}}} \; |\bigcup_{i \in N} \pi(i) \cap O| \leq c_{\text{lim}}(O).$$

The way we can encode this in the extended disjunctive partition language is by adding an extra agent n_{dummy} with her item equivalence relation $\sim_{n_{\text{dummy}}} := \sim_{\text{lim}}$ and capacity $c_{n_{\text{dummy}}}(O) := |O| - c_{\text{lim}}(O)$. We make sure to set the utility he receives for these items to some very high utility $u_{\max}$, i.e for all $o \in O$ we set $u_{n_{\text{dummy}}}(o) = u_{\max}$. We can take $u_{\max}$ to be a number higher than $\sum_{o \in O} \sum_{i \in N} u_i(o)$ for example.

The idea behind this is that we always want that agent n_{dummy} gets the maximum number of items he could receive according to his capacity function $c_{n_{\text{dummy}}}$. Because we set the utility he obtains for every item that he receives very high, any allocation that maximizes utilitarian welfare will give agent n_{dummy} all items he can possibly receive. If an allocation were to assign fewer items to n_{dummy}, the loss from not assigning the items to n_{dummy} would always be greater than the gain of the other agents.

Since any utilitarian welfare optimal allocation will assign exactly $|O| - c_{\text{lim}}(O)$ items to agent n_{dummy}, this means that there are only $c_{\text{lim}}(O)$ items that can still be divided between the other agents. Thus, if we calculate the utilitarian welfare of the fair division problem with the extra dummy agent and at the end subtract all the utility $u_{\max}$ that agent n_{dummy} received for his items, we can

calculate the allocation that yields maximum social welfare given the capacity constraints $c_{\lim}(O)$.

6.4 Relation to combinatorial auctions

As we already discussed in Section 5.9, maximizing utilitarian welfare in fair division is closely related to the winner determination problem in combinatorial auctions. We already saw that conjunctive weighted bids are similar to OR bids. However, now we are studying disjunctive weighted bids instead of conjunctive weighted bids.

A closely related bidding language found in the combinatorial auctions literature is the OXS bidding language. In the OXS language, an agent can describe their bids as an OR of a group of singleton atomic bids, combined by XORs (Müller, 2004a).

Definition 6.18 (OXS bids)**.** An agent $i \in N$ can specify multiple groups of singleton atomic bids, where the bids within a group are combined by XOR's and the groups themselves are combined by OR's. In other words, his valuation function v_i is of the form

$$v_i = [\ (\{g_{1,1}\}, p_{1,1}) \text{ XOR } \dots \text{ XOR } (\{g_{1,r_1}\}, p_{1,r_1})\ ] \text{ OR } \dots \text{ OR } [\ (\{g_{s,1}\}, p_{s,1}) \text{ XOR } \dots \text{ XOR } (\{g_{s,r_s}\}, p_{s,r_s})\ ],$$

where each $g_{j,k} \in G$ and $p_{j,k} \in \mathbb{N}$.

We claim that any fair division instance in the disjunctive partition language can actually also be described using OXS bids.

Proposition 6.19. Any expression in the (basic) disjunctive partition language can be expressed equivalently in OXS bids.

Proof sketch. Consider an arbitrary expression in the disjunctive partition language. This expression consists of a set of utility functions $\boldsymbol{u} = (u_1, \dots, u_n)$ and equivalence relations $\boldsymbol{\sim} = (\sim_1, \dots, \sim_n)$. Given an agent $i \in N$, define the following valuation function v_i in the OXS language:

$$v_i = \operatorname*{OR}_{O \in \mathcal{O}/\sim_i} \left[\operatorname*{XOR}_{o \in O} (\{o\}, u_i(o)) \right]$$

Now we claim that valuation function $v_i : 2^G \to \mathbb{N}$ is equivalent to the utility function $u_i : 2^G \to \mathbb{N}$ as defined in Definition 6.2. To see this, note that within every partition $O \in \mathcal{O}/\sim_i$, we can receive utility for at most one item $o \in O$ according to u_i. We ensure that this is the case in v_i by combining the singleton atomic bids $(o, u_i(o))$ using XOR.

Furthermore, to make sure that bids from different equivalence classes $O, O' \in \mathcal{O}/\sim_i$ can be accepted simultaneously, we combine these using OR. Note that with the OR operator, two bids can only be accepted simultaneously if they do not have any overlap. In our case, since O and O' are equivalence classes in a partition, they never have any overlap, i.e., $O \cap O' = \emptyset$. Hence, the OR operator also behaves as we would expect according to u_i. □

Because of this connection between the disjunctive partition language and the OXS bids language, we can relate algorithmic results from the combinatorial auctions literature to the disjunctive partition language. Lehmann et al. (2006) have shown the existence of a polynomial-time algorithm for the winner determination problem in case of OXS bids. The extended model we describe in Section 6.3 is however more general than the OXS bids language and allows us to also represent more complicated preferences.

Some of the extensions that we describe in Section 6.3 have also been studied separately. For example, Rothkopf et al. (1998) describe an algorithm for the winner determination problem in case all bids are specified as 2-conjunctive preferences (compare with Section 6.3.2). Müller (2004b) also investigates and finds a tractable algorithm for the winner determination problem additive language where convex discounts can be specified, which is similar to the (super)linear penalties we introduced in Section 6.3.4. The extended partition language we present however is more general, since in this model we can combine all of these extensions, including some extra extensions, all in the same representation language.

6.5 The conjunctive partition language

As we have shown in the previous sections, the idea of representation languages with partitions allowed us to find tractable algorithms in the case of disjunctive preferences. In this section, we look at the (in)tractability results when we use a similar model, but now with conjunctions instead of disjunctions.

From Section 5.2.2.2, we already know that conjunctive weighted bid languages are intractable. It would be interesting to see whether the conjunctive language also becomes tractable once you impose the additional restrictions that bids may not overlap. To do so, we first define the conjunctive partition language formally.

Definition 6.20 (Conjunctive partition language)**.** An expression in the *conjunctive partition language* consists of a set of equivalence relations $\boldsymbol{\sim} = (\sim_1, \ldots, \sim_n)$ on the set of items $\mathcal{O}$ and a set of utility function $\boldsymbol{u} = (u_1, \ldots, u_n)$ on these equivalence classes. Each $u_i : \{[o]_i \mid o \in \mathcal{O}\} \to \mathbb{N}$ indicates the utility of having every item in that equivalence class according to agent i.

Definition 6.21 (Utility of conjunctive partition language)**.** Given an expression in the conjunctive partition language $(\boldsymbol{u}, \boldsymbol{\sim})$, we define the utility of an agent i given a bundle $S \subseteq \mathcal{O}$ as

$$u_i(S) = \sum_{\{[o]_i \mid o \in \mathcal{O} \text{ and } [o]_i \subseteq S\}} u_i([o]_i).$$

Note that for the conjunctive partition language, we specify a single utility value for each equivalence class. This is unlike the disjunctive partition language, where we specified a utility value for every item (also those itsm that are in the same equivalence class). The reason for this is that a conjunctive bid can only be satisfied if all items from that bid are given to the agent. So either we receive the utility of all the items from the bid or we receive no utility at all. Thus, even

if we specified a utility value for every item, this would not make our language more expressive.

However, even in a very restricted form, the conjunctive partition language already becomes intractable. Our hardness proof relies on the following result from the literature.

Lemma 6.22. INDEPENDENT SET for graphs where each node has degree at most 3 is NP-complete.

Proof sketch. Garey et al. (1974) show that the vertex cover problem for graphs with degree of at most 3 is NP-complete. There is a close relation between the vertex cover problem and the independent set problem: a graph of n nodes has an independent set of size k if and only if that graph has a vertex cover of size $n - k$. This shows that INDEPENDENT SET DEGREE 3 is NP-complete. □

Theorem 6.23. USW for the conjunctive partition language is NP-complete, even if every equivalence class has size at most 3 and the utility of every equivalence class for any agent is equal to 1.

Proof. We will prove NP-hardness by reducing from INDEPENDENT SET DEGREE 3. Suppose that $G = (V, E)$ is a graph with degree at most 3. We define the following conjunctive partition language fair division problem $(N, \mathcal{O}, \boldsymbol{u}, \sim)$, where

$$N = V \cup \{n_{\text{dummy,i}} \mid 1 \leq i \leq d\},$$
$$\mathcal{O} = E \cup \{o_{\text{dummy,i}} \mid 1 \leq i \leq d\},$$

with $d = \lceil |E|/2 \rceil + 1$. For every agent $v \in V$ we define the following equivalence relation $\sim_v$ consisting of $d + 1$ equivalence classes:

- the first equivalence class is $E(v)$ (the edges originating from v; we have that $|E(v)| \leq 3$),
- for the rest we split up $\mathcal{O}/E(v)$ in d equivalence classes such that any equivalence class has size at most 3 and contains exactly one dummy item $o_{\text{dummy,i}}$. Any way to do this will suffice for our reduction; it is always possible to find such a partition in polynomial time.

For any agent $n_{\text{dummy,i}}$, we define the following equivalence relation $\sim_{n_{\text{dummy,i}}}$ consisting of d equivalence classes:

- one equivalence class $\{o_{\text{dummy,i}}\}$,
- we split up $\mathcal{O}/\{o_{\text{dummy,i}}\}$ in $d - 1$ equivalence classes such that any equivalence class has size at most 3 and contains exactly one dummy item $o_{\text{dummy,j}}$ (except for $o_{\text{dummy,i}}$). We can always find such a partition in polynomial time; any partition satisfying these properties is sufficient.

We furthermore let $u_i([o])$ be equal to 1 for any agent i and any equivalence class $[o]$. In Example 6.24, we illustrate how this reduction works for a specific graph.

We claim that G has an independent set of size k if and only if $(N, \mathcal{O}, \boldsymbol{u}, \sim)$ has an allocation with utilitarian social welfare of $k + d$.

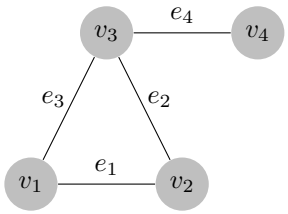
($\Rightarrow$) Suppose that G contains an independent set $V' \subseteq V$ of size k. We then define the following allocation:

$$\pi(i) = \begin{cases} E(i) & \text{if } i \in V' \\ \emptyset & \text{if } i \in V/V' \\ \{o_{\text{dummy,j}}\} & \text{if } i = n_{\text{dummy,j}} \end{cases}$$

Note that this allocation does not assign any item twice, since for any two nodes $i, j \in V'$, $E(i) \cap E(j) = \emptyset$ and every dummy agent receives 'his' dummy item. Furthermore, every agent in V' receives utility 1 and every dummy item receives utility 1 according to π and thus $\text{USW}(\pi) = k + d$.

($\Leftarrow$) Suppose there exists an allocation π with $\text{USW}(\pi) = k + d$. Since there are only d dummy items and every agent has at most one equivalence class without a dummy item, there must be at least k agents that got assigned an equivalence class with no dummy item. All these agents must be non-dummy agents, so we have a set $V' \subseteq V$ of size k with $v, v' \in V'$ implying that $E(v) \cap E(v') = \emptyset$. Hence, V' is indeed an independent set of size k. □

Example 6.24. As an illustrating example for the reduction defined in the proof of Theorem 6.23, suppose we are given the following graph G with degree at most 3 of which we want to find an independent set:



Using the reduction, we end up with the following fair division problem.

- $N = \{v_1, v_2, v_3, v_4, n_{d,1}, n_{d,2}, n_{d,3}\}$,
- $\mathcal{O} = \{e_1, e_2, e_3, e_4, o_{d,1}, o_{d,2}, o_{d,3}\}$,
- $u_i(o) = 1$ for all $i \in N$ and $o \in \mathcal{O}$,
- The following equivalence classes for the agents:

$$\begin{aligned}
\mathcal{O}/{\sim_{v_1}} &= \{\{e_1, e_3\}, \{o_{d,1}, e_2, e_4\}, \{o_{d,2}\}, \{o_{d,3}\}\} \\
\mathcal{O}/{\sim_{v_2}} &= \{\{e_1, e_2\}, \{o_{d,1}, e_3, e_4\}, \{o_{d,2}\}, \{o_{d,3}\}\} \\
\mathcal{O}/{\sim_{v_3}} &= \{\{e_2, e_3, e_4\}, \{o_{d,1}, e_1\}, \{o_{d,2}\}, \{o_{d,3}\}\} \\
\mathcal{O}/{\sim_{v_4}} &= \{\{e_4\}, \{o_{d,1}, e_2, e_3\}, \{o_{d,2}, e_4\}, \{o_{d,3}\}\} \\
\mathcal{O}/{\sim_{n_{d,1}}} &= \{\{o_{d_1}\}, \{o_{d,2}, e_1, e_2\}, \{o_{d,3}, e_3, e_4\}\} \\
\mathcal{O}/{\sim_{n_{d,2}}} &= \{\{o_{d_2}\}, \{o_{d,1}, e_1, e_2\}, \{o_{d,3}, e_3, e_4\}\} \\
\mathcal{O}/{\sim_{n_{d,3}}} &= \{\{o_{d_3}\}, \{o_{d,1}, e_1, e_2\}, \{o_{d,2}, e_3, e_4\}\}
\end{aligned}$$

G has an independent set of size k if and only if this fair division problem has an allocation with utilitarian welfare of $k + 3$. △

If we restrict the problem even further to requiring that every equivalence class has size at most 2, then the problem of maximizing USW does become tractable. We have actually already described this language in Section 6.3.2, where we discussed 2-conjunctive preferences. We showed here that these kind of preferences are expressible in the extended disjunctive partition language, for which we have a polynomial-time algorithm to compute MAX USW.

Theorem 6.25. MAX USW for the conjunctive partition language is computable in polynomial time if the size of all equivalence classes is at most 2.

Chapter 7

The partition language for leximin

In the previous chapter, we have looked at the disjunctive partition language and found tractable algorithms for maximizing utilitarian welfare. This raises the question of whether we can also find the maximum egalitarian welfare, a leximin-optimal allocation or the maximum Nash welfare of the disjunctive partition language efficiently. This is not possible without making further restrictions, as the next result shows.

The reason behind this is that the disjunctive partition language is still a more general version of the additive language. Since ESW, NSW and EEF are already NP-complete for the additive language, we will not be able to find a tractable algorithm for these solution concepts if we take the disjunctive partition language without any further restrictions.

Proposition 7.1. ESW, NSW and EEF for the disjunctive partition language are NP-hard.

Proof sketch. We show that it is possible to encode any expression in the additive language in the disjunctive partition language. We can do this easily by choosing the same utility functions $\boldsymbol{u}$, but setting the equivalence relations $\sim_i$ to have a separate equivalence class for every item (every item is only similar to itself). In this case, our restriction that S contains at most one object of each equivalence class is trivially satisfied and thus we end up with the same problem as the additive language without any partitions. As ESW, NSW and EEF are already NP-hard for the additive language (see Theorems 4.4, 4.5 and 4.6), this implies that these decision problems are also NP-hard for this language. □

However, if we make a similar restriction as we did in Section 4.3 – namely, restricting the utilities of single items to only 0 and 1 – there does exist a polynomial-time algorithm that finds a leximin-optimal allocation (and thereby also the maximum egalitarian welfare allocation).

Theorem 7.2. MAX LEX for the binary-valued disjunctive partition language is computable in polynomial time.

In this chapter, we will provide an algorithm that solves this problem in polynomial time and prove its correctness. In order to do this, we first need to introduce a few definitions and lemmas. We start by proving a sufficient condition for proving that an allocation π is a leximin-optimal allocation in Section 7.1. In Section 7.2, we will subsequently describe an algorithm that returns an allocation satisfying this condition and prove that this is indeed the case. Next, we will show in Section 7.3 that the leximin-optimal allocation actually coincides with the allocation that maximizes Nash welfare in this setting. And finally, in Section 7.4 we look at which generalizations of this language are not possible to make whithout losing tractability by studying different hardness results.

7.1 A sufficient optimality condition

The goal of this section is to define an optimality condition for a leximin-optimal allocation. This will be helpful later on for proving that an algorithm returns a leximin-optimal allocation, since we then just need to show that the algorithm's output satisfies this optimality condition.

We first define the notion of a partial preference graph, which we will use in the optimality condition.

Definition 7.3 (Partial preference graph of N')**.** Given a fair division problem in the binary-valued disjunctive partition language $(N, \mathcal{O}, \boldsymbol{u}, \sim)$ and a subset of agents $N' \subseteq N$, we define the bipartite *partial preference graph of* N' $G_{p,N'} = (V, E)$ in the following way:

$$V = A \cup B \text{ with } A = \{(i, [o]_i) \mid o \in \mathcal{O}, i \in N'\} \text{ and } B = \mathcal{O}$$
$$E = \{\langle (i, O), o' \rangle \in A \times B \mid o' \in O \text{ and } u_i(o') = 1\}$$

This graph is similar to the preference graph defined earlier (see Definition 6.6), but only contains the item equivalence classes of agents in N'. Furthermore, the edges have no weights anymore, since we restrict ourselves to the 0/1 scenario now and they are only connected if the utility of that item for that agent is 1. If the utility is 0, the items are not connected. We give a concrete example of a fair division problem in the binary-valued disjunctive partition language and its partial preference graph in Example 7.8.

In a partial preference graph, we can represent an allocation π by a matching in this graph.

Definition 7.4. Given an allocation π and a set of agents N', we define $M_{\pi,N'}$ as the following matching in $G_{p,N'} = (V, E)$:

$$M_{\pi,N'} = \{\langle (i, O), o' \rangle \in E \mid o' \in \pi(i)\}$$

Note that $M_{\pi,N'}$ is indeed a matching, since an allocation π only assigns every good to one agent and at most one good to every equivalence class of an agent.

The general idea of the optimality condition will be that an allocation satisfies this condition if $M_{\pi,N'}$ is a maximum matching for various groups $N' \subseteq N$.

Next, we will prove a few lemmas that show the connection between matchings in this partial preference graph and utilitarian welfare. For this, we will also define the notion of utilitarian welfare for a certain group N'. This is simply equal to the sum of utilities of every agent's bundle in N'.

Definition 7.5. The utilitarian social welfare of a group of agents $N' \subseteq N$ according to an allocation π is defined as follows:

$$\mathrm{USW}_\pi(N') = \sum_{i \in N'} u_i(\pi(i))$$

The following connection then exists between the matching $M_{\pi,N'}$ we defined and utilitarian welfare of a group N'.

Lemma 7.6. Let $\pi : N \to \mathcal{O}$ be an allocation and $N' \subseteq N$ a subset of the agents. Then we have that

$$\mathrm{USW}_\pi(N') = |M_{\pi,N'}|.$$

Proof. Let $p(i) := \{o \in \mathcal{O} \mid u_i(o) = 1\}$ be the set of items that agent i gives utility 1 to. We have that

$$\mathrm{USW}_\pi(N') = \sum_{i \in N'} u_i(\pi(i)) = \sum_{i \in N'} \sum_{o \in \pi(i)} u_i(o) = \sum_{i \in N'} |\pi(i) \cap p(i)|.$$

We can split up the matching in disjoint sets $M_{\pi,N'} = \biguplus M_{\pi,N',i}$ with $M_{\pi,N',i}$ only containing all edges from an equivalence class of agent i. Then we have that

$$|M_{\pi,N'}| = \sum_{i \in N'} |M_{\pi,N',i}|.$$

Note that there is a bijection between $\pi(i) \cap p(i)$ and $M_{\pi,N',i}$. Every item $o \in \pi(i) \cap p(i)$ corresponds to exactly one edge $\langle (i, [o]_i), o) \rangle \in M_{\pi,N',i}$. Because of this bijection we have that $|M_{\pi,N',i}| = |\pi(i) \cap p(i)|$, which suffices to prove our claim that $\mathrm{USW}_\pi(N') = |M_{\pi,N'}|$. □

Lemma 7.7. Let $\pi : N \to \mathcal{O}$ be an allocation and $N' \subseteq N$ a subset of the agents. If $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$, then $\mathrm{USW}_\pi(N') \geq \mathrm{USW}_{\pi'}(N')$ for all allocations π'.

Proof. Suppose that $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$. Now suppose, towards a contradiction, that there exists an allocation π' with $\mathrm{USW}_{\pi'}(N') > \mathrm{USW}_\pi(N')$. Then by Lemma 7.6,

$$|M_{\pi',N'}| = \mathrm{USW}_{\pi'}(N') > \mathrm{USW}_\pi(N') = |M_{\pi,N'}|.$$

However, this implies that there exists a matching $M_{\pi',N'}$ larger than $M_{\pi,N'}$, contradicting our assumption that $M_{\pi,N'}$ is a maximum matching. □

Let us now illustrate these definitions and lemmas with an example.

Example 7.8. Consider a fair division problem in the binary-valued disjunctive partition language with three agents $N = \{1, 2, 3\}$ and four items $\mathcal{O} = \{o_1, o_2, o_3, o_4\}$. The agents have the following preferences on the items:

$$\begin{array}{llll} u_1(o_1) = 1 & u_1(o_2) = 1 & u_1(o_3) = 0 & u_1(o_4) = 0 \\ u_2(o_1) = 1 & u_2(o_2) = 1 & u_2(o_3) = 1 & u_2(o_4) = 0 \\ u_3(o_1) = 1 & u_3(o_2) = 1 & u_3(o_3) = 1 & u_3(o_4) = 1 \end{array}$$

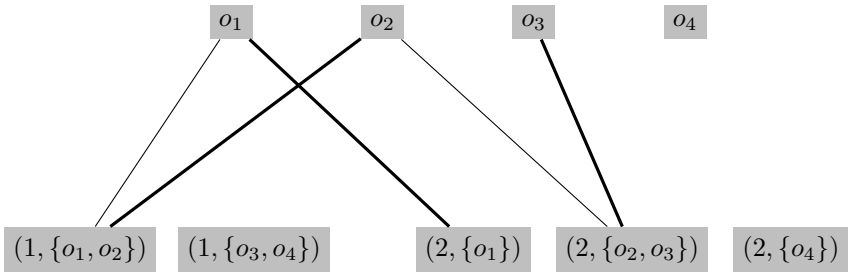
Furthermore, the agents' equivalence relations on the items are the following:

$$\begin{aligned} \mathcal{O}/{\sim_1} &= \{\{o_1, o_2\}, \{o_3, o_4\}\} \\ \mathcal{O}/{\sim_2} &= \{\{o_1\}, \{o_2, o_3\}, \{o_4\}\} \\ \mathcal{O}/{\sim_3} &= \{\{o_1, o_2, o_3\}, \{o_4\}\} \end{aligned}$$

Now consider the coalition $N' = \{1, 2\}$ and an allocation π with

$$\pi(1) = \{o_2\}, \pi(2) = \{o_1, o_3\}, \pi(3) = \{o_4\}.$$

The partial preference graph of N', $G_{p,N'}$, now looks like this:



The thick lines in the graph indicate the matching $M_{\pi,N'}$ corresponding to allocation π. Lemma 7.6 dictates that $\mathrm{USW}_\pi(N') = |M_{\pi,N'}| = 3$, so the utilitarian welfare of the coalition $\{1, 2\}$ for allocation π is 3. As $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$, Lemma 7.7 tells us that there is no other allocation π' that has a strictly higher utilitarian welfare for this coalition. △

We are now ready to define an optimality condition for when an allocation is leximin-optimal. Note that with the notation $N_{u,\dots,u',\pi}$, we mean all agents having a utility between u and u' according to allocation π. For convenience, we often leave out the allocation π in this notation if this is clear from the context.

Definition 7.9 (Optimality condition)**.** An allocation π satisfies the *optimality condition* if for all $u' \in \{u_{\min}, \dots, u_{\max}\}$ there exists a coalition N' with $N_{u_{\min},\dots,u'} \subseteq N' \subseteq N_{u_{\min},\dots,u'+1}$ such that $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$.

We will now show that this condition that we have defined is indeed an optimality condition: namely, any allocation that satisfies this condition is leximin-optimal.

Lemma 7.10. If an allocation π satisfies the *optimality condition*, then π is leximin-optimal.

Proof. Suppose towards a contradiction that there exists an allocation π' with a strictly higher lexicographic score. Let $(u_1^{\pi,s}, u_2^{\pi,s}, \dots, u_n^{\pi,s})$ be the sorted

utility vector of the agents according to allocation π and $(u_1^{\pi',s}, u_2^{\pi',s}, \ldots, u_n^{\pi',s})$ the sorted utility vector of allocation π'. Then, for a certain k, we have that $u_i^{\pi,s} = u_i^{\pi',s}$ for $1 \leq i \leq k-1$ and $u_k^{\pi',s} > u_k^{\pi,s}$.

As allocation π satisfies the optimality condition, there exists a coalition N' with $N_{u_{\min},\ldots,u_k^{\pi,s}} \subseteq N' \subseteq N_{u_{\min},\ldots,u_k^{\pi,s}+1}$ such that $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$. By lemma 7.7 this implies that $\mathrm{USW}_\pi(N') \geq \mathrm{USW}_{\pi'}(N')$.

Note first of all that

$$\mathrm{USW}_\pi(N') = \sum_{i=1}^{|N'|} u_i^{\pi,s},$$

since N' is the set of the $|N'|$ 'poorest' agents according to allocation π. To see why this is the case, remark that every agent not in N' has a utility higher or equal than $u_k^{\pi,s}+1$ and thus every agent not in N' has a higher or equal utility than the agents in N'. Hence, N' indeed consists of the $|N'|$ 'poorest' agents according to π.

Since all agents in N' have a utility of at most $u_k^{\pi,s}+1$, we have that

$$\begin{aligned}
\mathrm{USW}_\pi(N') &= \sum_{i=1}^{|N'|} u_i^{\pi,s} \\
&= \sum_{i=1}^{k-1} u_i^{\pi,s} + u_k^{\pi,s} + \sum_{i=k+1}^{|N'|} u_i^{\pi,s} \\
&\leq \sum_{i=1}^{k-1} u_i^{\pi,s} + u_k^{\pi,s} + (|N'| - k - 1) \cdot (u_k^{\pi,s} + 1).
\end{aligned}$$

Now, we do not know the exact utilities that π' assigns to the agents in N', but in the worst case they receive the $|N'|$ lowest utilities according to the sorted utility vector. Hence,

$$\begin{aligned}
\mathrm{USW}_{\pi'}(N') &\geq \sum_{i=1}^{|N'|} u_i^{\pi',s} \\
&= \sum_{i=1}^{k-1} u_i^{\pi',s} + u_k^{\pi',s} + \sum_{i=k+1}^{|N'|} u_i^{\pi',s} \\
&= \sum_{i=1}^{k-1} u_i^{\pi,s} + u_k^{\pi',s} + \sum_{i=k+1}^{|N'|} u_i^{\pi',s} && (u_i^{\pi,s} = u_i^{\pi',s} \text{ for } 1 \leq i \leq k-1) \\
&> \sum_{i=1}^{k-1} u_i^{\pi,s} + u_k^{\pi,s} + \sum_{i=k+1}^{|N'|} u_i^{\pi',s} && (u_k^{\pi,s} < u_k^{\pi',s}) \\
&\geq \sum_{i=1}^{k-1} u_i^{\pi,s} + u_k^{\pi,s} + (|N'| - k - 1) \cdot (u_k^{\pi,s} + 1). && (u_i^{\pi',s} \geq u_k^{\pi',s} \text{ for } i > k)
\end{aligned}$$

However, this would imply that $\mathrm{USW}_\pi(N') < \mathrm{USW}_{\pi'}(N')$, which is a contradiction. Therefore, allocation π must be leximin-optimal. □

7.2 The algorithm

Now that we have defined an optimality condition, let us define the algorithm that calculates a leximin-optimal allocation in polynomial time for the binary-valued disjunctive partition language and prove its correctness. The algorithm consists of two parts: the general algorithm (Algorithm 1) and the *FindSwap* algorithm (Algorithm 2).

Algorithm 1: The general algorithm

```
1 Take an empty allocation π' ;
2 π := π' ;
3 u' := 0 ;
4 while u' ≤ u_max (according to π) do
5 |   while FindSwap(π, u') returns a swap do
6 |   |   π := FindSwap(π, u');
7 |   end
8 |   u' := u' + 1 ;
9 end
```

The general algorithm starts with an empty allocation and $u' = 0$. The algorithm keeps calling *FindSwap* to find any swap that improves the utility of an agent with utility u' until there are no more swaps possible for that u'. The idea is that if we can find no more swaps for u', then the resulting allocation is leximin-optimal for those items with a utility smaller or equal than u'. We then increment u' by one and repeat the same process for agents having this utility until u' is larger than the utility of any of the agents in π.

Next we describe the *FindSwap* algorithm, which given an allocation π and a utility value u finds a swap that increases the utility of an agent with utility u'.

Algorithm 2: The *FindSwap* algorithm

Input: An allocation π and utility value u'

```
1  N' := N_{u_min,...,u'} ;
2  while M_{π,N'} has an augmenting path P starting from an N_{u'} node in G_{p,N'}
     do
3  |   n := the free A-node in augmenting path P ;
4  |   o := the free B-node in augmenting path P ;
5  |   M' := M_{π,N'} △ P ;
6  |   if o is unassigned or π^{-1}(o) ∈ N_{>u'+1} then
7  |   |   π' := π[M'] ;
8  |   |   return π' ;
9  |   end
10 |   N' := N' ∪ {π^{-1}(o)} ;
11 end
12 return no swaps possible ;
```

Here, $\triangle$ denotes the symmetric difference of two sets of edges. An augmenting path always contains one free A-node n (at the start of the path) and one free B-node o (at the end of the path). The node n represents (the equivalence class of) the agent that will get extra utility, while o represents the extra item that

will be assigned to one of the agents in order to realize this. So, $M_{\pi,N'} \triangle P$ represents the new matching in the partial preference graph $G_{p,N'}$ after applying augmenting path P.

The notation $\pi[M']$ (with M' some matching in $G_{p,N'}$) denotes an allocation where every item having an edge to an agent in M' is assigned to that agent, while every other item is assigned as in π. More formally,

$$\pi[M']^{-1}(o) := \begin{cases} i & \text{if } \langle (i, O), o \rangle \in M' \text{ for some } i \text{ and } O, \\ \pi^{-1}(o) & \text{otherwise.} \end{cases}$$

Thus, the new allocation π' is the same as π except for the agents involved in the augmenting path. Agent n will have a utility of one more in the new allocation π'. If o is already assigned to an agent, then that agent $\pi^{-1}(o)$ will receive a utility of one less, since o is no longer being assigned to him. All other agents involved in the augmenting path will keep the same utility, since they lose an item and gain another item.

So the idea of the *FindSwap* algorithm is that we are looking for a swap that increases the utility of an agent with utility u', while only decreasing the utility of some agent with utility $> u' + 1$. This way, any such swap has the net effect that the number of agents with utility u' decreases by one. If we find an augmenting path ending with an agent $\pi^{-1}(o)$ having utility precisely $u' + 1$, we add that agent to our coalition N' and try again to find a swap.

In order to prove the correctness of this algorithm, we will first show that the allocation that this algorithm outputs satisfies the optimality condition defined earlier. Secondly, we will show that this algorithm always finishes within a polynomial number of steps.

Proposition 7.11. The allocation outputted by Algorithm 1 satisfies the optimality condition.

Proof. We split up this proof in two claims. Let us say that π satisfies *the optimality condition for* u if there exists coalition N' with $N_{u_{\min},\ldots,u'} \subseteq N' \subseteq N_{u_{\min},\ldots,u'+1}$ and $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$.

Claim 7.12. Let π be an allocation that already satisfies the optimality condition for $u_{\min}$ up to $u - 1$. If *FindSwap*(π, u) cannot find a swap, then π also satisfies the optimality condition for u.

Proof of claim. In Algorithm 2, on input π and u, we first initialize N' as $N_{u_{\min},\ldots,u}$ and during the while-loop we potentially add agents $\pi^{-1}(o)$ to N'. Note that $\pi^{-1}(o) \in N_{u+1}$, since $\pi^{-1}(o) \notin N_{>u+1}$. So, it always holds that $N_{u_{\min},\ldots,u'} \subseteq N' \subseteq N_{u_{\min},\ldots,u'+1}$.

Now, whenever Algorithm 2 cannot find a swap anymore, it must be the case that $M_{\pi,N'}$ has no augmenting path in $G_{p,N'}$ starting from an N_u node in $G_{p,N'}$. We show that this implies that $M_{\pi,N'}$ has no augmenting path in $G_{p,N'}$ from any node.

Suppose there did exist an augmenting path starting from a node in N_{u+1}, then that node was added to N' in some iteration of the while-loop. But we only add nodes to N' if there exists an augmenting path from a node in N_u to that node, so then there must also exist an augmenting path starting from a N_u node. Furthermore, there cannot be an augmenting path starting from a node in $N_{u'}$ with $u' < u$, since this contradicts that π already satisfies the optimality condition for u'

Since there exists no augmenting path, this must mean that $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$ and so π satisfies the optimality condition for u. ■

Claim 7.13. Suppose that π is an allocation that satisfies the optimality condition for u and let $\pi' := \mathit{FindSwap}(\pi, u')$ for some $u' > u$. Then π' also satisfies the optimality condition for u.

Proof of claim. Since π satisfies the optimality condition for u, there exists N' with $N_{u_{\min},\dots,u'} \subseteq N' \subseteq N_{u_{\min},\dots,u'+1}$ such that $M_{\pi,N'}$ is a maximum matching in $G_{p,N'}$.

Allocation π' is the the same as π, with the exception of one swap starting from a node n_s in $N_{u'}$ and ending in a node n_e with $N_{>u'}$. After this swap, the only agents with a different net utility are agents n_s and n_e (all other agents involved in the swap keep the same net utility), both of which are not in N'. Hence, $\mathrm{USW}_\pi(N') = \mathrm{USW}_{\pi'}(N')$ and we have that $|M_{\pi,N'}| = |M_{\pi',N'}|$.

$M_{\pi',N'}$ must thus also be a maximum matching, since we know that $M_{\pi,N'}$ is a maximum matching and $M_{\pi',N'}$ is a matching with the same size. This shows that π' also satisfies the optimality condition for u. ■

We can now show that Algorithm 1 indeed outputs an allocation satisfying the optimality condition for all u. The algorithm starts by finding swaps for $u_{\min}$ until *FindSwap* can no longer find any swaps. Then we have by the first claim that the optimality property for $u_{\min}$ holds for the resulting allocation. The algorithm will then find swaps for $u_{\min} + 1$. Because of the second claim, the allocations after the swap will all still satisfy the optimality condition for $u_{\min}$. The algorithm continues doing this until no more swaps are possible, implying that the optimality condition is met for $u_{\min} + 1$. It will continue this process for all larger u. Hence, when the algorithm terminates the resulting allocation will satisfy the optimality condition for all u. □

Proposition 7.14. Algorithm 1 always finishes within a polynomial number of steps.

Proof. We start our proof by calculating a bound on the time a single *FindSwap* call (Algorithm 2) takes. This algorithm starts by checking whether $M_{\pi,N'}$ has an augmenting path. If so, then either the end node of this path is an unassigned object or an object belonging to an agent in $N_{>u+1}$. In these cases, we can return the swap right away. Otherwise, we add a single agent $\pi^{-1}(o)$ in N_{u+1} to N' and again search for an augmenting path. In the worst case, we might have to add all agents in N_{u+1} to N' one by one before the algorithm terminates (note that once all agents in N_{u+1} are added, any remaining augmenting path

directly gives a swap). So after at most n iterations, the *FindSwap* algorithm will terminate.

Next, we claim that if *FindSwap*(π, u) finds a swap, then the number of agents with utility u has decreased by one in the new allocation π'. To see this, if *FindSwap*(π, u) finds a swap, then the path that it found always starts with a node n_s in N_u and ends with a node n_e that is either in $N_{>u+1}$ or unassigned. The utility of n_s will increase by one and the utility of n_e will decrease by one, while the utility of all other agents will stay the same. So the net result is that there is one agent less with utility u, since agent n_s now has utility $u+1$ and agent n_e still has a utility strictly larger than u.

Now we are ready to give a bound on the complete algorithm by bounding the number of times *FindSwap* is called. Algorithm 1 calls the procedure *FindSwap* a number of times, for every u between 0 and $u_{\max}$. Note that by the claim above, after every swap *FindSwap*(π, u), the number of agents with utility u decreases by one. In the worst case, all n agents have utility u, so we have found all swaps for utility u after at most n calls to *FindSwap*(π, u). As we repeat this process for every u between 0 and $u_{\max}$ and the maximum utility possible is m, the number of goods, there are at most $n \cdot m$ calls to *FindSwap* in total.

So in order to calculate the final allocation, we need to make at most $n \cdot m$ calls to *FindSwap* that each again take at most n iterations. So in total, the number of steps needed to calculate the allocation is bounded by a polynomial.

It is also possible to prove an even tighter bound on the number of times *FindSwap* is called, but this is not needed to prove that Algorithm 1 runs in polynomial time. □

Now we have all the ingredients to prove the main theorem (Theorem 7.2) that there exists a polynomial-time algorithm for MAX LEX for the binary-valued disjunctive partition language.

Proof (of Theorem 7.2). By Proposition 7.11 we know that Algorithm 1 always outputs an allocation satisfying the optimality condition. Since any allocation satisfying the optimality condition is also leximin-optimal (because of Lemma 7.10), this shows that Algorithm 1 indeed returns a leximin-optimal solution whenever it terminates.

Finally, by Proposition 7.14, we know that Algorithm 1 always returns an allocation in a polynomial amount of time. Thus, it is indeed possible to calculate a leximin-optimal solution in polynomial time. □

7.3 Connection to Nash welfare

In Section 4.3, we studied the binary-valued additive language. This representation language can be seen as a more restricted version of the binary-valued disjunctive partition language. Namely, if we only have singleton equivalence classes, then the disjunctive partition language becomes the same as the additive language.

For the binary-valued additive language, Theorem 4.14 tells us that the Nash-optimal allocation and the leximin-optimal allocation coincide. It is therefore interesting to investigate the relation between these two solution concepts for the binary-valued disjunctive partition language, as this is a more general version. We will show in this section that these two solution concepts still coincide in this more general representation language.

Lemma 7.15. If an allocation π is Nash-optimal, π is also leximin-optimal.

Proof. Suppose that allocation π is not leximin-optimal. Then by Lemma 7.10 π does not satisfy the optimality condition. This means that there exists a $u' \in \{u_{\min}, \dots, u_{\max}\}$ such that for all coalitions N' with $N_{u_{\min},\dots,u'} \subseteq N' \subseteq N_{u_{\min},\dots,u'+1}$ there is an augmenting path for $M_{\pi,N'}$ in $G_{p,N'}$.

We claim that this implies that there exists a swap that increases the utility of an agent with utility $\leq u'$ by one and decreases the utility of an agent with utility $> u'+1$ at most by one (while keeping the utility of the other agents the same). Take $N' = N_{u_{\min},\dots,u'}$, then $M_{\pi,N'}$ has an augmenting path. If the last node (item o) of this augmenting path is unassigned or $\pi^{-1}(o) \in N_{>u'+1}$, then we have such a swap. Otherwise, we have that $\pi^{-1}(o) \in N_{u'+1}$.

In this case, we take $N' := N' \cup \{\pi^{-1}(o)\}$. There also must be an augmenting path for $M_{\pi,N'}$ in $G_{p,N'}$. If the last node (item o) of this augmenting path is unassigned or $\pi^{-1}(o) \in N_{>u'+1}$, then we have found a swap. Otherwise, we again add this agent $\pi^{-1}(o)$ to N' and repeat the process. At some point, we must either have found an augmenting path ending in an agent with utility $> u'+1$ or an unassigned node, or $N' = N_{u_{\min},\dots,u'+1}$. In the first case, we have found a swap and in the second case, $M_{\pi,N'}$ must still contain an augmenting path and this augmenting path must now end in an unassigned node or a node belonging to an agent in $N_{>u'+1}$. So we can indeed always find such a swap.

Finally, we show that such a swap implies that π is not Nash-optimal. Let π' be the allocation after the swap. Then there exist two agents $i, j \in N$ with $u_j(\pi(j)) - u_i(\pi(i)) > 1$ such that $u_i(\pi'(i)) = u_i(\pi(i)) + 1$, $u_j(\pi'(j)) = u_j(\pi(j)) - 1$ and $u_k(\pi'(k)) = u_k(\pi(k))$ for all $k \in N/\{i,j\}$.

Then the following holds:

$$\begin{aligned}
\mathrm{NSW}(\pi') &= \prod_{i \in N} u_i(\pi'(i)) \\
&= u_i(\pi'(i)) \cdot u_j(\pi'(j)) \cdot \prod_{k \in N/\{i,j\}} u_k(\pi'(k)) \\
&= (u_i(\pi(i)) + 1) \cdot (u_j(\pi(j)) - 1) \cdot \prod_{k \in N/\{i,j\}} u_k(\pi(k)) \\
&= (u_i(\pi(i) \cdot u_j(\pi(j)) + u_j(\pi(j)) - u_i(\pi(i)) - 1) \cdot \prod_{k \in N/\{i,j\}} u_k(\pi(k)) \\
&> u_i(\pi(i) \cdot u_j(\pi(j)) \cdot \prod_{k \in N/\{i,j\}} u_k(\pi(k)) \\
&= \mathrm{NSW}(\pi)
\end{aligned}$$

Hence, π' has a strictly higher Nash welfare than π, meaning that π is not Nash-optimal. □

Theorem 7.16. An allocation π is Nash-optimal if and only if π is leximin-optimal.

Proof. We know from Lemma 7.15 that if π is Nash-optimal, than π is leximin-optimal as well.

Conversely, let π be a leximin-optimal allocation and let π' be a Nash-optimal allocation. By Lemma 7.15, π' must then also be leximin-optimal. Because the lexicographic order is a total order, any two leximin-optimal allocation must have the same utility profile (with possibly a different permutation of agents). Since the Nash welfare of an allocation is calculated as the product of that utility profile, any two leximin-optimal allocations have the same Nash welfare. So π has the same Nash welfare as π', meaning that π must also be Nash-optimal. □

Now that we know that the notion of a Nash-optimal allocation and a leximin-optimal allocation coincide, we can conclude from this that we can also calculate the allocation with maximum Nash welfare in polynomial time using the same algorithm as we described in Section 7.2.

Corollary 7.17. MAX NSW for the binary-valued disjunctive partition language is computable in poynomial time.

Proof. We calculate a leximin-optimal allocation. By Theorem 7.2, we can calculate a leximin-optimal allocation for the binary-valued disjunctive partition language in polynomial time. And because of Theorem 7.16 this leximin-optimal allocation that we calculated also maximizes Nash welfare. □

7.4 Hardness results

Until now, we have mostly focused on tractability results in this chapter. Now, we will discuss a few variations and extensions of the disjunctive partition language that make the language intractable. This shows us the limits of the extent to which we can generalize this representation language while maintaining tractability. In order to make the intractability results as strong as possible, we prove NP-hardness results for a language that is otherwise as restricted as possible.

The first variation we discuss is specifying conjunctive partitions instead of disjunctive partitions. In Section 6.5, we defined a partition language with conjunctions instead of disjunctions and found that USW for such a language is intractable for partitions of size larger or equal than 3. We can pose the same question for egalitarian welfare. In this case, it turns out that allowing partitions of size larger or equal than 2 already suffices to obtain an intractability result.

Theorem 7.18. ESW for the binary-valued conjunctive partition language is NP-complete, even if every equivalence class has size exactly 2.

Proof. Consider the following decision problem:

THREE-DIMENSIONAL MATCHING
Input: Three disjoint sets X, Y and Z with $|X| = |Y| = |Z|$ and a relation $T \subseteq X \times Y \times Z$.
Question: Does there exist a perfect matching $M \subseteq T$ - that is, a subset M such that all $x \in X$, $y \in Y$, $z \in Z$ occur in exactly one triple of M?

This problem is known to be NP-complete (Karp, 1972). We will prove our theorem by reducing the problem of finding a perfect three-dimensional matching to the problem of finding an allocation with egalitarian welfare 1.

Let $n_x := |\{(y, z) \mid (x, y, z) \in T\}|$ be the number of edges in T in which $x \in X$ occurs. The general idea is to have an agent for every edge (x, y, z) in T. Now look at this from the perspective of a single element $x \in X$. If there exists a perfect matching, exactly one of the n_x edges containing x should be in this matching and the other $n_x - 1$ edges should not be.

We translate this into a fair division problem by adding $n_x - 1$ dummy items for each $x \in X$ such that we only need to satisfy a real bid for one of the n_x agents; we can satisfy all other $n_x - 1$ agents using the dummy items. We then give that agent (x, y, z) the items $\{y, z\}$, meaning that this edge (x, y, z) will be used in the perfect matching we construct. This ensures that y and z are only assigned to one agent and thus to only one edge of the perfect matching we are constructing. If we can do this for all items $x \in X$, we are able to extract a perfect matching from the resulting allocation.

Now we will define the reduction more formally and prove its correctness. We define a fair division problem $(N, \mathcal{O}, \sim, \boldsymbol{u})$ with $N = T$ and

$$\mathcal{O} = \bigcup_{x \in X} \{o_{x,i,j} \mid 1 \leq i \leq n_x - 1, j \in \{a, b\}\} \cup Y \cup Z.$$

For each agent $(x, y, z) \in N$ we define the following equivalence relation $\sim_{(x,y,z)}$ and utilities $u_{(x,y,z)}$:

- One equivalence class of $\{y, z\}$ with utility 1.
- For every i with $1 \leq i \leq n_x - 1$ an equivalence class of the form $\{o_{x,i,a}, o_{x,i,b}\}$ with utility 1.
- Lastly, we take all items that we have not yet divided $\mathcal{O}_{\text{rest}}$ and split them up in equivalence classes each of size 2. We assign utility 0 to all these equivalence classes.

We illustrate the idea of this reduction with an example in Figure 7.1.

$$
\begin{array}{lcl}
(x_1, y_1, z_1) & & \mathcal{O}/{\sim_{(x_1,y_1,z_1)}} = \{\{y_1, z_1\}, \{o_{x_1,1,a}, o_{x_1,1,b}\}, \{o_{x_1,2,a}, o_{x_1,2,b}\}, \dots\} \\
(x_1, y_2, z_3) & \Rightarrow & \mathcal{O}/{\sim_{(x_1,y_2,z_3)}} = \{\{y_2, z_3\}, \{o_{x_1,1,a}, o_{x_1,1,b}\}, \{o_{x_1,2,a}, o_{x_1,2,b}\}, \dots\} \\
(x_1, y_1, z_2) & & \mathcal{O}/{\sim_{(x_1,y_1,z_2)}} = \{\{y_1, z_2\}, \{o_{x_1,1,a}, o_{x_1,1,b}\}, \{o_{x_1,2,a}, o_{x_1,2,b}\}, \dots\} \\
\vdots & & \vdots
\end{array}
$$

Figure 7.1: An example of how to transform edges $(x, y, z) \in T$ into agents and their equivalence classes using the reduction. We have excluded in this example the equivalence classes consisting of $\mathcal{O}_{\text{rest}}$.

We now claim that there exists a perfect matching $M \subseteq T$ if and only if $(N, \mathcal{O}, \sim, \boldsymbol{u})$ has an allocation with egalitarian welfare 1.

($\Rightarrow$) Suppose that M is a perfect matching. Then we define the following allocation π. For every agent $(x, y, z) \in M$, we assign $\pi((x, y, z)) = \{y, z\}$ so that he receives a utility of 1. For every agent $(x, y, z) \notin M$, we assign $\pi((x, y, z)) = \{o_{x,i,a}, o_{x,i,b}\}$ for some i that is not yet assigned, giving that agent utility 1 as well. Note that we have enough $o_{x,i,a}$ and $o_{x,i,b}$ to do so, since there $n_x - 1$ such items and n_x agents who possibly want these objects. Exactly one of these agents $(x, y, z) \in M$ and thus does not need to receive such an $o_{x,i,a}$ or $o_{x,i,b}$. Hence, this is a valid allocation and every agent receives utility 1.

($\Leftarrow$) Conversely, suppose that π is an allocation with egalitarian welfare 1, meaning that every agent has utility at least 1. The only way for an agent $(x, y, z) \in N$ to receive a utility of at least 1 is if he receives either items y and z or items $o_{x,i,a}$ and $o_{x,i,b}$ for some i. Because for a given $x \in X$ there are n_x agents containing x and only $n_x - 1$ $o_{x,i,a}$ and $o_{x,i,b}$ items, at least one agent gets assigned y and z.

So we can define the following matching:

$$M = \{(x, y, z) \in T \mid y, z \in \pi((x, y, z))\}$$

By the argument above, for every $x \in X$ there exists y and z such that $(x, y, z) \in M$. Note furthermore that each $y \in Y$ and $z \in Z$ can only appear at most once in an edge in the matching M, since y and z can only have been assigned to one agent in allocation π. Hence, $|M| \leq |Y| = |Z| = |X|$.

Because we know that every $x \in X$ occurs at least once in an edge of M and $|M| \leq |X|$, every $x \in X$ occurs exactly once in an edge of M and $|M| = |X|$. Thus, M is indeed a perfect matching. □

A different relaxation that is interesting to investigate is to allow more different utility values than just 0 and 1. All our tractability results in this chapter for egalitarian, lexicographic and Nash welfare assumed that the agents' utilities for an item could only be either 0 or 1. We already know that allowing arbitrary (positive) utilities makes maximizing Nash and egalitarian welfare intractable, since NSW and ESW for additive languages is NP-hard (see Theorem 4.4 and Theorem 4.5).

However, it could still be that up to a certain k, languages allowing utilities $\{0, 1, 2, \dots, k\}$ remain tractable to maximize Nash or egalitarian welfare for. However, it turns out that this becomes intractable from $k = 2$ already, as we will show in a moment.

But first of all, note that if the agents can specify utilities $\{0, 1, 2\}$ to items in the additive language, then the Nash-optimal and leximin-optimal solution no longer coincide. This is in contrast to the additive language with utilities $\{0, 1\}$ (see Theorem 4.14) and the binary-valued disjunctive partition language (see Theorem 7.16), where we know that these two concepts are the same.

Proposition 7.19. For the additive language where the utilities that agents can assign to items are restricted to $\{0, 1, 2\}$, the Nash-optimal allocation and the leximin-optimal allocation do not necessarily coincide.

Proof. Consider the following additive fair divison problem with $N = \{1, 2\}$ and $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5, o_6\}$. Agent 1 assigns utility 1 to every item and agent 2 assigns utility 2 to every item. That is, for all $o \in \mathcal{O}$, $u_1(o) = 1$ and $u_2(o) = 2$.

An allocation π that maximizes Nash welfare is to let $\pi(1) = \{o_1, o_2, o_3\}$ and $\pi(2) = \{o_4, o_5, o_6\}$. Then the utility vector is $(3, 6)$ and hence $\mathrm{NSW}(\pi) = 3 \cdot 6 = 18$. However, we can also obtain utility vector $(4, 4)$ by choosing allocation π' with $\pi'(1) = \{o_1, o_2, o_3, o_4\}$ and $\pi'(2) = \{o_5, o_6\}$. Since $(4, 4)$ is lexicographically better than $(3, 6)$, π cannot be a leximin-optimal allocation. □

Now, we will show that both ESW and NSW are NP-hard to decide for this representation language.

Theorem 7.20. ESW for the additive language with the restriction that the utility of every item is in $\{0, 1, 2\}$ is NP-complete.

The proof of this theorem was found by Golovin (2005). We include the proof here, because we make use of the same construction to prove NP-hardness for Nash welfare.

Proof of Theorem 7.20. We reduce from the problem (3,B2)-SAT, which is known to be NP-hard (Berman et al., 2003).

> (3,B2)-SAT
> *Input:* A formula φ in 3-CNF such that every literal occurs exactly twice in φ.
> *Question:* Is φ satisfiable?

Let φ be a formula consisting of n variables $\{x_1, \dots, x_n\}$ and k clauses $\{C_1, \dots C_k\}$ such that every literal occurs exactly twice. We specify the following fair division problem $F(\varphi) = (N, \mathcal{O}, \boldsymbol{u})$. We create $2n$ agents $N := \{(x_i, v) \mid 1 \leq i \leq n \text{ and } v \in \{\text{true}, \text{false}\}\}$ and three types of items $\mathcal{O} = \mathcal{O}_B \cup \mathcal{O}_C \cup \mathcal{O}_D$.

$\mathcal{O}_B = \{o_1^B, \dots, o_n^B\}$ consists of n goods with the utility

$$u_a(o_i^B) = \begin{cases} 2 & \text{if } a \in \{(x_i, \text{true}), (x_i, \text{false})\}, \\ 0 & \text{otherwise.} \end{cases}$$

$\mathcal{O}_C = \{o_1^C, \ldots, o_k^C\}$ is a set of k goods with the utility

$$u_a(o_i^C) = \begin{cases} 1 & \text{if } a = (x_j, v) \text{ and setting } x_j \text{ to } v \text{ makes clause } C_i \text{ true,} \\ 0 & \text{otherwise.} \end{cases}$$

And finally, $\mathcal{O}_D = \{o_1^D, \ldots, o_{2n-k}^D\}$ is a set of $2n-k$ dummy goods with $u_a(o_i^D) = 1$ for every agent $a \in N$.

Claim 7.21. $\varphi \in$ (3,B2)-SAT if and only if $F(\varphi)$ has an allocation such that the utility of every agent is 2.

Proof of claim. Suppose that $F(\varphi)$ has an allocation that gives every agent utility 2. Because the maximum sum of utilities possible in this fair division problem is $2 \cdot n + 1 \cdot k + 1 \cdot (2n - k) = 2 \cdot 2n$ and there are $2n$ agents, this is only possible if every good is assigned to an agent receiving utility for it. Then the (partial) assignment $A = \{\{x_i, v\} \mid o_j^C \in \pi(\{x_i, v\}) \text{ for some } j\}$ must make all the k clauses of φ true. Note that $\{x_i, \text{false}\}$ and $\{x_i, \text{true}\}$ cannot be both in the assignment, since either $\{x_i, \text{false}\}$ or $\{x_i, \text{true}\}$ must have received item o_i^B. Because this item already gives utility 2 to that agent, this agent cannot have received any other item from $\mathcal{O}_C$. Hence, the assignment A is valid and makes φ true.

Conversely, suppose that φ is satisfiable and let $A \subseteq N$ be an assignment that makes φ true. We define the following allocation π. For every $a \in N$, if $a = (x_i, v) \notin A$, then $\pi(a) := \{o_i^B\}$. For every clause C_i we choose (one of the) literals (x_j, v) that are satisfied according to A and assign o_i^C to agent (x_j, v). Because every literal occurs exactly twice, there will be no agent with utility more than 2. Finally, we assign all $\mathcal{O}_D$ to the agents that do not yet have utility 2. Because we assigned every item to an agent who receives utility for it and no agent receives utility more than 2, this must mean that π gives every agent utility exactly 2. This idea is also illustrated in Figure 7.2. ■

We can easily check whether there exists an allocation that gives every agent a utility of 2 by checking whether there exists an allocation with egalitarian welfare of 2. Hence, ESW is NP-hard for this language. □

		$\pi((a, \text{true})) = \{\boldsymbol{o_1^C}, o_1^D\}$
$\varphi = (\boldsymbol{a} \vee \neg b \vee \neg c) \wedge (\neg a \vee c \vee \boldsymbol{b})$		$\pi((a, \text{false})) = \{o_a^B\}$
$a \to \text{true}$		$\pi((b, \text{true})) = \{\boldsymbol{o_2^C}, o_2^D\}$
$b \to \text{true}$	$\Rightarrow$	$\pi((b, \text{false})) = \{o_b^B\}$
$c \to \text{false}$		$\pi((c, \text{true})) = \{o_c^B\}$
makes φ satisfiable		$\pi((c, \text{false})) = \{o_3^D, o_4^D\}$
		gives everyone utility 2

Figure 7.2: An example of a (3,B2)-satisfiable formula φ and how we can transform it into an allocation that gives every agent utility 2.

Theorem 7.22. NSW for additive languages with the restriction that the utility of every item is in $\{0, 1, 2\}$ is NP-hard

Proof. We make use of the same reduction as in the previous proof. We claim that $F(\varphi)$ has an allocation such that the utility of every agent is 2 if and only if there exists an allocation with Nash welfare 2^{2n}.

Suppose π is an allocation such that $u_a(\pi(a)) = 2$ for every agent $a \in N$. Then,

$$\text{NSW}(\pi) = \prod_{a \in N} u_a(\pi(a)) = 2^{2n}.$$

Conversely, suppose $F(\varphi)$ has no allocation that makes the utility of every agent 2. Take an arbitrary allocation π' of $F(\varphi)$, then we know that this allocation does not give every agent utility 2. We have that

$$\sqrt[|N|]{\text{NSW}(\pi')} = \sqrt[|N|]{\prod_{a \in N} u_a(\pi'(a))} \leq^{(*)} \frac{\sum_{a \in N} u_a(\pi'(a))}{|N|} = \frac{\text{USW}(\pi')}{|N|} \leq 2.$$

where $(*)$ holds because of the inequality of arithmetic and geometric mean. $(*)$ is an equality if and only if it is the case that $a, b \in N$ implies $u_a(\pi'(a)) = u_b(\pi'(b))$.

We now claim that $\text{NSW}(\pi') < 2^{2n}$. We only need to consider the cases where the utilities of all agents are the same, since in all other cases $(*)$ is a strict inequality and it follows from the equation above that the Nash welfare is strictly smaller than 2. We know that π' cannot assign every agent's utility to 2^{2n}, so the only cases left are when every agent receives utility 1 or 0. But in those cases, the Nash welfare is respectively 1 and 0 and thus also strictly smaller than 2^{2n}.

Hence, for every allocation π' of $F(\varphi)$, $\text{NSW}(\pi') < 2^{2n}$, meaning that there exists no allocation of $F(\varphi)$ with Nash welfare 2^{2n}.

Combining this with Claim 7.21, we can thus find out whether formula $\varphi \in$ (3,B2)-SAT by constructing fair division problem $F(\varphi)$ and checking whether $F(\varphi)$ contains an allocation with Nash welfare 2^{2n}. This proves that NSW is NP-hard for this language. □

Chapter 8

Fixed-parameter tractability

In the previous chapters, we have looked at various representation languages and studied the computational complexity of finding the allocations elected by the solution concepts for these languages. Many times, we also found intractability results such as NP-hardness, which show us that it is impossible to find a polynomial-time algorithm for that problem unless P = NP.

It might however be interesting to study these intractable problems on a finer scale. In all of the representation languages we have considered so far, various parameters can be identified in the input of the problem, such as the number of agents, the number of items or the number of bids. It could be the case that the problem is actually exponential in a single parameter such as the number of agents and only requires polynomial time in the other parameters.

In this chapter, we will perform such an analysis: we analyze the role that the different parameters of a fair division problem play in the computational complexity of the fair division problems we studied earlier. To do so, we make use of concepts from parameterized complexity theory. In Section 8.1, we give a short overview and explanation of the concepts that we need from parameterized complexity. If the reader is already familiar with parameterized complexity, this section can be skipped.

We will first look at fixed-parameter tractability in general for representation languages that have polynomial-time computable utility functions (Section 8.2). Afterwards, we look in more depth at fixed-parameter tractability results specifically for the additive language (Section 8.3) and for the conjunctive weighted bids language (Section 8.4).

8.1 Preliminaries in parameterized complexity

In this section, we give a quick overview of the definitions and concepts from parameterized complexity that we use later in this chapter. For a more thorough overview of parameterized complexity, see, e.g., Cygan et al. (2015).

A classical decision problem $\mathcal{L} \subseteq \Sigma^*$ is defined as a subset of strings over the alphabet Σ. In a parameterized problem on the other hand, every string in the

language also includes an integer value called the parameter, i.e., a parameterized problem is a language $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$. As an example, suppose that we parameterize the SAT problem by the number of variables. Then a pair (φ, n) belongs to this parameterized language if and only if formula φ has n variables and is satisfiable.

We can now define what it means for a problem $\mathcal{L}$, parameterized by k, to be fixed-parameter tractable. Intuitively, this is the case whenever there exists an algorithm that decides P and runs in time that is allowed to be exponential (or worse) in k, but polynomial in all the other parameters of the input.

Definition 8.1 (Fixed-parameter tractability)**.** We call a parameterized problem $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$ *fixed-parameter tractable* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that there exists an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$ decides whether $(x, k) \in \mathcal{L}$ in time $f(k) \cdot |(x, k)|^{O(1)}$.

Usually, f here is an exponential function, although f is allowed to be any computable function. We call the class of all parameterized problems that are fixed-parameter tractable FPT.

For showing that a parameterized problem is fixed-parameter tractable, it suffices to describe an algorithm that solves the problem within the specified time bounds. To prove that a parameterized problem is fixed-parameter intractable, we require a different approach.

Just like we believe NP-hard problems to be intractable, there exists a class of parameterized problems called W[1]-hard problems that we believe to be fixed-parameter intractable. Although we have no hard evidence that these problems are indeed fixed-parameter intractable, we make the assumption that W[1] $\neq$ FPT, just like we often make the assumption in our intractability proofs that P $\neq$ NP.

For our purposes, we do not need to know how the complexity class W[1] is defined (a full definition of this complexity class is given in Cygan et al., 2015). We can show W[1]-hardness of parameterized problems by reducing from other parameterized problems that are known in the literature to be W[1]-hard. For parameterized problems, we need a slightly different version of a reduction to ensure fixed-parameter tractability is preserved under reductions.

Definition 8.2. A function $f : (x, k) \mapsto (x', k')$ is a *parameterized reduction* from parameterized problem $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$ to $\mathcal{L}' \subseteq \Sigma^* \times \mathbb{N}$ if it holds that:

- $(x, k) \in \mathcal{L}$ if and only if $(x', k') \in \mathcal{L}'$,
- $f((x, k))$ can be computed in time $f(k) \cdot |(x, k)|^{O(1)}$, and
- $k' \leq g(k)$ for some computable function g.

A parameterized reduction differs from a polynomial-time reduction in the requirement that there should be a bound on the new parameter k' in terms of the old parameter k. On the other hand, f is allowed to take exponential (or worse) time to compute in k, as long as it takes polynomial time in the other parameters.

Fixed-parameter tractability of parameterized problems is closed under parameterized reductions. We omit the proof of this property (for a proof, we refer to

Cygan et al., 2015).

Proposition 8.3. If there is a parameterized reduction from $\mathcal{L}$ to $\mathcal{L}'$ and $\mathcal{L}'$ is fixed-parameter tractable, then $\mathcal{L}$ is fixed-parameter tractable as well.

Conversely, if we know that problem $\mathcal{L}$ is fixed-parameter intractable and there exists a parameterized reduction from $\mathcal{L}$ to $\mathcal{L}'$, then $\mathcal{L}'$ must be fixed-parameter intractable as well. Moreover, parameterized reductions are transitive.

Proposition 8.4. If there is a parameterized reduction from $\mathcal{L}$ to $\mathcal{L}'$ and there is a parameterized reduction from $\mathcal{L}'$ to $\mathcal{L}''$, then there also exists a parameterized reduction from $\mathcal{L}$ to $\mathcal{L}''$

8.2 General algorithms

We start by analyzing the fixed-parameter tractability of computing the allocations elected by solution concepts from a general perspective. We consider algorithms that work on any representation language for which we can compute the utility of an agent's bundle in polynomial time. Recall that we defined this class of languages as representation languages with *polynomial-time computable utility functions* in Section 2.4.

An easy exponential-time approach to find the allocations that are elected by solution concepts for any representation language with polynomial-time computable utility functions is to iterate over all possible allocations π and compute the solution concept for each allocation. This leads to the following algorithm.

Theorem 8.5. Let R be a representation language with polynomial-time computable utility functions running in time $p(|x|)$. Then MAX USW, MAX LEX and MAX NSW are computable in time bounded by $n^m \cdot n \cdot p(|x|)$.

Proof sketch. This can be done be enumerating over all possible allocations. If we have n agents and m goods, then there are n^m ways to assign these goods to the agents (for each good we have n choices of agents to which we can assign it). For each allocation π, we compute the utility of every agent's bundle, which can be done in time $n \cdot p(|x|)$. We calculate either the utilitarian welfare, Nash welfare or the lexicographic vector and keep track of the current best allocation we have seen so far. The allocation that is returned at the end maximizes the solution concept. □

Corollary 8.6. MAX USW, MAX LEX and MAX NSW are fixed-parameter tractable in $n + m$.

This raises the question of whether these problems are also fixed-parameter tractable in just the number of items m or in just the number of agents n. For the number of items, it turns out that this is indeed the case. Such an algorithm may take exponential time in m, but can only take polynomial time in all the other parameters. For example, we cannot iterate over all allocations, since this also takes exponential time in n.

A different idea is to iterate over all possible partitions of the set of items $\mathcal{O}$ (of all sizes). The number of such partitions depends only on m – the size of $\mathcal{O}$ – and not on anything else. In the literature, the number of different ways to partition a set with exactly m elements is also called the mth Bell number.

Proposition 8.7. There are at most m^m ways to partition a set with exactly m elements.

Proof. First of all, note that a set of m elements can have at most m partitions. This is the case only when every element of the set is placed in a different partition.

Let S be a set with $|S| = m$. A single particular way to partition S can be described by a function $f : S \to \{1, \ldots, m\}$, which indicates for each element $s \in S$ to which of the (at most) m partitions it belongs.

Clearly, we can describe any possible way of partitioning S by a map of type $S \to \{1, \ldots, m\}$ (although some maps will refer to the same partitioning). Hence, the number of maps from S to $\{1, \ldots, m\}$ is an upper bound on the number of ways to partition S. This number of such maps is equal to m^m. □

Note that there exist other tighter upper bounds on Bell numbers in the literature, but for our purposes this simple upper bound suffices. We can namely use this bound to create an algorithm and show that this algorithm is fixed-parameter tractable with respect to the number of items m.

Theorem 8.8. Let R be a representation language with polynomial-time computable utility functions running in time $p(|x|)$. Then MAX USW is computable in time bounded by $O(m^m \cdot ((n+m)^3 + n \cdot m \cdot p(|x|)))$ and hence fixed-parameter tractable in m.

Proof sketch. We present an algorithm that enumerates over all possible partitions of the m items. By Proposition 8.7, there are at most m^m such partitions. For each partition, we do the following.

Let $P = \{\mathcal{O}_1, \ldots, \mathcal{O}_p\}$ be the partition of $\mathcal{O}$ we are currently enumerating, where $p \leq m$. The idea is that every agent i receives at most one partition $\mathcal{O}_l$ of P, which becomes agent i's bundle. If an agent does not receive any partition, his bundle becomes the empty set. Our goal is to find the assignment of agents to partitions such that the utilitarian social welfare is maximized.

To this end, we construct the following weighted bipartite graph $G_{\text{USW}} = (V, E, w)$:

$$\begin{aligned} V &= P \cup N \\ E &= P \times N \\ w(\langle \mathcal{O}_l, i \rangle) &= u_i(\mathcal{O}_l) \end{aligned}$$

Note that the maximum weighted matching of G indeed corresponds to the assignment maximizing utilitarian social welfare that we are looking for. G_{USW} consists of $n + m$ vertices and is bipartite, so computing the maximum weighted matching can be done in $O((n+m)^3)$ time (Munkres, 1957). Computing the weights of the graph takes time $O(n \cdot m \cdot p(|x|))$.

If we repeat this process for every partition of $\mathcal{O}$ while keeping track of the current best assignment, we will find the allocation that maximizes USW in time $O(m^m \cdot ((n+m)^3 + n \cdot m \cdot p(|x|)))$. Since the running time is only exponential in m, MAX USW is fixed-parameter tractable in m. □

We can use a similar approach as we did for utilitarian welfare to find the maximum Nash welfare. However, we now need to maximize the product of the weights of the matching instead of the sum of the weights. We can achieve this by taking the sums over the logarithms of the weights.

Theorem 8.9. Let R be a representation language with polynomial-time computable utility functions running in time $p(|x|)$. Then MAX NSW is computable in time bounded by $O(m^m \cdot ((n+m)^3 + n \cdot m \cdot p(|x|)))$ and hence fixed-parameter tractable in m.

Proof sketch. We use the same approach as in the proof of Theorem 8.9. We enumerate all partitions of the items $\mathcal{O}$.

For each partition $P = \{\mathcal{O}_1, \ldots, \mathcal{O}_p\}$, we now construct the following weighted bipartite graph $G_{\text{NSW}} = (V, E, w)$:

$$\begin{aligned} V &= P \cup N \\ E &= P \times N \\ w(\langle \mathcal{O}_l, i \rangle) &= \log u_i(\mathcal{O}_l) \end{aligned}$$

The difference between G_{NSW} and G_{USW} (in the proof of Theorem 8.9) is that we define the weights as the logarithms of the original utility. If we now compute the maximum weighted matching in G_{NSW}, we essentially find the matching maximizing the sum of the logarithms of the utilities. This is equivalent to maximizing the product of the original utilities.

The rest of the algorithm is equivalent to the algorithm for maximizing utilitarian welfare and hence also runs in time bounded by $O(m^m \cdot ((n+m)^3 + n \cdot m \cdot p(|x|)))$. □

For maximizing egalitarian welfare, we construct the same bipartite graph as we did for utilitarian welfare of which we need to find a matching. Now however, we want to find a matching of size n that maximizes the minimum weight in the matching.

Theorem 8.10. Let R be a representation language with a polynomial-time computable utility function running in time $p(|x|)$. Then MAX ESW is computable in time bounded by $O(m^m \cdot ((n+m)^3 + n \cdot m \cdot p(|x|)))$ and hence fixed-parameter tractable in m.

Proof. Just as in the previous proofs, we enumerate all partitions of the items $\mathcal{O}$. We construct the following weighted bipartite graph $G_{\text{ESW}} = (V, E, w)$:

$$\begin{aligned} V &= P \cup N \\ E &= P \times N \\ w(\langle \mathcal{O}_l, i \rangle) &= u_i(\mathcal{O}_l) \end{aligned}$$

This graph is the same as G_{USW} (defined in the proof of Theorem 8.9). The optimal matching that we are now looking for is however defined by another metric. We are looking for a matching of size n such that the minimum weight of the edges in the matching is maximized. This corresponds to assigning agents to the bundles in such a way that the least happy agent's utility is maximized. This problem is also known as the linear bottleneck assignment problem in the literature and can be solved efficiently in $O((n+m)^3)$ steps (Burkard and Derigs, 1980; Armstrong and Jin, 1992).

The total running time of the algorithm is $O(m^m \cdot ((n+m)^3 + n \cdot m \cdot p(|x|)))$ and thus fixed-parameter tractable in m. □

Now that we have seen that the problem of finding the allocations that are elected by various solution concepts is fixed-parameter tractable in the number of items m, we can ask ourselves the same question for the number of agents n. However, here the result is negative.

Theorem 8.11. If one of USW, ESW, NSW or EEF EXISTENCE is fixed-parameter tractable in n for any representation language with polynomial-time computable utility functions R, then P=NP.

Proof. Suppose that USW is fixed-parameter tractable in n for any representation language with polynomial-time computable utility functions. Note that the dichotomous language (see Chapter 3) has polynomial-time computable utility functions, since we can check easily given a propositional formula φ and a bundle whether that formula is true. This means that there must exist an algorithm solving USW for expressions in the dichotomous language running in time $f(n) \cdot p(|x|)$ with f some function (which is allowed to be exponential), p a polynomial, $|x|$ the input size and n the number of agents.

This also implies that there exists an algorithm running in time $f(2) \cdot p(|x|)$ – i.e., a polynomial algorithm – for solving USW for dichotomous language with only two agents. But from Corollary 3.5 we know that USW is already NP-hard for the dichotomous language and two agents.

The proof goes similarly for ESW, NSW and EEF EXISTENCE, since Corollaries 3.6, 3.7 and 3.8 respectively show that these decision problems are also already NP-hard for the dichotomous language and two agents. □

8.3 Algorithms for the additive language

Since the additive language has polynomial-time computable utility functions, computing an optimal solution for any decision problem is fixed-parameter tractable in the number of items m. Although we have proved in the section above that there is no fixed-parameter tractable algorithm in the number of agents n that works on any representation language with polynomial-time computable utility functions, there could still exist such an algorithm specifically for the additive language. However, it turns out that this is not the case.

Theorem 8.12. ESW, NSW and EEF EXISTENCE for the additive language are fixed-parameter intractable in n, unless P=NP.

Proof. By Theorem 4.4 (for ESW), Theorem 4.5 (for NSW) and Theorem 4.7 (for EEF EXISTENCE), these decision problems are already NP-hard for the additive language restricted to two agents ($n = 2$). A fixed-parameter tractable algorithm in n for one of these decision problems would give a polynomial-time algorithm when restricted to the special case of $n = 2$, which would imply that we have found a polynomial algorithm for an NP-hard problem. □

In order to find a fixed-parameter tractability result in n, we might try to look at a restricted version of the problem. One idea is to look at the additive language where all utilities are furthermore encoded in unary, as we also did in Section 4.2. We already know that the problem remains NP-hard under this restriction, but perhaps such a restriction might make the problem fixed-parameter tractable in the number of agents n.

It turns out unfortunately that this is not the case, as the following hardness result shows us. This hardness result does however require a different (stronger) complexity-theoretic assumption

Theorem 8.13. ESW, NSW and EEF EXISTENCE, with all preferences are encoded in unary, are fixed-parameter intractable in the number of agents n, unless FPT = W[1].

Proof. From the proof of Theorem 4.8, we know that UNARY BIN PACKING $\leq_p$ ESW. But it is known that UNARY BIN PACKING is W[1]-hard (Jansen et al., 2013) when parameterized by the number of bins. Looking at the reduction defined in the proof of Theorem 4.8, the number of bins corresponds linearly to the number of agents. So this also is a parameterized reduction from UNARY BIN PACKING in the number of bins to unary-encoded ESW in the number of agents.

Thus, we can conclude that unary-encoded ESW is W[1]-hard in the number of agents n. So if there did exist a fixed-parameter tractable algorithm for UNARY ESW in n, any problem in W[1] would have a fixed-parameter tractable algorithm, implying that FPT = W[1].

For NSW and EEF EXISTENCE, the proof works in the exact same way, but we now make use of Theorem 4.9 and Theorem 4.10 respectively for the reduction. □

This theorem shows us that there exists no fixed-parameter tractable algorithm for these solution concepts if the parameter is the number of agents n. However, as we will show below, there does exist a fixed-parameter tractable algorithm in $u_{\max}$ and n, where $u_{\max}$ is the maximum utility that a single agent can achieve. That is, $u_{\max} = \max_{i \in N} \sum_{o \in \mathcal{O}} u_i(o)$.

Theorem 8.14. MAX ESW and MAX NSW are fixed-parameter tractable in $u_{\max}$ and n.

Proof. We use a dynamic programming approach to design an algorithm. We will construct for all $k \leq m$ and $v_i \leq u_{\max}$ the following state:

$$\texttt{possible}(k, v_1, \ldots, v_n) = \begin{cases} 1 & \text{if } \exists \pi \text{ with } \bigcup_{i \in N} \pi(i) = \{o_1, \ldots, o_k\} \text{ s.t. } \forall i \in N : u_i(\pi(i)) \geq v_i \\ 0 & \text{otherwise} \end{cases}$$

In words, we let $\texttt{possible}(k, v_1, \ldots, v_n)$ be equal to 1 if and only if there exists an allocation π that only assigns the first k objects such that every agent i has at least v_i utility.

Once we have constructed this table of states, we can easily find the maximum egalitarian welfare and the maximum Nash welfare. For the maximum egalitarian welfare, we loop over all states $\texttt{possible}(k, v_1, \ldots, v_n)$ that are possible (i.e., equal to 1) and find the one where the smallest utility v_i is the largest. In formula form,

$$\max_{\pi} \text{ ESW}(\pi) \;=\; \max_{\substack{v_1,\ldots,v_n \\ \text{s.t. } \texttt{possible}(m,v_1,\ldots,v_n)=1}} \left(\min_{i \in N} v_i \right).$$

For Nash welfare, we also loop over all possible states, but now we find the one where the product of the utilities v_i is the largest. Thus,

$$\max_{\pi} \text{ NSW}(\pi) \;=\; \max_{\substack{v_1,\ldots,v_n \\ \text{s.t. } \texttt{possible}(m,v_1,\ldots,v_n)=1}} \left(\prod_{i \in N} v_i \right).$$

Now, we describe how to construct the dynamic programming table. We start by assigning $\texttt{possible}(0, 0, \ldots, 0) = 1$ and $\texttt{possible}(0, v_1, \ldots, v_n) = 0$ for all other values of v_i. We use the following recursive relation:

$$\texttt{possible}(k, v_1, \ldots, v_n) = \max_{i \in N}(\texttt{possible}(k{-}1, v_1, \ldots, \max(0, v_i{-}u_i(o_k)), \ldots, v_n))$$

The intuition behind this equation is that if it is possible to achieve utilities of at least $v_1, \ldots, v_n$ using the first k items, item o_k must have been given to one of the agents $i \in N$. We check every such possibility. If o_k is given to agent i, then he must have received $v_i - u_i(o_k)$ utility from the first $k-1$ items, while the other agents must have received the same utility v_j.

We wrap $v_i - u_i(o_k)$ into $\max(0, v_i - u_i(o_k))$, because it might also happen that $v_i - u_i(o_k)$ is negative, which indicates that agent i receives more utility than v_i by getting assigned item o_k. Since we are only checking whether it is possible for agent i to obtain *at least* utility v_i, we want to allow this.

We assume here that the allocation has to divide all the items, but it is easy to adapt the recursive relation to also allow non-complete allocation. We can do this by also returning 1 whenever $\texttt{possible}(k-1, v_1, \ldots, v_n) = 1$.

As for the time complexity, the dynamic programming table contains $O((u_{\max})^n \cdot m)$ entries and we need $O(n)$ time to compute each entry. Hence, our algorithm runs in time $O((u_{\max})^n \cdot m \cdot n)$ in total. □

Note that we can provide a slightly tighter running time on this algorithm. If $u_{i,\max}$ represents the maximum achievable utility for agent i, then a bound on the running time of the algorithm is also $O((\prod_{i \in N} u_{i,\max}) \cdot m \cdot n)$.

Now let us compare this algorithm to the two general exponential algorithms for computing ESW and NSW: the $O(n^m)$ algorithm described in Theorem 8.5 and the $O(m^m)$ algorithm described in Theorem 8.9 and Theorem 8.10. If every agent assigns a (positive) non-zero utility to every item, $u_{\max}$ quickly becomes

very large. In particular, this implies that $u_{\max} \geq m$, making the dynamic programming algorithm run in time $\Omega(m^n)$.

On the other hand, the dynamic programming algorithm is especially useful whenever each agent only assigns non-zero utility to a few items (and utility 0 to all the other items). In this situation, $u_{\max}$ can be much smaller than m, making an $O((u_{\max})^n)$ algorithm possibly a better choice than a $O(n^m)$ or $O(m^m)$ algorithm.

Another useful observation is that we do not need to compute the complete dynamic programming table when we are only interested in finding out whether an allocation exists with an egalitarian welfare of at least x. For this, it suffices to compute the table entry $\texttt{possible}(k, x, \ldots, x)$, so we do not need to compute the table for entries where $v_i \geq x$. Hence, the running time of our algorithm is bounded by $O(x^n \cdot m \cdot n)$. This algorithm is thus in particular useful when there are only a few agents and the egalitarian welfare value x that we wish to obtain is relatively low. This leads us to the following corollary.

Corollary 8.15. ESW is fixed-parameter tractable in $n + x$, where x is the egalitarian welfare value we wish to find the answer to whether an allocation with this value exists.

Lastly, for computing EEF EXISTENCE, a slightly different approach is required. Bliem et al. (2016) have devised such an algorithm that computes EEF EXISTENCE for additive preferences – also based on dynamic programming – running in time $O((u_{\max})^{n^2} \cdot mn^2)$.

8.4 Algorithms for the conjunctive weighted bids language

First of all, note that the conjunctive weighted bids language has polynomial-time computable utility functions, meaning that the fixed-parameter tractability results for $n + m$ and m from Section 8.2 also apply to this language. We can again ask ourselves whether there also exists a fixed-parameter tractable algorithm parameterized just by the number of agents n.

For utilitarian welfare, we already know that the problem remains NP-hard when restricted to 3 agents, so there cannot exist such a fixed-parameter tractable algorithm in n for USW.

Theorem 8.16. USW for the conjunctive weighted bids language is fixed-parameter intractable in n, unless P=NP.

Proof. By Theorem 5.25, USW for the conjunctive weighted bids language is already NP-hard when restricted to 3 agents. If there did exist a fixed-parameter tractable algorithm for USW in n, this would give a polynomial-time algorithm for the case of 3 fixed agents, implying that we found a polynomial-time algorithm for an NP-hard problem. □

Regarding conjunctive weighted bids for the solution concepts ESW, NSW and EEF, we already showed in Section 5.2.1 that finding the allocations elected by

these concepts remains NP-hard, even when all the weights can only be equal to 1. However, even with this restriction, there does not exist a fixed-parameter tractable algorithm for these solution concepts.

Theorem 8.17. ESW, NSW and EEF EXISTENCE for the conjunctive weighted bids language with weights restricted to 1 are fixed-parameter intractable in n, unless W[1]=FPT.

Proof sketch. We make use of Theorem 5.7, which tells us that any fair division problem in the unary additive language can be expressed in the conjunctive weighted bids language with weights restricted to 1. From Theorem 8.13 we furthermore already know that ESW, NSW and EEF EXISTENCE for the unary additive language are W[1]-hard.

Note that the reduction in the proof of Theorem 5.7 keeps the number of agents n the same. Thus, this reduction is a parameterized reduction in n. Combining this with Theorem 5.7, we find that ESW, NSW and EEF EXISTENCE for the conjunctive weighted bids setting with weights 1 must also be W[1]-hard. □

Because of the close connection between USW for conjunctive weighted bids and WEIGHTED INDEPENDENT SET (see Section 5.2.2.2), fixed-parameter intractability results for WEIGHTED INDEPENDENT SET also imply the same intractability result for USW for conjunctive weighted bids.

Proposition 8.18. USW for the conjunctive weighted bids language is fixed-parameter intractable in x, unless W[1]=FPT. Here, x is the utilitarian welfare value we wish to find out of whether an allocation with this value exists.

Proof sketch. By Downey and Fellows (1999) we know that WEIGHTED INDEPENDENT SET is W[1]-hard in k, where we want to find a weighted independent set with sum of weights k. Note that the reduction from WEIGHTED INDEPENDENT SET to USW in Theorem 5.21 is also a parameterized reduction in k. Hence, USW for the conjunctive weighted bids language is also W[1]-hard. □

Lastly, we study the conjunctive weighted bids language from a different parameter: the number of bids b submitted in total by the agents. If we parameterize the problem in this way, we can find a fairly easy $O(2^b)$ algorithm.

Theorem 8.19. USW, ESW and NSW for conjunctive weighted bids are fixed-parameter tractable in b, where b is the total number of bids.

Proof sketch. Let B_{tot} be the set of all b bids that the agents have. We iterate over every possible subset of $B' \subseteq B_{\text{tot}}$. For every subset B', we try to create an assignment π that satisfies all bids occurring in B'. That is, if $(B_{i,j}, u_{i,j})$ is a bid from agent i, we try to assign all items $B_{i,j}$ to agent i (i.e., $B_{i,j} \subseteq \pi(i)$). If this is not possible (for example because an item is assigned to two different agents), we skip this subset and continue with the next.

Otherwise, we end up with an allocation π that satisfies every bid in B'. We now compute the solution concept (USW, ESW or NSW) for this allocation π and keep track of the current maximum. Once we have iterated over all subsets of B_{tot}, we return the allocation that maximizes the solution concept. □

This of course raises the question of whether we can do any better than this. For utilitarian welfare, we show that a sub-exponential algorithm in the number of bids b is not possible, at least assuming that the Exponential Time Hypothesis is true.

The Exponential Time Hypothesis is another complexity theoretic assumption that is often made in proving lower bounds for computational problems. Note that the truth of the Exponential Time Hypothesis also implies that FPT $\neq$ W[1] (and hence also that P $\neq$ NP). Thus, this is a stronger complexity-theoretic assumption than we have made so far.

Theorem 8.20. Under the Exponential Time Hypothesis, there exists no $2^{o(b)}$ algorithm[1] that decides USW for the conjunctive weighted bids language.

Proof. By Lokshtanov et al. (2013) we know that there exists no $2^{o(v)}$ algorithm that decides INDEPENDENT SET if the Exponential Time Hypothesis is true. Here, v denotes the number of vertices of the graph in which we want to find an independent set.

Now suppose that there exists a $2^{o(b)}$ algorithm that decides USW for conjunctive weighted bids and let $G = (V, E)$ be a graph of which we want to find out whether there exists an independent set of size k. In Theorem 5.21, we provide a construction to transform G into a fair division problem F such that F has an allocation with utilitarian welfare k if and only if G has an independent set of size k. The construction assumes that a weighted graph is given of which we want to find the weighted independent set, but we can transform this to the unweighted variant by setting the weight of all edges equal to 1.

Since we create one bid for each vertex in G, the fair division problem F contains v bids. If we now use the $2^{o(b)}$ algorithm to decide whether there exists an allocation in F with utilitarian welfare k, we also know whether G has an independent set of size k. But this whole procedure runs in time $2^{o(v)}$, which would mean that we have found a $2^{o(v)}$ algorithm that decides INDEPENDENT SET. Hence, there cannot exist a $2^{o(b)}$ algorithm that decides USW for conjunctive weighted bids. □

[1]The little-o notation here expresses that the function grows asymptotically strictly slower than the other function. There are various ways to define this formally. Here, we use the definition that $f(n) \in o(g(n))$ if and only if $\lim_{n\to\infty} f(n)/g(n) = 0$.

Chapter 9

Conclusion

9.1 Summary

In this thesis, we have looked at many different representation languages for specifying utility functions over sets of objects and studied the computational complexity of finding the allocations that are elected by various solution concepts for these languages in the setting of fair division of indivisible goods. We have included an overview of the different complexity results we obtained for these languages in Table 9.2. This table is not intended to be a complete overview of all results obtained in the thesis, but it gives more insight in the relation between the different (in)tractability results that we proved.

The goal of this thesis was to find representation languages that have polynomial-time algorithms for finding allocations elected by solution concepts, while at the same time remaining as expressive as possible. The disjunctive partition language (see Chapter 6 and 7) is an example of such a tractable representation language that we discovered in this thesis.

Besides looking for tractable algorithms, we also proved various intractability results for different representation languages. These results are useful in two ways. First of all, these results show us the limits of what is possible to do in polynomial time and secondly, intractability results for restricted versions of representation languages also allow us to understand better from which part of the problem the intractability stems. From these intractability results, we can learn which restrictions on representation languages do not help us to achieve tractable results and which restrictions could help.

We started by studying the dichotomous language in Chapter 3. Although this language still looks fairly restrictive – agents can only have utility 0 or 1 – we found that for all solution concepts that we considered, finding allocations elected by these concepts was NP-hard even after simplifying the setting further.

In Chapter 4, we considered the additive language and saw that finding allocations elected by solution concepts is NP-hard for all solution concepts besides utilitarian welfare. While restricting the language by requiring a unary encoding of the utility values did not make the problem easier, restricting the language even

further to binary-valued utilities did make tractable algorithms possible for more solution concepts. The problem remains however that agents cannot represent any relations between objects in their utility functions.

Conjunctive and disjunctive weighted bids are a way to represent such relations between objects, which we study in Chapter 5. Because this is a generalization of the additive language, the only solution concept that could possibly have a polynomial-time algorithm is utilitarian welfare. We show that this setting is very similar to that of combinatorial auctions and prove NP-completeness for the general case of conjunctive weighted bids. When restricting ourselves to two agents, we do find a polynomial-time algorithm, but the problem becomes NP-complete when there are three (or more) agents, as we prove. For disjunctive weighted bids, we discover that, for all solution concepts considered, calculating allocations elected by these concepts is intractible in a very restrictive setting already and note that this may be because of the overlap that is allowed between the bids.

Motivated by this, we consider in Chapter 6 and 7 a restricted version of the disjunctive weighted bids language where there is no overlap between bids of a single agent: the disjunctive partition language. We find that this representation language has interesting complexity-theoretic properties.

For utilitarian welfare, there exists a tractable algorithm for calculating an optimal allocation based on weighted matchings. We subsequently study various language extensions that allow for more preferences to be represented in the disjunctive partition language, while still maintaining tractability.

For egalitarian welfare and the leximin-optimal allocation, we need to look at the binary-valued version of the disjunctive partition language. This is because the disjunctive partition language is more general than the additive language, which is only tractable for these solution concepts when we have binary-valued utility functions. We present and prove correctness of a polynomial-time algorithm that calculates allocations elected by these solution concepts. The idea of this algorithm is based on iteratively finding swaps of items that increase the lexicographic value of the allocation. Moreover, we show that in this setting the Nash-optimal allocation coincides with the leximin-optimal allocation.

We can thus conclude that the disjunctive partition language seems like a good example of a representation language that still allows agents to express non-trivial preferences and relations between objects and on the other hand is still tractable for various solution concepts to compute optimal allocations for.

Finally, in Chapter 8 we look at the computational problems from a different perspective, namely using parameterized complexity. We showed fixed-parameter tractability in the number of items for any reasonable representation language, which gives us a way to compute optimal allocations efficiently if the number of items is not too large. If we parameterize by the number of agents however, we find mostly fixed-parameter intractability results. Lastly, we discussed two fixed-parameter tractable algorithms for the additive language and the conjunctive weighted bids language specifically.

9.2 Directions for future research

Most representation languages that we studied in depth in this thesis were based on weighted bids. We looked for restrictions on this language that made it possible to compute the allocations that are elected by various solution concepts in polynomial time. One interesting direction for further research is to study other types of representation languages and find restrictions on them that make computing elected allocations possible in polynomial time or to look for different types of restrictions that might yield interesting complexity results.

For example, a different type of representation language to study is a language where each agent specifies their preferences in an ordinal way. The representation language then transforms this order to cardinal utilities by a positional scoring vector such as $(4, 3, 2, 1, 0)$ (meaning utility 4 for the most preferred item, utility 3 for the second-most preferred item and so on). We know that if this scoring vector is of the form $(1, 1, \dots, 1, 0, 0, \dots)$ that we can compute an optimal allocation efficiently in most cases, since this is (a special case of) the binary-valued additive language.

On the other hand, Baumeister et al. (2013) and Darmann and Schauer (2015) have shown that for the lexicographic and the Borda scoring vector, optimizing Nash and egalitarian welfare becomes NP-complete. It might be interesting to try to generalize this result further for other types of positional scoring vectors: for what types of scoring vectors do there exist tractable algorithms? What kind of restrictions do we need to put on these scoring vectors such that we can compute the allocations elected by various solution concepts efficiently?

Besides looking at different representation languages, we can also study the same representation languages but with other solution concepts. Perhaps we need different kinds of restrictions on the representation languages in order to make computing the allocations elected by these solution concepts tractable. For example, while we studied some complexity results for envy-free and Pareto efficient allocations (EEF EXISTENCE), we did not find many tractability results for this solution concept. We focused mostly on restrictions on representation languages that made leximin-optimal and Nash-optimal solutions tractable to compute. It could be interesting to look at which restrictions are necessary to make EEF EXISTENCE tractable to compute and whether this is possible without making the language completely trivial.

Another specific possible direction would be to study the algorithm we designed in Chapter 7 that computes a leximin-optimal allocation for the binary-valued disjunctive partition language in more detail. It might be possible to generalize the idea of the algorithm even further so that it works for a more expressive language as well. In that case, it is also interesting to investigate whether Nash welfare and the leximin-optimal allocation still coincide in this language. Does there exist a representation language where the Nash-optimal allocations and the leximin-optimal allocations are different, but they are both computable in polynomial time?

Finally, further research could also look in more depth at fixed-parameter tractability. In Chapter 8, we obtained some basic tractability and intractability results, but there are still many directions that we have not yet explored. One idea would

be to represent a fair division problem as a graph in some way (for example as we did with the bid graph in Section 5.2.2.2) and study the complexity of finding allocations elected by various solution concepts when parameterizing over the treewidth or cliquewidth of the graph[1]. By studying different parameters, one can easily obtain new fixed-parameter tractability and intractability results.

[1]These are measures that capture in a way how complex the structure of the graph is. A full definition of these concepts in the context of fixed-parameter tractability is given in Cygan et al. (2015).

	USW	ESW	LEX	NSW	EEF
dichotomous					
- identical preferences, 2 agents, pos. formulas	NP-c (Cor. 3.5)	NP-c (Cor. 3.6)	NP-c (Cor. 3.6)	NP-c (Cor. 3.7)	NP-h (Cor. 3.8)
- clauses	P (Thm. 3.10)	P (Thm. 3.11)	-	P (Thm. 3.12)	P (Thm. 3.13)
additive					
- general	P (Prop. 4.3)	NP-c (Thm. 4.4)	NP-c (Thm. 4.4)	NP-c (Thm. 4.5)	NP-h (Thm. 4.6)
- unary encoding	P (Prop. 4.3)	NP-c (Thm. 4.8)	NP-c (Thm. 4.8)	NP-c (Thm. 4.9)	NP-h (Thm. 4.10)
- binary-valued	P (Prop. 4.3)	P (Thm. 4.12)	P (Thm. 7.2)	P (Thm. 4.13)	NP-c (Thm. 4.15)
- 0/1/2-valued	P (Prop. 4.3)	NP-c (Thm. 7.20)	NP-c (Thm. 7.20)	NP-c (Thm. 7.22)	NP-h (Thm. 4.15)
conj. weighted bids					
- weights 1	NP-c (Cor. 5.23)	NP-c (Cor. 5.8)	NP-c (Cor. 5.8)	NP-c (Cor. 5.8)	-
- 2 agents	P (Thm. 5.24)	-	-	-	-
- 3 agents	NP-c (Thm. 5.25)	-	-	-	-
disj. weighted bids					
- 2 agents, weights 1, identical preferences	NP-c (Cor. 5.30)	NP-c (Cor. 5.31)	NP-c (Cor. 5.31)	NP-c (Cor. 5.32)	-
disjunctive partition					
- general	P (Cor. 6.9)	NP-c (Prop. 7.1)	NP-c (Prop. 7.1)	NP-c (Prop. 7.1)	NP-h (Prop. 7.1)
- with various extensions	P (Cor. 6.15)	NP-c (Prop. 7.1)	NP-c (Prop. 7.1)	NP-c (Prop. 7.1)	NP-h (Prop. 7.1)
- binary-valued	P (Cor. 6.9)	P (Thm. 7.2)	P (Thm. 7.2)	P (Cor. 7.17)	NP-h (Thm. 4.15)
conjunctive partition					
- size eq. class ≤ 3, all utilities 1	NP-c (Thm. 6.23)	-	-	-	-
- size eq. class $= 2$, binary-valued	P (Thm. 6.25)	NP-c (Thm. 7.18)	NP-c (Thm. 7.18)	-	-

Table 9.1: An overview of the complexity results for the different representation languages studied in this thesis. NP-hardness and NP-completeness are shortened to NP-h and NP-c respectively in this table.

Bibliography

Amanatidis, Georgios, Georgios Birmpas, and Evangelos Markakis (2018). "Comparing Approximate Relaxations of Envy-Freeness". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-2018)*, pp. 42–48.

Armstrong, Ronald and Zhiying Jin (1992). "Solving Linear Bottleneck Assignment Problems via Strong Spanning Trees". In: *Operations Research Letters* 12.3, pp. 179–180.

Arora, Sanjeev and Boaz Barak (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press.

Aziz, Haris and Simon Rey (2020). "Almost Group Envy-free Allocation of Indivisible Goods and Chores". In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI-2020)*.

Bansal, Nikhil and Maxim Sviridenko (2006). "The Santa Claus Problem". In: *Proceedings of the 38th annual ACM symposium on Theory of Computing*, pp. 31–40.

Barman, Siddharth, Sanath Kumar Krishnamurthy, and Rohit Vaish (2018). "Greedy Algorithms for Maximizing Nash Social Welfare". In: *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2018)*, pp. 7–13.

Baumeister, Dorothea, Sylvain Bouveret, Jérôme Lang, Trung Thanh Nguyen, Jörg Rothe, and Abdallah Saffidine (2013). "Positional Scoring Rules for the Allocation of Indivisible Goods". In: *Proceedings of the 11th European Workshop on Multi-Agent Systems (EUMAS-2013)*, pp. 1–14.

Berman, Piotr, Marek Karpinski, and Alexander Scott (2003). "Approximation Hardness of Short Symmetric Instances of MAX-3SAT". In: *Electronic Colloquium on Computational Complexity (ECCC)* 49.

Bliem, Bernhard, Robert Bredereck, and Rolf Niedermeier (2016). "Complexity of Efficient and Envy-Free Resource Allocation: Few Agents, Resources, or Utility Levels". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-2016)*, pp. 102–108.

Bouveret, Sylvain, Yann Chevaleyre, and Nicolas Maudet (2016). "Fair Allocation of Indivisible Goods". In: *Handbook of Computational Social Choice*. Cambridge University Press. Chap. 12, pp. 284–310.

Bouveret, Sylvain and Jérôme Lang (2008). "Efficiency and Envy-Freeness in Fair Division of Indivisible Goods: Logical Representation and Complexity". In: *Journal of Artificial Intelligence Research (JAIR)* 32, pp. 525–564.

Brooks, Rowland Leonard (1941). "On Colouring the Nodes of a Network". In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 37. 2. Cambridge University Press, pp. 194–197.

Burkard, Rainer and Ulrich Derigs (1980). "The Linear Bottleneck Assignment Problem". In: *Assignment and Matching Problems: Solution Methods with FORTRAN-Programs*. Springer, pp. 16–24.

Chevaleyre, Yann, Ulle Endriss, Sylvia Estivie, and Nicolas Maudet (2008). "Multiagent Resource Allocation in k-additive Domains: Preference Representation and Complexity". In: *Annals of Operations Research* 163.1, pp. 49–62.

Cramton, Peter, Yoav Shoham, Richard Steinberg, et al. (2004). *Combinatorial Auctions*. MIT Press.

Cygan, Marek, Fedor Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh (2015). *Parameterized Algorithms*. Springer.

Darmann, Andreas and Joachim Schauer (2015). "Maximizing Nash Product Social Welfare in Allocating Indivisible Goods". In: *European Journal of Operational Research* 247.2, pp. 548–559.

Downey, Rodney and Michael Fellows (1999). *Parameterized Complexity*. Springer-Verlag.

Garey, Michael, David Johnson, and Larry Stockmeyer (1974). "Some Simplified NP-Complete Problems". In: *Proceedings of the 6th annual ACM symposium on Theory of Computing*, pp. 47–63.

– (1976). "Some Simplified NP-Complete Graph Problems". In: *Theoretical Computer Science* 1.3, pp. 237–267.

Gary, Michael and David Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, New York.

Golovin, Daniel (2005). *Max-Min Fair Allocation of Indivisible Goods*. Tech. rep. School of Computer Science, Carnegie Mellon University. URL: `http://ra.adm.cs.cmu.edu/anon/usr0/ftp/usr/ftp/2005/CMU-CS-05-144.pdf`.

Halpern, Daniel, Ariel Procaccia, Alexandros Psomas, and Nisarg Shah (2020). *Fair Division with Binary Valuations: One Rule to Rule Them All*. arXiv: 2007.06073 [cs.GT].

Jansen, Klaus, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter (2013). "Bin Packing with Fixed Number of Bins Revisited". In: *Journal of Computer and System Sciences* 79.1, pp. 39–49.

Karp, Richard (1972). "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Springer, pp. 85–103.

de Keijzer, Bart, Sylvain Bouveret, Tomas Klos, and Yingqian Zhang (2009). "On the Complexity of Efficiency and Envy-Freeness in Fair Division of Indivisible Goods with Additive Preferences". In: *Proceedings of the 1st International Conference on Algorithmic Decision Theory (ADT-2009)*, pp. 98–110.

Kumar, T.K. Satish (2003). "Incremental Computation of Resource-Envelopes in Producer-Consumer Models". In: *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-2003)*. Springer, pp. 664–678.

Kyropoulou, Maria, Warut Suksompong, and Alexandros A. Voudouris (2019). "Almost Envy-Freeness in Group Resource Allocation". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-2019)*, pp. 400–406.

Lehmann, Benny, Daniel Lehmann, and Noam Nisan (2006). "Combinatorial Auctions with Decreasing Marginal Utilities". In: *Games and Economic Behavior* 55.2, pp. 270–296.

Lipton, Richard, Evangelos Markakis, Elchanan Mossel, and Amin Saberi (2004). "On Approximately Fair Allocations of Indivisible Goods". In: *Proceedings of the 5th ACM conference on Electronic Commerce (EC-2004)*, pp. 125–131.

Loker, David and Kate Larson (2010). "Parameterizing the Winner Determination Problem for Combinatorial Auctions". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2010)*, pp. 1483–1484.

Lokshtanov, Daniel, Dániel Marx, Saket Saurabh, et al. (2013). "Lower Bounds Based on the Exponential-Time Hypothesis". In: *Bulletin of the European Association for Theoretical Computer Science* 3.105, pp. 41–71.

Markakis, Evangelos (2017). "Approximation Algorithms and Hardness Results for Fair Division with Indivisible Goods". In: *Trends in Computational Social Choice*. AI Access. Chap. 12, pp. 231–247.

Müller, Rudolf (2004a). "Bidding Languages". In: *Combinatorial Auctions*. MIT Press. Chap. 9, pp. 319–336.

Müller, Rudolf (2004b). "Tractable Cases of the Winner Determination Problem". In: *Combinatorial Auctions*. MIT Press. Chap. 13, pp. 319–336.

Munkres, James (1957). "Algorithms for the Assignment and Transportation Problems". In: *Journal of the Society for Industrial and Applied Mathematics* 5.1, pp. 32–38.

Nguyen, Nhan-Tam, Trung Thanh Nguyen, Magnus Roos, and Jörg Rothe (2014). "Computational Complexity and Approximability of Social Welfare Optimization in Multiagent Resource Allocation". In: *Autonomous Agents and Multi-Agent Systems* 28.2, pp. 256–289.

Nisan, Noam (2000). "Bidding and Allocation in Combinatorial Auctions". In: *Proceedings of the 2nd ACM conference on Electronic Commerce (EC-2000)*, pp. 1–12.

Plummer, Michael and László Lovász (1986). *Matching Theory*. Elsevier.

Roos, Magnus and Jörg Rothe (2010). "Complexity of Social Welfare Optimization in Multiagent Resource Allocation". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2010)*, pp. 641–648.

Rothkopf, Michael, Aleksandar Pekeč, and Ronald Harstad (1998). "Computationally Manageable Combinational Auctions". In: *Management Science* 44.8, pp. 1131–1147.

Uckelman, Joel, Yann Chevaleyre, Ulle Endriss, and Jérôme Lang (2009). "Representing Utility Functions via Weighted Goals". In: *Mathematical Logic Quarterly* 55.4, pp. 341–361.

Zając, Mariusz (2018). *A Short Proof of Brooks' Theorem*. arXiv: `1805.11176 [math.CO]`.

Zhang, Yujiao, Xiao Duan, Xuerong Yue, and Zhibin Chen (2018). "A New Efficient Algorithm for Weighted Vertex Cover in Bipartite Graphs Based on a Dual Problem". In: *Proceedings of the 9th International Conference on Information Technology in Medicine and Education (ITME-2019)*, pp. 20–23.