Distributing tasks in committees

An algorithmic research in a cooperative game theory problem

Ivo van Kreveld

A thesis presented for the degree of
Master of Science
in
Computer Science

Algorithmics group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Delft, the Netherlands

May 2020

Distributing tasks in committees

An algorithmic research in a cooperative game theory problem

Ivo van Kreveld

Abstract

Cooperative game theory studies multi-agent environments where agents are able to make binding agreements. A lot has been written about dividing goods or other positive gains among the agents. This study investigates ways to distribute tasks with a negative utility in a strategyproof way. The intended application is a group of people or companies who can distribute such tasks between them to benefit from each other. The agents value tasks being done, but would rather not do it themselves. They can however, distribute the tasks to mutually benefit. Agents value tasks differently and also have different costs for them. This study investigates the theoretical properties of this problem. Particularly, we look at the Core, which is the set of solutions where agents have no incentive to form coalitions between them and ignore the result of the mechanism. Then, two algorithms are proposed to solve the problem. Finally, experiments are done to predict what results would occur in practice.

Thesis Committee:

Chair:	Dr. Mathijs de Weerdt
Committee Member:	Dr. Frans Oliehoek
Committee Member:	Dr. Robbert Fokkink

Preface

This is the report of my thesis project for the degree of Master of Science in Computer Science at the Delft University of Technology. In this project, I aim to gain new insights in multi-agent resource allocation by studying a game-theoretic model of an application in that area. The project was done at Delft University of Technology.

Firstly, I would like to thank my daily supervisor Mathijs de Weerdt. He was open to ideas for subjects that I was interested in, which allowed me to do research in the area I find most interesting. Also, he always stayed calm and was patient in what was a bit of a rough start of the project for me.

Secondly, I would like to thank the ones that helped me by checking my report, listening to me clearing up my thoughts, offering suggestions with their own perspectives, or helped me stay sane in general during the last year.

Ivo van Kreveld
May 28, 2020

Contents

Chapter 1

Introduction

In this chapter, we introduce fair division problems. We discuss applications of a task distribution problem, which we use to explain some assumptions in the model we use in this research. Then, we state the research question and explain its sub-questions. Finally, we provide an outline of this report.

1.1 Context

1.1.1 Fair division problems

Sometimes, multiple people or parties have to come to a general consensus on how to solve a specific problem. For example, 3 people are hiking and have collectively brought two apples, five cookies, and a sandwich to snack during the hike. Since they all need something to eat, they need to find a way to share these snacks among them. Of course, they can all just eat what they brought themselves. But that may not be ideal. Maybe, one person who brought an apple would rather eat a cookie, while another person who brought a cookie would rather eat an apple. Clearly, it would make both of these persons happier if they would trade snacks. Maybe it could be even better if they would just redistribute all snacks entirely. They can then find the ideal distribution given the preferences of the people for certain snacks. These type of problems are called *fair division problems*. This goods distribution problem is just one example of a fair division problem. Other examples exist, such as how to share rent in an apartment you share with several people who all have different room sizes, or how to distribute tasks for a school project. Even for the goods distribution problem, there are different assumptions you can make which lead to very different problems. For example, can we assume the individual snacks can be split? Can we assume players value two cookies twice as much as one cookie? Do we consider who brought what snacks in how good some distribution is?

Spliddit (A. Procaccia, Goldman, Shah, & Kurokawa, 2020) is an application where you can add your data for one of the five available fair division

problems (similar to the discussed goods distribution problem). The application then solves that problem for you. Chevaleyre et al. (2006) mention four application domains where fair division problems are relevant. One of those is the exploitation of Earth Observation Satellites, where multiple countries or companies fund a project together and share its exploitation. This shows that fair division problems are relevant in both personal smaller scale situations as well as economic larger scale situations.

Chevaleyre et al. (2006) discuss various fair division models. These models vary in what questions about the problem are asked, and what assumptions can be made. In this research, we consider a *task distribution* problem where not all tasks have to be done, but players gain utility for each task that is done.

1.1.2 Applications

One example where tasks need to be distributed is in a committee with different members. A committee generally does tasks to accomplish something. Each task contributes something to the project, which the committee members value. However, the members of the committee rather have someone else do it, because doing it themselves costs time. Therefore, they value a task being done positively, but doing the task themselves negatively. Additionally, different members of the committee might value the importance of each task and the cost of doing that task differently, which makes the positive and negative gain different for each member for each task.

A larger example could be different nations having multiple common goals, like in NATO. The tasks could be military tasks or taking measures for climate purposes. These tasks cost individual nations money, but the result has a positive value for all nations. Again, these costs and positive values can vary per nation.

An example that is less professional, but one that might have more interesting cases, is a group of friends organizing events for each other. The people in the group like that there are events, but it costs them time to organize the events themselves. In other examples, it makes sense to assign tasks to the people who value the task the most. However, in this specific example, that might not be optimal because if someone likes some event the most, say a scavenger hunt, he might not want to actually organize it, because he cannot participate as the organizer.

1.1.3 Task distribution

If the goal of a task distribution is to distribute the workload evenly, it can be solved in a similar way as a goods distribution problem (Chevaleyre et al., 2006). Instead of distributing *goods* with certain gains evenly, *tasks* with certain costs are distributed evenly. However, distributing evenly is not always the goal. In the applications above, certain parties might have way more to gain than other parties if the tasks are done, so it might be reasonable that they carry more workload. Besides that, parties in the applications can generally decide to not

do their assigned tasks, making the proposed distribution useless. Therefore, we would like a distribution where each party has an incentive to cooperate. We call this strategyproofness. This is the main motivation for this research. It studies a model where tasks need to be allocated to different parties. Different parties can have different costs for these tasks. In addition, these parties all gain something if tasks are done, and these gains can be different per party as well.

1.2 Research questions

In this research, the main question we would like to answer is:

How can we optimally distribute a set of tasks among a set of parties in a strategyproof way?

To answer this question, we need to answer the following sub-questions:

1. *What guarantees can an algorithm give about the total welfare and the strategyproofness of the task distributions it produces?*

   A distribution would be ideal if it had the maximum total welfare, and is strategyproof. We would like to know if such a strategyproof distribution always exists, and if it exists, what guarantees it can give us about the total welfare.

2. *Can the distribution with the optimal total welfare be found in a feasible amount of time?*

   To determine if the optimal total welfare can be found in a feasible amount of time, we prove whether this problem is efficiently solvable.

3. *Can a strategyproof distribution be found in a feasible amount of time?*

   To determine if a strategyproof distribution can be found in a feasible amount of time, we prove whether this problem is efficiently solvable and we theoretically and experimentally investigate the running time of the proposed algorithm.

1.3 Outline

This report is structured as follows. In Chapter 2, we give an overview of related work, and introduce the formal notation of such problems. In Chapter 3, we discuss the model, and introduce formal notation of this specific problem. In Chapter 4, we discuss the theoretical properties of this model. In Chapter 5, we propose two algorithms to solve the problem, a pseudo-polynomial exact algorithm, and a polynomial approximation algorithm. In Chapter 6, we disucss the results of the experiments with the implemented algorithms. Finally, in Chapter 7, we answer the research question, and give suggestions for future research.

Chapter 2

Background and related work

In Chapter 1, we introduced fair division problems in a non-technical way. In this chapter, we discuss the theory behind those fair division problems. In Chapter 3, we will use this to make a model for this research. This theory includes the assumptions and way of modelling used in cooperative game theory, the existing multi-agent resource allocation models, the different measures to maximize in an optimal (fair) model, and the assumptions and definitions of strategyproofness measures. Finally, we report on some relevant results for related problems.

2.1 Cooperative game theory

In game theory, there is a distinction between cooperative and non-cooperative game theory. What theory to use in a certain model very much depends on the specific properties of the application. Chalkiadakis, Elkind, and Wooldridge (2011) show this with a famous example of a non-cooperative game, namely the Prisoner's Dilemma.

It works as follows. Two players committed a crime and are held in two separate cells for interrogation. Both players are now presented with the choice to confess their crime, or stay silent. They know that if they both stay silent, there is only enough evidence to jail them for 1 year each. However, if one player confesses, that player will be freed while the other player will be jailed for 3 years. If both players confess, both players will be jailed for 2 years each.

The trick of this game is that even though it seems best for both players to stay silent, rational players will always confess. This is because the other player will not know your decision before he makes his. Therefore, it is always best to make the decision that is best for you. Say the other player confesses. In that case, if you stay silent, you will be jailed for 3 years. However, if you confess, you will only be jailed for 2 years, which is better. Now assume the other player

stays silent. In that case, if you stay silent as well, you will be jailed for 1 year. However, if you confess, you will be freed, which is also better. This means that whatever the other player does, it is always best for you to confess. Since this holds for both players, both players will confess, resulting in 2 years in jail for both of them.

Now the question is, how can we change the rules of the game to make sure both players will stay silent, which is the strictly better result? A possible answer is binding agreements. If, before they are brought to their cells, the players would be able to make a binding agreement that forces them to stay silent, the game would have a different result. As a player, you know that if no agreement is reached and the other player acts rationally, you will be in jail for 2 years. If however, an agreement is reached, both players have to stay silent, and you will be in jail for only 1 year. Therefore, it is best for both players to make this binding agreement.

These binding agreements are the distinction between cooperative and non-cooperative game theory. In cooperative games, we assume there is a possibility to make a binding agreement. Generally, we try to design algorithms that produce those agreements. In the context of this research, those agreements are called allocations. This is further discussed in Section 2.2. Of course, it should be in a player's interest to make a binding agreement. If a player or group of players can achieve a better result without making the binding agreement, they will not make the agreement at all, making it useless. Such an agreement is unstable, which is further discussed in Section 2.4.

In cooperative games, players usually have to cooperate to gain value. Often, we would like to find which players gain a high value by working together. These groups of players are called coalitions. Each coalition has a certain value they gain by cooperating. For example, a game might have players N_1, N_2 and N_3 where N_1 has a left glove, and N_2 and N_3 have right gloves, and the goal is to get a set of gloves. In this case, the coalitions $\{N_1, N_2\}$, $\{N_1, N_3\}$ and $\{N_1, N_2, N_3\}$ can all make this pair, so they have a higher value than all other coalitions, $\{N_1\}$, $\{N_2\}$, $\{N_3\}$ and $\{N_2, N_3\}$.

Chalkiadakis et al. (2011) define a cooperative game with an ordered pair $\langle N, v\rangle$, where N is the set of players and v is the characteristic function $v : 2^N \longrightarrow \mathbb{R}$. In such a game, $v(P)$ is the value a subset of players P can gain by cooperating.

2.1.1 Non-transferable utility games

The description for a cooperative game $\langle N, v\rangle$ with the amount of players N and the characteristic function $v : 2^N \longrightarrow \mathbb{R}$ is used for games with transferable utility. That is why the characteristic function only defines a utility for the group. Because of the transferable utility, the utility can be split between the group of players in any way. Therefore, the only relevant value of a coalition is the maximum utility they can gain. For example, if in a certain coalition, N_1 gains 12 utility and N_2 gains 0 utility, it is never relevant that there is another solution where N_1 and N_2 both gain 5 utility, as they could redistribute the

first situation to both gain 6 utility. In games with non-transferable utility, this redistribution is not possible. In this case, both situations from the example may be relevant, so a characteristic function with for all coalitions one value is insufficient. Instead, a characteristic function should contain all choices the players can make.

Chalkiadakis et al. (2011) define a cooperative non-transferable utility game with a structure $G = (N, \lambda, v, \succeq_1, ..., \succeq_n)$, where N is the set of players, λ is a set of choices, $v : 2^N \longrightarrow 2^\lambda$ defines for every coalition the choices they can make, and $\succeq_1, ..., \succeq_n$ is the preference relation between the choices for each player.

2.2 Multi-agent resource allocation

Multi-agent resource allocation is an area of research that uses cooperative game theory. It is relevant to both Computer Science and Economics (Chevaleyre et al., 2006). In multi-agent resource allocation, there is a number of resources that has to be distributed among several agents. Since we model this distribution as a game, we will call these agents the players.

Resources can be a lot of different things, depending on the application. Resources can be divisible or not. For example, money is generally assumed to be a divisible resource, while houses are not. Also, costs can in a way also be seen as resources. For example, an application might need to distribute tasks in a certain way among players.

A certain distribution of resources among players is called an allocation. Players have preference over these allocations. These preferences can be cardinal or ordinal. In most applications, there is some form of structure in these preferences. For example, if players have to divide a number of goods between them, they are often indifferent about which goods another player gets, they care only about their own goods.

2.3 Fairness

The question in multi-agent resource allocation is how to distribute resources between players. This has to be done in the best way possible. In other words, we have to find the best possible allocation. However, it is not immediately clear what the best allocations are. Clearly, there is a best allocation for each individual player, as they have preferences over allocations. But the question remains what the best allocation is overall.

In cases where players have cardinal preferences, it is possible to make a social welfare function. This is a function of the utility of each player for the allocation. The allocation can be valued with this function. The two most famous social welfare functions are the Utilitarian Welfare and the egalitarian welfare (Grant, Kajii, Polak, & Safra, 2010; Harsanyi, 1975; Myerson, 1981). The Utilitarian Welfare is the sum of the utilities of each player, while the egalitarian is the minimum of the utilities of each player.

The differences can be illustrated with an example. Say there are two players N_1 and N_2 and two allocations A and A^*, and let $u_i(X)$ denote the utility of player N_i for allocation X. Given $u_1(A) = 3$, $u_2(A) = 0$, $u_1(A^*) = 1$ and $u_2(A^*) = 1$. In this case, allocation A would be best from the utilitarian principle, because it gives the highest total utility, while allocation A^* would be best from the egalitarian principle, because the utility is distributed more evenly. Another social welfare function is the Nash Welfare (Kaneko & Nakamura, 1979), which is the product of utilities of all players.

Particularly between the utilitarian and the egalitarian principle, there has been a lot of discussion (Harsanyi, 1975; Myerson, 1981).

Besides these social welfare functions, there are a number of qualitative criteria with which you can value an allocation. Which of them makes sense depends on the specific problem. An example of such a criterion is envy-freeness. Informally, an allocation is envy-free if no player prefers the allocation of another player to his own allocation.

2.4 Strategyproofness

In Section 2.3, we discussed what an optimal allocation would be. However, as noted earlier, it is also important that the players are incentivized to make the agreement for that allocation. If a player (or group of players) can be better off without the agreement, it would not be rational to make it, making it useless to calculate it. For example, say player N_1 owns a book which gives him a utility of 1, and another player N_2 likes that book better, it would give him a utility of 2. If we would like to maximize total welfare, in an optimal allocation N_1 would give the book to N_2. However, since N_1 owns the book, he has the right to not give the book. And since he has incentive to keep it, he would do so if he is rational. He would just not participate in an agreement where he has to give the book to N_2.

This concept is formally known as the stable set or the Core. It was first used by Von Neumann, Morgenstern, and Kuhn (1953). They defined a dominance relation where some allocation A^* dominates some allocation A if there is a sufficient amount of players who prefers A^* over A. They also mention a set of allocations X (the Core) which contains two properties:

1. No $A \in X$ is dominated by an $A^* \in X$
2. Every $A \notin X$ is dominated by some $A^* \in X$

They also note that the relation of dominance is not transitive. If an allocation A^* dominates an allocation A and an allocation A' dominates the allocation A^*, that does not mean that A' dominates A. Because of this, there may not be an allocation that is in X. X can be empty.

Gillies (1959) first explicitly introduced the Core for games with transferable utility. Aumann (1961) adjusted this definition for games without transferable utility. He stated that an allocation A^* dominates an allocation A if:

1. there is a coalition P that prefers A^* to A, and

2. this preference is "not idle," i.e. P can actually achieve at least its portion of A^*

The Core is the set of allocations that is not dominated by any other allocation. Intuitively, we would like our algorithm to give an allocation A that is in the Core as otherwise, if all players in P are rational, they would not follow allocation A, but instead the allocation A^* that dominates A.

2.5 Allocation problems

In the most intuitive allocation problem, some goods with a positive value have to be distributed among several players. In most researches, it is assumed these values are additive. In this setting, while it is not always possible to find an envy-free allocation, Caragiannis et al. (2019) show that it is possible to find an allocation that is envy-free up to one good, with the maximum Nash Welfare solution. Ramezani and Endriss (2009) show that in general, finding this maximum Nash Welfare solution is NP-hard. It is also possible to compare utilities of players with the utility they would have gotten if they would be able to partition the goods in n bundles and receive the lowest valued bundle. In this case, A. D. Procaccia and Wang (2014) show that it is always possible to find an allocation where each player gains at least 2/3 of that value, and that this allocation can be found in polynomial time, given that the amount of players is constant.

In some models, players already own goods and an allocation needs to redistribute those goods. In these models, strategyproofness has to be taken into account. Allocations where a player is worse off than what he started with, or allocations where a coalition can redistribute goods on their own making them all better off, will not work because some players will not participate in the allocation. Shapley and Scarf (1974) show that in such markets, a non-empty Core always exists. Ma (1994) relates this Core property with the assumptions of individual rationality, Pareto optimality and strategyproofness.

Airiau and Endriss (2014) study a multiagent resource allocation model where resources can be shared among several players. They show several properties of specific instances of their model.

Weber and Wiesmeth (1991) use an allocation problem to model decisions in NATO. They model the problem more specifically with additional variables, and propose an efficient allocation.

2.6 Conclusion

Using the theory from this chapter, we will formalize the model introduced in Chapter 1, and propose measures to value an allocation.

Chapter 3

Model and definitions

Using the applications from Chapter 1 and the theory from Chapter 2, in this chapter, we propose a model and the measures we want to optimize. Firstly, we formalize the model in an abstract way. Secondly, we define the Utilitarian Welfare and the Core, the two optimization measures we study in this research.

3.1 Model

The problem we study in this research consists of a set of tasks T and a set of players N with $t = |T|$ and $n = |N|$. Each player i has a gain for each task j, denoted as g_{ij}, and a cost for each task j, denoted as c_{ij}. We assume valuations are additive, so $u_x = \sum_{j \in T | j \text{ is done}} g_{xj} - \sum_{j \in T | x \text{ does } j} c_{xj}$. Also, we assume all gains g_{ij} and costs c_{ij} are non-negative integers. An allocation $A = (A_1, ..., A_n)$ is a list of disjoint subsets of T, where A_i is the set of tasks player i is allocated. This means a task can only be done by one player, and not all tasks need to be distributed. The utility of each player i is his gain for all tasks that are allocated to anyone minus his cost for all tasks allocated to him, $u_x = \sum_{i \in N} \sum_{j \in A_i} g_{xj} - \sum_{j \in A_x} c_{xj}$. The goal is to find an allocation where the utility of the players is as high as possible. Since there are multiple players, this is a multi-objective problem. In Section 3.2, we explain the measures which we use to value such an allocation.

The data of the model consists of 2 tables. The first table G contains g_{ij} for each $i \in N$ and $j \in T$. The second table C contains c_{ij} for each $i \in N$ and $j \in T$. Note that these costs are visualized as negative numbers for readability. In this report, we will write this in the format as shown in Table 3.1.

Example *An example of an instance can thus look like Table 3.2. An example of an allocation could be $A = (A_1, A_2)$ where:*

$$A_1 = \{T_1, T_4\}$$
$$A_2 = \{T_2\}$$

Table 3.1: Format of the model

	Utility from task done		
	T_1	...	T_t
N_1	$g_{1,1}$	...	$g_{1,t}$
...	...	...	...
N_n	$g_{n,1}$	...	$g_{n,t}$
	Disutility for doing task		
	T_1	...	T_t
N_1	$c_{1,1}$	...	$c_{1,t}$
...	...	...	...
N_n	$c_{n,1}$	...	$c_{n,t}$

Table 3.2: Example of an instance

	Utility from task done			
	T_1	T_2	T_3	T_4
N_1	12	1	1	2
N_2	1	7	6	2
	Disutility for doing task			
	T_1	T_2	T_3	T_4
N_1	-2	-6	-4	-8
N_2	-1	-8	-5	-6

To calculate the utility for player N_1, we have to look at the value he gains for the tasks that are done and subtract the cost for doing tasks himself. Since tasks T_1, T_2 and T_4 are done (they are in $A_1 \cup A_2$), player N_1 gains value for those, and since N_1 does T_1 and T_4 (they are in A_1), player N_1 pays the cost for those. This gives them a utility of:

$$\begin{aligned} u_1(A) &= \sum_{i \in N} \sum_{j \in A_i} g_{1,j} - \sum_{j \in A_1} c_{1,j} \\ &= g_{1,1} + g_{1,2} + g_{1,4} - c_{1,1} - c_{1,4} \\ &= 12 + 1 + 2 - 2 - 8 = 5 \end{aligned}$$

Similarly,

$$\begin{aligned} u_2(A) &= \sum_{i \in N} \sum_{j \in A_i} g_{2,j} - \sum_{j \in A_2} c_{1,j} \\ &= g_{2,1} + g_{2,2} + g_{2,4} - c_{2,2} \\ &= 1 + 7 + 2 - 7 = 3 \end{aligned}$$

Note that this is not an optimal allocation. How to value such an allocation and what optimal allocations are is explained in Section 3.2.

3.2 Optimization measures

It is not immediately clear what an optimal allocation is. For this research, we will use the Utilitarian Welfare as well as a measure for stability, as defined by Von Neumann et al. (1953), as criteria.

3.2.1 Utilitarian Welfare

The Utilitarian Welfare is the sum of the utility of all players.

Definition 3.2.1. The *Utilitarian Welfare* of an allocation A is defined as $UW(A) = \sum_{i \in N} u_i(A)$.

Example *For example, the Utilitarian Welfare in the example allocation above is* $UW(A) = \sum_{i \in N} u_i(A) = u_1(A) + u_2(A) = 5 + 3 = 8$. *The allocation that maximizes this measure is* $A = (A_1, A_2)$ *where:*

$$\begin{aligned} A_1 &= \{T_2, T_3\} \\ A_2 &= \{T_1\} \end{aligned}$$

This gives the players a utility of

$$\begin{aligned} u_1(A) &= g_{1,1} + g_{1,2} + g_{1,3} - c_{1,2} - c_{1,3} \\ &= 12 + 1 + 1 - 6 - 4 = 4 \\ u_2(A) &= g_{2,1} + g_{2,2} + g_{2,3} - c_{2,1} \\ &= 1 + 7 + 6 - 1 = 13 \end{aligned}$$

That gives a Utilitarian Welfare of $UW(A) = u_1(A) + u_2(A) = 4 + 13 = 17$.

3.2.2 The Core

As discussed in Section 2.4, the Core is the set of allocations that is not dominated by another allocation. An allocation A^* dominates an allocation A if:

1. there is a coalition P that prefers A^* to A, and
2. this preference is "not idle", i.e. P can actually achieve at least its portion of A^*

In the model studied in this research, the first condition means that all players in P prefer A^* to A. The second condition means that the players in P do not need any help from players outside P to achieve A^*. In other words, all players outside the coalition P do no tasks in A^*. This can be summarized in the following two definitions.

Definition 3.2.2. An *allocation A^* dominates an allocation A* if there exists a coalition $P \subseteq N$ where $u_i(A^*) > u_i(A)$ for each $i \in P$ and $A_i^* = \emptyset$ for each $i \in N \setminus P$.

Definition 3.2.3. The *Core* consists of all allocations A which are not dominated by any other allocation A^*.

> ***Example*** *The example allocation as well as the allocation that maximized the Utilitarian Welfare are not in the Core. This is because there exists an allocation $A^* = (A_1^*, A_2^*)$ where:*
>
> $$A_1^* = \{T_1\}$$
> $$A_2^* = \emptyset$$
>
> *In this allocation $u_1(A^*) = g_{1,1} - c_{1,1} = 12 - 2 = 10$. Since $u_1(A^*) > u_1(A)$ and $A_2^* = \emptyset$, it holds for each $i \in N$ that either $u_i(A^*) > u_i(A)$ or $A_i^* = \emptyset$. Thus, A^* dominates A, which means A is not in the Core.*

3.2.3 The Epsilon Core

The ϵ-Core is similar to the Core, but a weaker criterion. The idea is that it costs players a value ϵ to collude. Therefore, if an allocation A^* dominates an allocation A but there is a player i that gains at most ϵ, A^* does not ϵ-dominate A. So although A is not in the Core, it might be in the ϵ-Core.

Definition 3.2.4. An *allocation A^* ϵ-dominates an allocation A* if there exists a coalition $P \subseteq N$ where $u_i(A^*) - \epsilon > u_i(A)$ for each $i \in P$ and $A_i = \emptyset$ for each $i \in N \setminus P$.

Definition 3.2.5. The *ϵ-Core* consists of all allocations A which are not ϵ-dominated by any other allocation A^*.

Example *Since $u_1(A^*) - u_1(A) = 6$, the allocation A^* ϵ-dominates the allocation A for $\epsilon < 6$. If $\epsilon \geq 6$, $u_i(A^*) - \epsilon > u_i(A)$ is not true, so A^* doesn't ϵ-dominate A for $\epsilon \geq 6$. Since there is no allocation that ϵ-dominates A for $\epsilon \geq 6$, A is in the ϵ-Core for $\epsilon \geq 6$.*

3.3 Conclusion

We have formalized the model, and defined optimization measures to value an allocation. We can now reason about how well an algorithm can guarantee an allocation to be, and whether it is possible to find such an allocation efficiently.

Chapter 4

Properties of the problem

Using the model from Chapter 3, in this chapter, we answer the research questions theoretically. First, we show maximizing the Utilitarian Welfare can be done efficiently. We then show that a solution in the Core does not always exist, and finding a solution in the Core is NP-hard. We also show a lower and an upper bound on the minimum ϵ for which an ϵ-Core always exist. Finally, we show that when we optimize the Utilitarian Welfare, no guarantee can be given for the stability and vice versa.

4.1 Maximizing Utilitarian Welfare

Since each player's utility is a sum of gains and costs for individual tasks, maximizing the Utilitarian Welfare can be done by maximizing the Utilitarian Welfare for each task. The Utilitarian Welfare of task j is $\sum_{i \in N} g_{ij} - c_{kj}$ when it is assigned to player k (each player gains the value, and player k pays the cost) and 0 if it is not assigned (nobody gains the value, and nobody pays the cost). For each task, there are $n + 1$ choices we can make. We can assign the task to any of the players, or we can decide not to assign the task. If we decide to assign the task, since $\sum_{i \in N} g_{ij}$ is not dependent on the player we assign it to k, we can maximize the Utilitarian Welfare by minimizing the cost c_{kj}. This is done by assigning it to the player who has the lowest cost for that task. If that makes the Utilitarian Welfare bigger than 0, assigning it to that player gives us the maximum Utilitarian Welfare. If it isn't, it gives more Utilitarian Welfare to not assign the task. Doing this for each task gives the solution with the maximum Utilitarian Welfare. This algorithm is formalized in Algorithm 1.

Theorem 1. *Maximizing the Utilitarian Welfare can be done in $O(n \cdot t)$ time.*

Proof. This proof will show that Algorithm 1 maximizes the Utilitarian Welfare in $O(n \cdot t)$ time. We will first prove that the algorithm is optimal. Then, we will prove that it runs in $O(n \cdot t)$ time.

Algorithm 1 Maximize Utilitarian Welfare

Make a list, A, of empty sets with size n
for all $j \in T$ **do**
  $minCost \leftarrow \min_{i \in N} c_{ij}$
  $totalValue \leftarrow \sum_{i \in N} g_{ij}$
  if $totalValue > minCost$ **then**
    $k \leftarrow \arg\min_{i \in N} c_{ij}$.
    Add j to A_k
  end if
end for

The Utilitarian Welfare is defined as

$$UW(A) = \sum_{x \in N} u_x$$

where the utility of a player is defined as

$$u_x = \sum_{i \in N} \sum_{j \in A_i} g_{xj} - \sum_{j \in A_x} c_{xj}$$

This means, using substitution, the Utilitarian Welfare can be written as

$$UW(A) = \sum_{x \in N} \left(\sum_{i \in N} \sum_{j \in A_i} g_{xj} - \sum_{j \in A_x} c_{xj} \right)$$

By rewriting further, we have

$$\begin{aligned} UW(A) &= \sum_{x \in N} \left(\sum_{i \in N} \sum_{j \in A_i} g_{xj} - \sum_{j \in A_x} c_{xj} \right) \\ &= \sum_{x \in N} \sum_{i \in N} \sum_{j \in A_i} g_{xj} - \sum_{x \in N} \sum_{j \in A_x} c_{xj} \\ &= \sum_{x \in N} \sum_{i \in N} \sum_{j \in A_i} g_{xj} - \sum_{i \in N} \sum_{j \in A_i} c_{ij} \\ &= \sum_{i \in N} \sum_{j \in A_i} \left(\sum_{x \in N} g_{xj} - c_{ij} \right) \end{aligned}$$

Since A is a list of disjoint subsets of T, we have

$$\begin{aligned} UW(A) &= \sum_{i \in N} \sum_{j \in A_i} \left(\sum_{x \in N} g_{xj} - c_{ij} \right) \\ &\leq \sum_{j \in T} \max \left(\left(\sum_{x \in N} g_{xj} - \min_{i \in N} c_{ij} \right), 0 \right) \end{aligned}$$

Let A^* be the allocation given by Algorithm 1. In such an allocation A^*, $j \in A_k$ if and only if $\sum_{i \in N} g_{ij} > \min_{i \in N} c_{ij}$ and $k = \arg\min_{i \in N} c_{ij}$. This means that Algorithm 1 will give a Utilitarian Welfare of

$$\begin{aligned} UW(A^*) &= \sum_{i \in N} \sum_{j \in A_i} \left(\sum_{x \in N} g_{xj} - c_{ij} \right) \\ &= \sum_{j \in T \mid \sum_{i \in N} g_{ij} > \min_{i \in N} c_{ij}} \left(\sum_{x \in N} g_{xj} - \min_{i \in N} c_{ij} \right) \\ &= \sum_{j \in T} \max \left(\left(\sum_{x \in N} g_{xj} - \min_{i \in N} c_{ij} \right), 0 \right) \end{aligned}$$

Since the Utilitarian Welfare cannot be higher than this, Algorithm 1 gives the maximum Utilitarian Welfare.

Making a list of size n of empty sets costs $O(n)$ time. The for loop has t iterations. Calculating the minimum of n items, calculating the sum of n items and comparing those two costs $O(n)$ time. Adding j to a set takes $O(1)$ time. Since this is done t times, this takes $O(n \cdot t)$ time. This means that in total, Algorithm 1 takes $O(n \cdot t)$ time.

Since Algorithm 1 maximizes the Utilitarian Welfare and takes $O(n \cdot t)$ time, the Utilitarian Welfare can be maximized in $O(n \cdot t)$ time. □

4.2 Finding an allocation in the Core

Finding an allocation in the Core is a lot harder than finding the allocation with the maximum Utilitarian Welfare. First, we show that the Core can be empty. Then, we show that finding the Core is $NP - hard$.

4.2.1 Existence of the Core

Problem with 1 player

For the 1-player problems, it is trivial to see the Core is non-empty. The algorithm that maximizes the Utilitarian Welfare can optimize the utility of the player. Since the utility of the only player cannot be improved, there cannot exist a coalition that improves the utility of a subset of the players. Because of that, the algorithm always finds an allocation in the Core efficiently.

Theorem 2. *For problems where $n = 1$, there is always a solution in the Core.*

Proof. This proof will show that Algorithm 1 finds a solution that is in the Core.

We assume to the contrary that Algorithm 1 finds an allocation A that is dominated by another allocation A^*, and show a contradiction. Since there is only one player, there is only one possible coalition $\{N_1\}$. Since A is dominated

by A^* with coalition $\{N_1\}$, we know that $u_1(A) < u_1(A^*)$. Since there is only one player, $UW(A) = \sum_{x \in N} u_x(A) = u_1(A)$ and similarly, $UW(A^*) = u_1(A^*)$. This means $UW(A) < UW(A^*)$. Since this means that the Utilitarian Welfare of A is smaller than the Utilitarian Welfare of A^*, this contradicts Theorem 1, which states that Algorithm 1 finds an optimal allocation. Therefore, A is in the Core, so Algorithm 1 finds a solution in the Core. □

Problem with 2 players

Before we go further, it is good to realize some general properties of the problem.

Firstly, any allocation where one player receives less than 0 utility can not be in the Core. If a player does not like an allocation, that player could easily not do any tasks (other players can't do anything about that), which improves their utility to 0.

Lemma 3. *Any allocation A where for some $j \in N$, $u_j < 0$, is not in the Core.*

Proof. Consider the allocation $A^* = (A_1^*, ..., A_n^*)$ where for all $i \in N$, $A_i^* = \emptyset$. Since no tasks are done, $u_i(A^*) = 0$ for all i. Now, because $u_j(A) < 0$ and $u_j(A^*) = 0$ there exists a coalition $\{j\} \subseteq N$ where $u_j(A^*) > u_j(A)$, and $A_i^* = \emptyset$ for each $i \in N \setminus \{j\}$. This means A^* dominates A, so A is not in the Core. □

Secondly, if there exists an allocation A where the only allocations that dominate A are with the coalition that contains all players N, then there exists an allocation in the Core. This is because in the allocations that dominate A, everyone's utility improves. Since there were already no possibilities to form a coalition smaller than N, that will definitely still be true. Now, there might be more allocations that improve the utility of all players, but since there is a finite amount of allocations, there has to be one where it is not possible anymore to improve everyone's utility. Since the smaller coalitions could also not form, no coalition can form, so that allocation is in the Core.

Lemma 4. *If there exists an allocation A that is not dominated by any allocation A^* with coalition $P \subset N$, then the Core is non-empty*

Proof. Consider the allocation A. A can either be in the Core or not. If A is in the Core, the Core is non-empty. If A is not in the Core, then some allocations dominate A. Since we assumed there was no allocation that dominates A with coalition $P \subset N$, the allocations that dominate A do so with coalition N.

Consider the allocation A^* that dominates A with coalition N with the highest total utility $\sum_{i \in N} u_i$. We show A^* is in the Core by showing A^* can neither be dominated by another allocation with a coalition N nor by another allocation with coalition $P \subset N$.

If there would exist an allocation A' which dominates A^* with coalition $P = N$, then $u_i(A') > u_i(A^*)$ for each $i \in N$. This would mean $\sum_{i \in N} u_i(A') > \sum_{i \in N} u_i(A^*)$. Since A^* dominates A with coalition $P = N$, $\sum_{i \in N} u_i(A^*) > \sum_{i \in N} u_i(A)$, which means $\sum_{i \in N} u_i(A') > \sum_{i \in N} u_i(A)$. Now, A' dominates A with coalition N, and $\sum_{i \in N} u_i(A') > \sum_{i \in N} u_i(A^*)$, which contradicts the fact

that A^* is the allocation that dominates A with coalition N with the highest total utility. By this contradiction A' cannot exist.

If there existed an allocation A' which dominates A^* with coalition $P \subset N$, then $u_i(A') > u_i(A^*)$ for all $i \in P$ and $A'_i = \emptyset$ for all $i \in N \setminus P$. Since A^* dominates A with coalition N, we know $u_i(A^*) > u_i(A)$ for all $i \in N$. This means $u_i(A') > u_i(A)$ for all $i \in P$ and $A'_i = \emptyset$ for all $i \in N \setminus P$. Because of this, A' dominates A with coalition $P \subset N$, which is a contradiction to the original assumption that such a coalition did not exist. By this contradiction A' cannot exist.

By these contradictions, we have shown either A is in the Core, or an allocation A^* that dominates A with coalition N is in the Core, so the Core is non-empty. □

The 2-player problem is a bit more difficult than the 1-player problem. Algorithm 1 can optimize the utility of each player with the restriction that the other player does no tasks. Now, there exists an allocation where for each task, it is allocated to the player that it was allocated to in the individual optimizations of the players, or to an arbitrary player if it was allocated to both players. In this outcome, both players gain at least as much utility as they did in their individual optimization. That means that the coalitions that consist of 1 player cannot achieve a higher utility where the other player performs no tasks. And by Lemma 4, this means that there exists an allocation in the Core.

Theorem 5. *For problems where $n = 2$, there is always a solution in the Core.*

Proof. This proof will show that in a two player problem, there always exists a solution in the Core.

Consider the two allocations $A^1 = (A^1_1, A^1_2)$, where $u_1(A^1)$ is optimized with the restriction that $A^1_2 = \emptyset$, and $A^2 = (A^2_1, A^2_2)$, where $u_2(A^2)$ is optimized with the restriction that $A^2_1 = \emptyset$. Now consider the allocation $A = (A_1, A_2)$ where for each $j \in T$, $j \in A_1$ if $j \in A^1_1$, and $j \in A_2$ if $j \notin A^1_1$ and $j \in A^2_2$. We now know

$$\begin{aligned} u_1(A^1) &= \sum_{j \in A^1_1} g_{1,j} + \sum_{j \in A^1_2} g_{1,j} - \sum_{j \in A^1_1} c_{1,j} \\ &= \sum_{j \in A_1} g_{1,j} + 0 - \sum_{j \in A_1} c_{1,j} \\ &\leq \sum_{j \in A_1} g_{1,j} + \sum_{j \in A_2} g_{1,j} - \sum_{j \in A_1} c_{1,j} = u_1(A) \end{aligned}$$

$$
\begin{aligned}
u_2(A^2) &= \sum_{j \in A_1^2} g_{2,j} + \sum_{j \in A_2^2} g_{2,j} - \sum_{j \in A_2^2} c_{2,j} \\
&= 0 + \sum_{j \in A_2^2} g_{2,j} - \sum_{j \in A_2^2} c_{2,j} \\
&\leq \sum_{j \in A_1 \cup A_2} g_{2,j} - \sum_{j \in A_2} c_{2,j} = u_2(A)
\end{aligned}
$$

Since $u_1(A^1)$ was the highest utility for N_1 given that $A_2^1 = \emptyset$, for any allocation A^* where $A_2^* = \emptyset$, we know $u_1(A^*) \leq u_1(A^1) \leq u_1(A)$. This means there does not exist an allocation A^* that dominates A with coalition $\{N_1\}$. Similarly, there does not exist an allocation A^* that dominates A with coalition $\{N_2\}$. Since $\{N_1\}$ and $\{N_2\}$ are the only strict subsets of N, by Lemma 4, there exists a solution in the Core. □

Problem with more than 2 players

Tasks that cannot be done by a certain player For simplicity, some instance examples are missing some values c_{ij} for some players i and tasks j. This means that player i cannot be assigned task j. Note that this simplification does not change the results. If there would have to be values there, the missing values could just be very large negative numbers. That would mean that if player i was assigned task j, player i would have a negative utility, which means such an allocation is not in the Core (by Lemma 3). We denote those missing values with a "$-$".

An allocation is in the Core if there does not exist a coalition where all players in the coalition can improve their utility without the help of the rest of the players. This means that in order for a Core not to exist, all allocations need to have a coalition where all players in the coalition can improve their utility without the rest of the players. You can visualize all possible allocations as nodes in a graph, where directed edges represent allocations dominating other allocations. Sink nodes then correspond to allocations in the Core. If there do not exist sink nodes, then there has to be at least 1 directed cycle. These cycles are pretty unintuitive. As seen in Theorem 5, an algorithm can combine the optimal allocations for individual players to make an allocation where no individual player can improve upon. If you improve upon that allocation, no individual, but also not the full group can improve upon that allocation, so it is in the Core. The only potential problem lies in the fact that there could still be coalitions with a size bigger than 1 and smaller than n.

Intuitively, you could think you would always be able to find an allocation in the Core with a recursive strategy. For 2 players, you could combine optimal allocations for both players individually. For 3 players, you could combine the allocations in the Core for each group of 2 players, and so on. Since your utility cannot decrease if more tasks get done, it does not seem to be a problem. However, there is one problem that the example from Table 4.1 shows.

Table 4.1: Combining trades does not work

	Utility from task done		
	T_1	T_2	T_3
N_1	3	0	0
N_2	0	3	0
N_3	0	0	3
	Disutility for doing task		
	T_1	T_2	T_3
N_1	–	-2	-2
N_2	-2	–	–
N_3	-2	–	–

The allocation in the Core for N_1 and N_2 is that N_1 performs T_2 and N_2 performs T_1. They both gain 3 and pay 2 with that, gaining a net utility of 1. The allocation in the Core for N_1 and N_3 is that N_1 performs T_3 and N_3 performs T_1. Similarly, they both gain a net utility of 1. In this example specifically, combining these allocations doesn't give a satisfying result as N_1 cannot get the value for T_1 twice. In the allocations with 2 players, T_1 kind of trades tasks with the other player, both gaining 1 in the process. However, with 3 players, T_1 cannot gain from more than 1 trade because N_2 and N_3 both offer the same task. Because of this, the combined allocation is not in the Core.

In this example, there is a different allocation that is in the Core, but it does show that combining doesn't work if 2 players can do the same task. The actual example with an empty Core is similar to an unstable example in social choice. Say three people N_1, N_2 and N_3 have to choose between three alternatives A^1, A^2 and A^3, and the ability to form a coalition and change the outcome if 2 players would rather have a different outcome, with preferences as the following.

$$A^1 \succ_{N_1} A^2 \succ_{N_1} A^3$$
$$A^2 \succ_{N_2} A^3 \succ_{N_2} A^1$$
$$A^3 \succ_{N_3} A^1 \succ_{N_3} A^2$$

In this example, if the outcome is A^1, N_2 and N_3 would rather have A^3. If the outcome is A^2, N_1 and N_3 would rather have A^1. If the outcome is A^3, N_1 and N_2 would rather have A^2. This means that for each outcome, there is a coalition that wants another outcome and is able to get it (because they are the majority). If a similar example exists in the model of this research, then the Core can be empty. The example in Table 4.2 is such a similar example.

Again, the allocation in the Core for N_1 and N_2 is that N_1 performs T_2 and N_2 performs T_1. This gains N_1 a net utility of 2 and N_2 a net utility of 1. The

Table 4.2: Example of an instance with allocations forming a cycle

	Utility from task done		
	T_1	T_2	T_3
N_1	101	0	0
N_2	0	101	0
N_3	0	0	101
	Disutility for doing task		
	T_1	T_2	T_3
N_1	$-$	-99	-100
N_2	-100	$-$	-99
N_3	-99	-100	$-$

allocation in the Core for N_1 and N_3 is very similar and gains N_1 a net utility of 1 and N_3 a net utility of 2. And finally, the allocation in the Core for N_2 and N_3 gains N_2 a net utility of 2 and N_3 a net utility of 1. The effect of this is that N_1 would rather trade with N_2, N_2 would rather trade with N_3 and N_3 would rather trade with N_1. However, because of the symmetry, there exists another allocation where N_1 performs T_2, N_2 performs T_3 and N_3 performs T_1, gaining them all 2, making that the allocation in the Core.

To finally construct the problem with the empty Core, we have to break the symmetry and make the values of the tasks different. To make the trades fair again, we have to introduce tasks that act as side payments. Such tasks effectively allow a player i to pay a player k x utility.

Definition 4.2.1. We say that *player i can pay player j x utility* if there exists a task t for which $g_{j,t} = x$, $g_{k,t} = 0$ for each $k \in N \setminus j$, $c_{i,t} = x$ and $c_{k,t} = -$ for each $k \in N \setminus i$.

If we make the values of the tasks different, and introduce the ability for each player to give exactly the difference between the value of their task and the task of another player to that other player, the symmetry is broken, and there will not exist an allocation in the Core.

Theorem 6. *For instances with $n \geq 3$, the Core can be empty.*

Proof. This proof uses an example of an instance, with some observations that show the instance has an empty Core.

The instance from Table 4.3 has an empty Core. This can be verified with a few observations.

First of all, the maximum Utilitarian Welfare is 6, which can be verified with Algorithm 1. This means that whenever a player receives more than 6 in an allocation, someone else in that allocation receives less than 0. From this

Table 4.3: Instance with an empty Core

	Utility from task done					
	T_1	T_2	T_3	T_4	T_5	T_6
N_1	121	0	0	0	0	0
N_2	0	111	0	10	0	0
N_3	0	0	101	0	10	20
	Disutility for doing task					
	T_1	T_2	T_3	T_4	T_5	T_6
N_1	–	-109	-100	-10	–	-20
N_2	-120	–	-99	–	-10	–
N_3	-119	-110	–	–	–	–

follows that any allocation that gives more than 6 or less than 0 utility to any player is not in the Core by Lemma 3.

Let's first look at all possible allocations that can give N_1 a utility of at least 0 and at most 6. This can be done in 3 ways.

Somebody else does T_1, N_1 does T_2 and T_4 In this case, let's look at what utility N_2 receives. Since N_1 does T_2 and T_4, N_2 gains 121 utility. The only way to have at least 0 and at most 6 utility is by doing only T_1. Since T_3, T_5 and T_6 cannot be done by N_3, N_3 gains no utility, so the only way to gain at least 0 utility is by doing no task. This leaves only the allocation $A^1 = (\{T_2, T_4\}, \{T_1\}, \emptyset)$.

Somebody else does T_1, N_1 does T_3 and T_6 In this case, let's look at what utility N_3 receives. Since N_1 does T_4 and T_6, N_3 gains 101 utility. The only way to have at least 0 and at most 6 utility is by doing only T_1. Since T_2 and T_4 cannot be done by N_2, N_2 gains no utility, so the only way to gain at least 0 utility is by doing no task. This leaves only the allocation $A^2 = (\{T_3, T_6\}, \emptyset, \{T_1\})$.

Nobody does T_1, N_1 does no task In this case, let's look at what utility N_2 receives. Since N_1 was the only player able to do task T_4 and did not do it, N_2 either only gains 111 from T_2 or noting at all.

If N_2 gains 111 from T_2, the only way to have at least 0 and at most 6 utility is by doing T_3 and T_5. In that case, N_3 gains 111 utility, so the only way to gain at least 0 and at most 6 utility is by doing task T_2. This leaves only the allocation $A^3 = (\emptyset, \{T_3, T_5\}, \{T_2\})$.

If N_2 does nothing at all, the only way to have at most 6 utility is if nobody does T_2. Since T_1 and T_2 are done by nobody and N_3 cannot do T_3, T_4, T_5

and T_6, N_3 also does no task in this case. This leaves only the allocation $A^4 = (\emptyset, \emptyset, \emptyset)$.

The utilities for each player for each allocation are shown in Table 4.4.

Table 4.4: Utilities for each allocation

	A^1	A^2	A^3	A^4
N_1	2	1	0	0
N_2	1	0	2	0
N_3	0	2	1	0

Since N_2 and N_3 have a higher utility in A_3 than in A_4, and N_1 does no task in A_3, A_3 dominates A_4. Also, N_1 and N_3 have a higher utility in A_2 than A_3 and N_2 does no task in A_2, so A_2 dominates $A3$. Similarly, A_1 dominates A_2 and A_3 dominates A_1. Since every allocation is dominated by another allocation, the Core is empty. □

4.2.2 Complexity of finding the Core

Since we would like an algorithm that finds an allocation in the Core, we need to know how hard the problem is to find an allocation in the Core. However, we will first look at an intuitively easier problem. Given an allocation, we would like to know if it is in the Core.

Definition 4.2.2. Given an instance with gain table G, cost table C and an allocation A, the *Allocation-In-Core problem* (G, C, A) is to determine if the given allocation is in the Core.

To prove this is an NP-hard problem, we reduce the Subset-Sum problem to this problem in polynomial time. The idea is that in allocation A, N_1, N_2 and N_3 cooperate with each other to all gain utility. Now, N_1 and N_2 can also cooperate together to gain more utility between them in another allocation A^*. However, this utility is not fairly distributed for N_1, so N_1 has no incentive to do this. There is also a number of tasks with which N_2 can pay N_1. Only if there exists a set of those tasks that allows N_2 to pay exactly some value to N_1 can they redistribute utility such that A^* dominates A. This is essentially the Subset-Sum problem, and since that is an NP-hard problem, this problem is NP-hard as well.

Theorem 7. *For instances with $n \geq 3$, the Allocation-In-Core problem is NP-hard.*

Proof. This proof is by reduction from the Subset-Sum problem. We reduce the Subset-Sum problem in polynomial time to this Allocation-In-Core problem. If the Allocation-In-Core problem would not be $NP - hard$, the Subset-Sum problem would then also not be $NP - hard$. Since that is a contradiction, the Allocation-In-Core problem must also be $NP - hard$.

Reduction

Given an instance of the Subset-Sum problem $I = (S, W) = (\{w_1, ..., w_n\}, W)$ with W and all $w \in S$ natural numbers. Construct the instance of the Allocation-In-Core problem $f(I) = (G, C, A)$ with G and C according to Table 4.5, and $A = (A_1, A_2, A_3)$ where:

$$A_1 = \{t_{n+1}\}$$
$$A_2 = \{t_{n+2}\}$$
$$A_3 = \{t_{n+3}\}$$

Table 4.5: Reduced instance to prove determining if an allocation is in the Core is NP-hard

	Utility from task done					
	T_1	...	T_n	T_{n+1}	T_{n+2}	T_{n+3}
N_1	w_1	...	w_n	0	0	$W+2$
N_2	0	...	0	0	0	$W+2$
N_3	0	...	0	$W+2$	$W+2$	0
	Disutility for doing task					
	T_1	...	T_n	T_{n+1}	T_{n+2}	T_{n+3}
N_1	$-$	...	$-$	$-(W+1)$	$-$	$-2*W$
N_2	$-w_1$	...	$-w_n$	$-$	$-(W+1)$	$-$
N_3	$-$	...	$-$	$-$	$-$	$-2*W$

We can calculate the utilities of each player for allocation A:

$$u_1(A) = g_{1,n+1} + g_{1,n+2} + g_{1,n+3} - c_{1,n+1} = 0 + 0 + (W+2) - (W+1) = 1$$
$$u_2(A) = g_{2,n+1} + g_{2,n+2} + g_{2,n+3} - c_{2,n+2} = 0 + 0 + (W+2) - (W+1) = 1$$
$$u_3(A) = g_{3,n+1} + g_{3,n+2} + g_{3,n+3} - c_{3,n+3} = (W+2) + (W+2) + 0 - (2 \cdot W) = 4$$

Correctness of the reduction

We will prove the reduction is correct by showing that I has a subset that sums to W if and only if A from $f(I)$ is not in the Core.

If I has a subset that sums to W then allocation A from $f(I)$ is not in the Core Let $I = (S, W)$ be an arbitrary instance of Subset-Sum where there exists a subset $X \subseteq S$ which sum is W. Consider the instance $f(I) = (G, C, A)$ of Solution-In-Core. We show that A is not in the Core by constructing an allocation A^* that dominates A.

Consider the allocation $A^* = (A_1^*, A_2^*, A_3^*)$ where:

$$A_1^* = \{t_{n+3}\}$$
$$A_2^* = \{t_j \in T \mid w_j \in X\}$$
$$A_3^* = \emptyset$$

The utilities of N_1 and N_2 for allocation A^* are:

$$u_1(A^*) = \sum_{j \in [1,n], w_j \in X} g_{1,j} + g_{1,n+3} - c_{1,n+3} = W + (W+2) - (2 \cdot W) = 2$$
$$u_2(A^*) = \sum_{j \in [1,n], w_j \in X} -c_{2,j} + g_{2,n+3} = -W + (W+2) = 2$$

Since $u_1(A^*) > u_1(A)$, $u_2(A^*) > u_2(A)$ and $A_3^* = \emptyset$, it holds for each player $i \in N$ that either $u_i(A^*) > u_i(A)$ or $A_i^* = \emptyset$. Because of this A^* dominates A, so A is not in the Core.

If allocation A from $f(I)$ is not in the Core then I has a subset that sums to W Let $f(I) = (G, C, A)$ be an instance of Solution-In-Core reduced from an arbitrary instance $I = (S, W)$ of Subset-Sum, where A is not in the Core. Since A is not in the Core, there exists an allocation A^* that dominates A. We are going to show with which coalition A^* dominates A, and that this means there exists a $X \subseteq S$ that sums to W.

There has to be a coalition A^* that dominates A. We are going to look at all possible coalitions and check what allocations can exist.

- For the coalition $\{N_1\}$: if N_2 and N_3 do no tasks, the maximum utility N_1 can get is when $W = 1$ with the allocation $A^* = (\{t_{n+3}\}, \emptyset, \emptyset)$: $u_1(A^*) = g_{1,n+3} - c_{1,n+3} = (W+2) - (2 \cdot W) = 2 - W = 1$. Since $u_1(A^*) \leq u_1(A)$, A^* does not dominate A.

- For the coalition $\{N_2\}$: if N_1 and N_3 do no tasks, the maximum utility N_2 can get is with the allocation $A^* = (\emptyset, \emptyset, \emptyset)$: $u_2(A^*) = 0$. Since $u_2(A^*) \leq u_2(A)$, A^* does not dominate A.

- For the coalition $\{N_3\}$: if N_1 and N_2 do no tasks, the maximum utility N_3 can get is with the allocation $A^* = (\emptyset, \emptyset, \emptyset)$: $u_3(A^*) = 0$. Since $u_3(A^*) \leq u_3(A)$, A^* does not dominate A.

- For the coalition $\{N_1, N_3\}$: if N_2 does no tasks, the maximum total utility N_1 and N_3 can get is when $W = 1$ with the allocation $A^* = (\{t_{n+1}\}, \emptyset, \{t_{n+3}\})$: $u_1(A^*) + u_3(A^*) = g_{1,n+1} + g_{1,n+3} + g_{3,n+1} + g_{3,n+3} - c_{1,n+1} - c_{3,n+3} = 0 + (W+2) + (W+2) + 0 - (W+1) - (2 \cdot W) = 3 - W = 2$. Since $u_1(A^*) + u_3(A^*) \leq u_1(A) + u_3(A)$, either $u_1(A^*) \leq u_1(A)$ or $u_3(A^*) \leq u_3(A)$, so A^* does not dominate A.

- For the coalition $\{N_2, N_3\}$: if N_1 does no tasks, the maximum total utility N_2 and N_3 can get is when $W = 1$ with the allocation $A^* = (\emptyset, \{t_{n+2}\}, \{t_{n+3}\})$: $u_2(A^*) + u_3(A^*) = g_{2,n+2} + g_{2,n+3} + g_{3,n+2} + g_{3,n+3} - c_{2,n+2} - c_{3,n+3} = 0 + (W+2) + (W+2) + 0 - (W+1) - (2 \cdot W) = 3 - W = 2$. Since $u_2(A^*) + u_3(A^*) \leq A(u_2) + A(u_3)$, either $u_2(A^*) \leq u_2(A)$ or $u_3(A^*) \leq u_3(A)$, so A^* does not dominate A.
- For the coalition $\{N_1, N_2, N_3\}$: the maximum total utility N_1, N_2 and N_3 can get is with the allocation $A^* = (\{t_{n+1}\}, \{t_{n+2}\}, \{t_{n+3}\})$: $u_1(A^*) + u_2(A^*) + u_3(A^*) = g_{1,n+1} + g_{1,n+2} + g_{1,n+3} + g_{2,n+1} + g_{2,n+2} + g_{2,n+3} + g_{3,n+1} + g_{3,n+2} + g_{3,n+3} - c_{2,n+2} - c_{3,n+3} = 0 + 0 + (W+2) + 0 + 0 + (W+2) + (W+2) + (W+2) + 0 - (W+1) - (W+1) - (2 \cdot W) = 6$. Since $u_1(A^*) + u_2(A^*) + u_3(A^*) \leq u_1(A) + u_2(A) + u_3(A)$, either $u_1(A^*) \leq u_1(A)$ or $u_2(A^*) \leq u_2(A)$ or $u_3(A^*) \leq u_3(A)$, so A^* does not dominate A.

Since an A^* that dominates A did not exist with all other possible coalitions, it has to exist with the coalition $\{N_1, N_2\}$. That means there has to exist an allocation A^* where $u_1(A^*) > 1$, $u_2(A^*) > 1$ and $A_3^* = \emptyset$. Since all numbers are natural, that has to mean $u_1(A^*) \geq 2$ and $u_2(A^*) \geq 2$, which means $u_1(A^*) + u_2(A^*) \geq 4$. This can only happen if $t_{n+3} \in A_1^*$, $t_{n+1} \notin A_1^*$ and $t_{n+2} \notin A_2^*$. Since that gives a maximum total utility of 4, we know $u_1(A^*) = u_2(A^*) = 2$. Then, we can formulate the next equation:

$$u_1(A^*) = \sum_{j \in [1,n], t_j \in A_2^*} g_{1,j} + g_{1,n+3} - c_{1,n+3} = \sum_{j \in [1,n], t_j \in A_2^*} g_{1,j} + (W+2) - (2 \cdot W) = 2$$

$$\sum_{j \in [1,n], t_j \in A_2^*} g_{1,j} = W$$

By the construction of the reduction, that means

$$\sum_{j \in [1,n], t_j \in A_2^*} w_j = W$$

Now, we can construct $X = \{w_j \in S \mid t_j \in A_2^*\}$ where $\sum_{w_j \in X} w_j = W$. Because of this, X is a solution to the Subset-Sum problem.

Polynomial time

Creating 3 players and $n+3$ tasks takes $O(|S|)$ time. Assigning gains and costs for all tasks $t_1, ..., t_n$ takes $O(|S|)$ time. Assigning gains and costs for tasks $t_{n+1}, t_{n+2}, t_{n+3}$ takes $O(1)$ time. Creating the allocation takes $O(|S|)$ time. In total, this takes $O(|S|)$ time, which is polynomial.

□

Now, we will look at the problem to determine if an instance has a non-empty Core.

Definition 4.2.3. Given an instance with gain table G and cost table C, the *Non-Empty-Core problem* (G, C) is to determine if the instance has a non-empty Core.

To prove this is an NP-hard problem, we reduce the Subset-Sum problem to this problem in polynomial time. The reduced instance is very similar to the instance from Table 4.3. There are three allocations where in each allocation, two players can collude to improve their utility, decreasing the utility of the third. However, there is a fourth player N_4 who can cooperate with N_1. This cooperation would mean that N_1 has no incentive anymore to collude with N_3, which means the allocation where that happens is in the Core. This cooperation is only possible if N_4 can pay some exact value to N_1. Similar to the proof of Theorem 7, this makes it so that you essentially solve a Subset-Sum problem to solve this, meaning the Non-Empty-Core problem is NP-hard.

Theorem 8. *For instances with $n \geq 4$, the Non-Empty-Core problem is NP-Hard.*

Proof. This proof is by reduction from the Subset-Sum problem. We reduce the Subset-Sum problem in polynomial time to this Non-Empty-Core problem. If the Non-Empty-Core problem would not be $NP - hard$, the Subset Sum problem would then also not be $NP - hard$. Since that is a contradiction, the Non-Empty-Core problem must also be $NP - hard$.

Reduction

Given an instance of the Subset Sum problem $I = (S, W) = (\{w_1, ..., w_n\}, W)$ with W and all $w \in S$ natural numbers. Construct the instance of the Non-Empty-Core problem $f(I) = (G, C)$ according to Table 4.6.

Correctness of the reduction

We will prove the reduction is correct by proving that if and only if I has a subset that sums to W, then $f(I)$ has a non-empty Core.

If I has a subset that sums to W then $f(I)$ has a non-empty Core Let $I = (S, W)$ be an arbitrary instance of Subset-Sum where there exists a subset $X \subseteq S$ which sum is W. Consider the instance $f(I) = (G, C)$ of Non-Empty-Core. We show that there exists an allocation in the Core A.

Consider the allocation $A = (A_1, A_2, A_3, A_4)$ where:

$$
\begin{aligned}
A_1 &= \{T_{n+7}\} \\
A_2 &= \{T_{n+3}, T_{n+5}\} \\
A_3 &= \{T_{n+2}\} \\
A_4 &= \{T_{n+1}\} \cup \{T_j \in T \mid w_j \in X\}
\end{aligned}
$$

Table 4.6: Reduced instance to prove determining if the Core is empty is NP-hard

	Utility from task done									
	T_1	...	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}	T_{n+7}
N_1	w_1	...	w_n	121	0	0	0	0	0	0
N_2	0	...	0	0	111	0	10	0	0	0
N_3	0	...	0	0	0	101	0	10	20	0
N_4	0	...	0	0	0	0	0	0	0	$W + 120$
	Disutility for doing task									
	T_1	...	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}	T_{n+7}
N_1	$-$	...	$-$	$-$	-109	-100	-10	$-$	-20	$-$
N_2	$-$	...	$-$	-120	$-$	-110	$-$	-10	$-$	$-$
N_3	$-$	...	$-$	-119	-110	$-$	$-$	$-$	$-$	$-$
N_4	$-w_1$	...	$-w_n$	-120	$-$	$-$	$-$	$-$	$-$	$-$

This means:

$$\begin{aligned} u_1(A) &= \sum_{j \in [1,n], w_j \in X} g_{1,j} + g_{1,n+1} - c_{1,n+7} \\ &= W + 121 - (W + 120) = 1 \end{aligned}$$

$$\begin{aligned} u_2(A) &= g_{2,n+2} - c_{2,n+3} - c_{2,n+5} \\ &= 111 - 99 - 10 = 2 \end{aligned}$$

$$\begin{aligned} u_3(A) &= g_{3,n+3} + g_{3,n+5} - c_{3,n+2} \\ &= 101 + 10 - 110 = 1 \end{aligned}$$

$$\begin{aligned} u_4(A) &= g_{4,n+7} - \sum_{j \in [1,n], w_j \in X} c_{4,j} - c_{4,n+7} \\ &= W + 120 - W - 120 = 0 \end{aligned}$$

We are going to show that there exists no allocation A^* that dominates A via every possible coalition.

In an allocation A^*, a coalition P must exist where for each $i \in P$, $u_i(A^*) > u_i(A)$ and for each $i \in N \setminus P$, $A^*_i = \emptyset$. Since all values are natural numbers, the first condition means that $u_i(A^*) \geq u_i(A) + 1$ for each $i \in P$, which also means that $\sum_{i \in P} u_i(A^*) \geq u_i(A) + |P|$.

With the algorithm to find the maximum Utilitarian Welfare, we can find the maximum $\sum_{i \in P} u_i(A^*)$ given that $A_i^* = \emptyset$ for every $i \in N \setminus P$.

As shown in Table 4.7, $\sum_{i \in P} u_i^* \not\geq \sum_{i \in P} u_i + |P|$ for every possible coalition. Therefore, we know A^* cannot exist, so A is in the Core.

Table 4.7: Comparison of the utility of players in a coalition for allocation A and any other allocation

Coalition P	Maximum total utility in coalition Maximum $\sum_{i \in P} u_i^*$	Total utility of coalition in A $\sum_{i \in P} u_i + \lvert P \rvert$
$\{N_1\}$	0	$1 + 1 = 2$
$\{N_2\}$	0	$2 + 1 = 3$
$\{N_3\}$	0	$1 + 1 = 2$
$\{N_4\}$	0	$0 + 1 = 1$
$\{N_1, N_2\}$	3	$3 + 2 = 5$
$\{N_1, N_3\}$	3	$2 + 2 = 5$
$\{N_1, N_4\}$	1	$1 + 2 = 3$
$\{N_2, N_3\}$	3	$3 + 2 = 5$
$\{N_2, N_4\}$	0	$2 + 2 = 4$
$\{N_3, N_4\}$	0	$1 + 2 = 3$
$\{N_1, N_2, N_3\}$	6	$4 + 3 = 7$
$\{N_1, N_2, N_4\}$	3	$3 + 3 = 6$
$\{N_1, N_3, N_4\}$	3	$2 + 3 = 5$
$\{N_2, N_3, N_4\}$	3	$3 + 3 = 6$
$\{N_1, N_2, N_3, N_4\}$	6	$4 + 4 = 8$

If $f(I)$ has a non-empty Core then I has a subset that sums to W Let $f(I) = (G, C)$ be an instance of Non-Empty-Core reduced from an arbitrary instance $I = (S, W)$ of Subset-Sum, which has a non-empty Core. This means there exists an allocation in the Core A. We show that there exists a subset $X \subseteq S$ which sum is W.

First of all, the maximum utilitarian utility is 6. This means that whenever a player (or a combination of players) receive(s) more than 6 in an allocation, someone else in that allocation receives less than 0, which means that allocation is not in the Core by Lemma 3.

Let's first look at all possible allocations A that can give N_2 a utility of at least 0 and at most 6. This can be done in 3 ways.

Somebody else does T_{n+2} and T_{n+4}, and N_2 does T_{n+1} but doesn't do T_{n+3} and T_{n+5} In this case, let's look at what utility N_1 and N_4 combined receive. Since N_2 does T_{n+1}, N_1 and N_4 combined gain 121 utility. Since only N_1 can do T_{n+4}, and we assumed T_{n+4} is done, the only way to have at least

0 and at most 6 utility is if N_1 does T_{n+2} and T_{n+4} and doesn't do T_{n+3} and T_{n+6}. Since T_{n+3} is neither done by N_1 nor by N_2, N_3 gains no utility, so the only way to gain at least 0 utility is by doing no task. Since we established the assignments of $T_{n+1}, ..., T_{n+6}$ and the other tasks don't influence N_2 and N_3, we can already determine their utilities:

$$\begin{aligned} u_2(A) &= g_{2,n+2} + g_{2,n+4} - c_{2,n+1} \\ &= 111 + 10 - 120 = 1 \end{aligned}$$

$$u_3(A) = 0$$

Now consider the allocation $A^* = (\emptyset, \{T_{n+3}, T_{n+5}\}, \{T_{n+2}\}, \emptyset)$. In this allocation,

$$\begin{aligned} u_2(A^*) &= g_{2,n+2} - c_{2,n+3} - c_{2,n+5} \\ &= 111 - 99 - 10 = 2 \end{aligned}$$

$$\begin{aligned} u_3(A^*) &= g_{3,n+3} + g_{3,n+5} - c_{3,n+2} \\ &= 101 + 10 - 110 = 1 \end{aligned}$$

The utilities of N_2 and N_3 are higher, and N_1 and N_4 perform no tasks, so the allocation will not be in the Core. That means that there cannot be an allocation in the Core under this assumption, which means this assumption was incorrect.

Nobody does T_{n+2} nor T_{n+4}, N_2 does no task In this case, let's look at what utility N_3 receives. Since N_2 was the only player able to do task T_{n+5} and did not do it, N_3 either gains 121 from T_{n+3} and T_{n+6} while costing 119 from T_{n+1} or gains nothing at all while also not doing a task.

In the first case, since we established the assignments of $T_{n+1}, ..., T_{n+6}$ and the other tasks don't influence the utility of the combination N_1 and N_4, we can already determine their total utility:

$$\begin{aligned} u_1(A) + u_4(A) &= g_{1,n+1} - c_{1,n+3} - c_{1,n+6} \\ &= 121 - 100 - 20 = 1 \end{aligned}$$

In the second case, since we established the assignments of $T_{n+2}, ..., T_{n+6}$ and the other tasks except for T_{n+1} don't influence the utility of the combination N_1

and N_4, their utility is only dependent on if T_{n+1} is done by N_4 or unassigned. Their combined utility is respectively:

$$\begin{aligned} u_1(A) + u_4(A) &= g_{1,n+1} - c_{4,n+1} \\ &= 121 - 120 = 1 \end{aligned}$$

$$u_1(A) + u_4(A) = 0$$

In all these cases, their combined utility is at most 1, which means that $u_1(A)$ is at most 1 in any allocation in the Core by Lemma 3. Also, since no task was done that gained N_2 anything and N_2 did no task,

$$u_1(A) \leq 1$$

$$u_2(A) = 0$$

Now consider the allocation $A^* = (\{T_{n+2}, T_{n+4}\}, \{T_{n+1}\}, \emptyset, \emptyset)$. In this allocation,

$$\begin{aligned} u_1(A^*) &= g_{1,n+1} - c_{1,n+2} - c_{2,n+4} \\ &= 121 - 109 - 10 = 2 \end{aligned}$$

$$\begin{aligned} u_2(A^*) &= g_{2,n+2} + g_{2,n+4} - c_{2,n+1} \\ &= 111 + 10 - 120 = 1 \end{aligned}$$

The utilities of N_1 and N_2 are higher, and N_1 and N_4 perform no tasks, so the allocation will not be in the Core. That means that there cannot be an allocation in the Core under this assumption, which means this assumption was incorrect.

Somebody else does T_{n+2}, nobody does T_{n+4}, and N_2 does T_{n+3} and T_{n+5} In this case, let's look at what utility N_3 receives. Since N_2 does T_{n+3} and T_{n+5}, N_3 gains 111 utility. The only way to have at least 0 and at most 6 utility is by doing T_{n+2} while nobody does T_{n+6}. Since we established the assignments of $T_{n+1}, ..., T_{n+6}$ and the other tasks don't influence N_3, we can already determine their utility:

$$\begin{aligned} u_3(A) &= g_{3,n+3} + g_{3,n+5} - c_{3,n+2} \\ &= 101 + 10 - 110 = 1 \end{aligned}$$

Now consider the allocation $A^* = (\{T_{n+3}, T_{n+6}\}, \emptyset, \{T_{n+1}\}, \emptyset)$. In this allocation,

$$\begin{aligned} u_1(A^*) &= g_{1,n+1} - c_{1,n+3} - c_{1,n+6} \\ &= 121 - 100 - 20 = 1 \end{aligned}$$

$$\begin{aligned} u_3(A^*) &= g_{3,n+3} + g_{3,n+6} - c_{3,n+1} \\ &= 101 + 20 - 119 = 2 \end{aligned}$$

Since we know the allocation is in the Core, it cannot be dominated by this allocation. Since $u_3(A^*) > u_3(A)$, and N_2 and N_4 perform no tasks, $u_1(A^*)$ cannot be higher than $u_1(A)$. Therefore, $u_1(A) \geq 1$.

By Lemma 3, we also know $u_4(A) \geq 0$. Therefore, $u_1(A) + u_4(A) \geq 1$. Since only T_{n+1} increases the utility of N_1 and N_4 combined, and T_{n+1} is not done by N_2 and N_3, it must be done by N_4. Since that costs N_4 utility and $u_4 \geq 0$, T_{n+7} must be done by u_1. Now,

$$\begin{aligned} u_1(A) + u_4(A) &= g_{1,n+1} - c_{4,n+1} \\ &= 121 - 120 = 1 \end{aligned}$$

Since $u_1(A) \geq 1$ and $u_4(A) \geq 0$ we know $u_1(A) = 1$ and $u_4(A) = 0$. The only thing we haven't established is the assignments of $T_1, ..., T_n$. Since only N_4 can do these, N_4 does some subset $Y \subseteq \{T_1, ..., T_n\}$. We can now establish

$$\begin{aligned} u_4(A) &= g_{4,n+7} - \sum_{j \in [1,n], T_j \in Y} c_{4,j} - c_{4,n+1} \\ &= (W + 120) - \sum_{j \in [1,n], T_j \in Y} w_j - 120 = 0 \end{aligned}$$

$$\sum_{j \in [1,n], T_j \in Y} w_j = W$$

Now, we can construct $X = \{w_j \in S \mid T_j \in Y\}$ where $\sum_{w_j \in X} w_j = W$. Because of this, X is a solution to the Subset Sum problem.

Polynomial time

Creating 4 players and $n + 7$ tasks takes $O(|S|)$ time. Assigning gains and costs for all tasks $t_1, ..., t_n$ takes $O(|S|)$ time. Assigning gains and costs for tasks $t_{n+1}, ..., t_{n+7}$ takes $O(1)$ time. In total, this takes $O(|S|)$ time, which is polynomial.

□

Finally, we will prove that finding the Core is also NP-hard. This follows trivially from Theorem 8

Definition 4.2.4. Given an instance with gain table G and cost table C, the *Find-Core problem* (G, C) is to find an allocation A in the Core.

Corollary 8.1. *For instances with $n \geq 4$, the Find-Core problem is NP-hard.*

Proof. This proof is by contradiction. We assume finding a solution in the Core can be done in polynomial time, and find a contradiction.

Let M be the algorithm that finds a solution in the Core in polynomial time. Now, Algorithm 2 is a polynomial time algorithm that determines if the Core is non-empty. It is correct because it outputs YES if and only if there is a solution in the Core. It runs in polynomial time, because M runs in polynomial time. Since this contradicts Theorem 8, the assumption cannot be true, so finding a solution in the Core is also NP-hard.

Algorithm 2 Determine if Core is non empty

```
Run M on I = (G, C)
if M finds a solution then
    return YES
else
    return NO
end if
```

□

4.3 The Epsilon Core

Now we know there might not be a Core, we cannot avoid a situation where players have an incentive to form a coalition. What we can do however, is to minimize that incentive. In other words, we want to minimize the utility players gain by joining a coalition. Since a coalition only works if all players in the coalition agree to it, we only have to minimize the difference in utility for one of the players, the player with the lowest difference. If this difference in an allocation is at most some value ϵ, we say the allocation is in the ϵ-Core (see Section 3.2.3). In this section, we discuss a bound on ϵ for which there always exists an ϵ-Core.

4.3.1 A bound on Epsilon

We would like to find a bound for this ϵ. However, we will never be able to find such a bound because by scaling the instance (multiplying all values of G and C with some constant), the maximum ϵ for which there exists a non-empty ϵ-Core scales the same way. Therefore, ϵ can become arbitrarily large.

We can also relate the epsilon to the utility a player already receives. However, with example from Table 4.8, we can establish that we will never find a bound this way as well. This can be seen as follows.

Table 4.8: Example showing that there is no bound on an epsilon relative to the utility of the player

	Utility from task done					
	T_1	T_2	T_3	T_4	T_5	T_6
N_1	$120 * X + 1$	0	0	0	0	0
N_2	0	$110 * X + 1$	0	$10 * X$	0	0
N_3	0	0	$100 * X + 1$	0	$10 * X$	$20 * X$
	Disutility for doing task					
	T_1	T_2	T_3	T_4	T_5	T_6
N_1	–	$-109 * X$	$-100 * X$	$-10 * X$	–	$-20 * X$
N_2	$-120 * X$	–	$-99 * X$	–	$-10 * X$	–
N_3	$-119 * X$	$-110 * X$	–	–	–	–

If any player gains a negative utility, they can increase their utility infinitely much by doing nothing. Therefore, such allocations are never in the Core for a certain bound. Like in the example from Theorem 6, the only allocations that give all players a non-negative value are $A^1 = (\{T_2, T_4\}, \{T_1\}, \emptyset)$, $A^2 = (\{T_3, T_6\}, \emptyset, \{T_1\})$, $A^3 = (\emptyset, \{T_3, T_5\}, \{T_2\})$ and $A^4 = (\emptyset, \emptyset, \emptyset)$. The utilities of the players in these allocations are as shown in Table 4.9.

Table 4.9: Utilities for each allocation

	A^1	A^2	A^3	A^4
N_1	X+1	1	0	0
N_2	1	0	X+1	0
N_3	0	X+1	1	0

For each allocation, there is another allocation where 2 players can increase their utility with some constant that cannot be bounded as X becomes infinitely

large, while the other player does nothing. Therefore, no bound can be found in this example.

In order to give such a bound, there has to be some maximum in the value a player can gain from the tasks. So given $MaxUtil$ is the maximum utility any player can get $MaxUtil = \max_{i \in N} \sum_{j \in T} G_{ij}$, we would like to find an allocation in the ϵ-Core where $\epsilon / MaxUtil$ is as low as possible.

Bounds on epsilon with respect to the maximum utility

A lower bound on the minimum $\epsilon / MaxUtil$ for which an ϵ-Core always exists can be proven with an example of an instance where there is no ϵ-Core for a certain $\epsilon / MaxUtil$. The example is very similar to the example from Theorem 6.

Theorem 9. *If $\epsilon / MaxUtil < 1/15$, the ϵ-Core can be empty.*

Proof. This proof uses an example of an instance. In this instance, for each $\epsilon < MaxUtil/15$, the ϵ-Core is empty. Because of this, the minimum $\epsilon / MaxUtil$ for which there always exists an ϵ-Core is $1/15$.

The example from Table 4.10 has an empty ϵ-Core for each $\epsilon < MaxUtil/15$. Since $MaxUtil = 15$, we have to proof each allocation is ϵ-dominated by another allocation for each $\epsilon < 1$. This can be verified with a few observations.

Table 4.10: Example showing the lower bound on epsilon with respect to the maximum utility

	Utility from task done					
	T_1	T_2	T_3	T_4	T_5	T_6
N_1	12	0	0	0	0	0
N_2	0	7	0	5	0	0
N_3	0	0	4	0	3	8
	Disutility for doing task					
	T_1	T_2	T_3	T_4	T_5	T_6
N_1	$-$	-5	-3	-5	$-$	-8
N_2	-11	$-$	-2	$-$	-3	$-$
N_3	-10	-6	$-$	$-$	$-$	$-$

First of all, any allocation that gives at most -1 utility to any player is not in the ϵ-Core, because the allocation where nobody does any task ϵ-dominates it. This is because that player could form a coalition and its utility would be 0, which is more than ϵ higher than -1, for each $\epsilon < 1$.

Let's first look at all assignments of task T_1.

N_2 does T_1 In this case, N_2 pays 11 utility. The only way to have more than -1 utility is if T_2 and T_4 are done, and N_2 does only T_1. Now, since N_1 has to

do T_4, he cannot do both T_3 and T_6. Since N_2 does not do T_3 and T_5, only one of T_3, T_5 and T_6 gets done. This means N_3 cannot do T_2. Therefore, N_1 has to do T_2. Since N_1 now does T_2 and T_4, he cannot do any other task. This leaves only the allocation $A^1 = (\{T_2, T_4\}, \{T_1\}, \emptyset)$.

N_3 does T_1 In this case, N_3 pays 10 utility. The only way to have more than -1 utility is if T_6 is done, and N_3 does not do T_2. Now, since N_1 has to do T_6, he cannot do T_2 or T_4. Since T_2 and T_4 are not done, N_2 can do no tasks. This means that for N_3 to get more than -1 utility, N_1 needs to do T_3. This leaves only the allocation $A^2 = (\{T_3, T_6\}, \emptyset, \{T_1\})$.

Nobody does T_1 In this case, N_1 gains no utility, so N_1 cannot do any task. Now, either N_3 does T_2 or nobody does T_2.

If N_3 does T_2, the only way for N_3 to gain more than -1 utility is for N_2 to do T_3 and T_5. This leaves only the allocation $A^3 = (\emptyset, \{T_3, T_5\}, \{T_2\})$.

If nobody does T_2, N_2 gains no utility, so N_2 cannot do any task. That means N_3 also gains no utility, so N_3 cannot do any task as well. This leaves only the allocation $A^4 = (\emptyset, \emptyset, \emptyset)$.

The allocations where each player has more than -1 utility are $A^1 = (\{T_2, T_4\}, \{T_1\}, \emptyset)$, $A^2 = (\{T_3, T_6\}, \emptyset, \{T_1\})$, $A^3 = (\emptyset, \{T_3, T_5\}, \{T_2\})$ and $A^4 = (\emptyset, \emptyset, \emptyset)$. The utilities of the players are shown in Table 4.11. Each allocation is dominated by an allocation via a coalition where all players can improve their utility by 1. Therefore, the smallest ϵ for which their exists an ϵ-Core is 1. Since $MaxUtil = 15$ (for player N_3), $\epsilon/MaxUtil = 1/15$. This means that if $\epsilon/MaxUtil < 1/15$, the ϵ-Core can be empty.

Table 4.11: Utilities for each allocation

	A^1	A^2	A^3	A^4
N_1	2	1	0	0
N_2	1	0	2	0
N_3	0	2	1	0

□

An upper bound of 1/2 on the minimum $\epsilon/MaxUtil$ for which an ϵ-Core always exist can be proven with some observations. Any instance without an ϵ-Core needs a cycle of dominating allocations. In this cycle, one player needs to be in two coalitions that are adjacent in this cycle. Otherwise, we would be able to combine the dominating allocations in the way discussed in Theorem 5 to get an allocation that would replace both allocations in the cycle. Since one player needs to increase his utility in both dominating allocations with ϵ, his utility in the last allocation must be at least $2 * \epsilon$. That means $2 * \epsilon \leq MaxUtil$.

Theorem 10. *If $\epsilon/MaxUtil \geq 1/2$, there exists an ϵ-Core.*

Proof. This proof is by contradiction. We assume there exists an instance for which the ϵ-Core for $\epsilon/MaxUtil = 1/2$ is empty and show a contradiction.

Given an arbitrary instance with an empty ϵ-Core for $\epsilon/MaxUtil = 1/2$. In the allocation $A = (\emptyset, ..., \emptyset)$, since nobody does any tasks, the utility of each player $i \in N$ is $u_i(A) = 0$. Since the ϵ-Core is empty, there has to exist an allocation that ϵ-dominates A. Let A^* be the allocation that ϵ-dominates A via the largest coalition P^*. Now, for each $i \in P^*$, $u_i(A^*) > MaxUtil/2$.

Now assume to the contrary that there does not exist an allocation A' that ϵ-dominates A^* with a coalition P' where there exists a player i that is both $i \in P'$ and $i \in P^*$. Since the ϵ-Core for $\epsilon = MaxUtil/2$ is empty, there has to exist an allocation that ϵ-dominates A^*. Let A' be any allocation that ϵ-dominates A^* via coalition P'. Now, let A'^* be the allocation where for each $i \in P^*$ and $j \in T$, $j \in A_i'^*$ if $j \in A_i^*$ and where for each $i \in P'$ and $j \in T$, $j \in A_i'^*$ if $j \in A_i'$ and $j \notin \cup_{k \in P^*} A_k^*$. Now, because P^* and P' are disjoint, for each $i \in P^*$

$$\begin{aligned} u_i(A'^*) &= \sum_{j \in \cup_{k \in P^* \cup P'} A_k'^*} g_{i,j} - \sum_{j \in A_i'^*} c_{i,j} \\ &= \sum_{j \in \cup_{k \in P^*} A_k^*} g_{i,j} + \sum_{j \in \cup_{k \in P'} A_k' | j \notin \cup_{k \in P^*} A_k^*} g_{i,j} - \sum_{j \in A_i^*} c_{i,j} \\ &\geq \sum_{j \in \cup_{k \in P^*} A_k^*} g_{i,j} - \sum_{j \in A_i^*} c_{i,j} = u_i(A^*) > MaxUtil/2 \end{aligned}$$

and for each $i \in P'$

$$\begin{aligned} u_i(A'^*) &= \sum_{j \in \cup_{k \in P^* \cup P'} A_k'^*} g_{i,j} - \sum_{j \in A_i'^*} c_{i,j} \\ &= \sum_{j \in \cup_{k \in P^*} A_k^*} g_{i,j} + \sum_{j \in \cup_{k \in P'} A_k' | j \notin \cup_{k \in P^*} A_k^*} g_{i,j} - \sum_{j \in A_i'} c_{i,j} \\ &\geq \sum_{j \in \cup_{k \in P'} A_k' | j \notin \cup_{k \in P^*} A_k^*} g_{i,j} - \sum_{j \in A_i'} c_{i,j} = u_i(A') > MaxUtil/2 \end{aligned}$$

Now, since $u_i(A'^*) > MaxUtil/2$ for each $i \in P^* \cup P'$ and $A_i'^* = \emptyset$ for each $i \in N \setminus P^* \setminus P'$, $A_i'^*$ ϵ-dominates A with $\epsilon = MaxUtil/2$ via coalition $P^* \cup P'$. This contradicts the fact that P^* is the largest coalition that ϵ-dominates A, which means the assumption that there does not exist an allocation A' that ϵ-dominates A^* with a coalition P' where there exists a player $i \in P'$ and $i \in P^*$ is incorrect.

Now, there exists an allocation A' that ϵ-dominates A^* with a coalition P' where there exists a player $i \in P'$ and $i \in P^*$. This means that for the player $i \in P'$ where $i \in P^*$ that $u_i(A') > u_i(A^*) + MaxUtil/2 > MaxUtil$. This contradicts the fact that $MaxUtil$ is the maximum utility of any player. Therefore, the original assumption that there exists an instance for which the ϵ-Core for

$\epsilon/MaxUtil = 1/2$ is empty is incorrect. This proves that if $\epsilon/MaxUtil \geq 1/2$, an ϵ-Core exists. □

4.4 Trade-off between Utilitarian Welfare and stability

Now that we know the limitations of the Utilitarian Welfare and the Core individually, we would like to know if we can find an allocation that performs decent in both measures. In this section, we show that when we maximize the Utilitarian Welfare, we cannot give a non-trivial bound on the minimum ϵ for which an ϵ-Core exists. Additionally, we show that when we find an allocation in the Core, we cannot give a non-trivial bound on the minimum Utilitarian Welfare.

We can show this with the instance from Table 4.12. In this instance, the allocation with the maximum Utilitarian Welfare is not in ϵ-Core for $\epsilon < Y$. Since Y is unbounded, no guarantee for ϵ can be given. The allocation in the Core has 0 Utilitarian Welfare while another allocation has Y Utilitarian Welfare. Since Y is unbounded, the Utilitarian Welfare can be arbitrarily worse if we require an allocation in the Core.

Table 4.12: Tradeoff between Utilitarian Welfare and stability

	Utility from task done
	T_1
N_1	$X + Y$
N_2	0
	Disutility for doing task
	T_1
N_1	$-$
N_2	$-X$

Theorem 11. *The lower bound on the minimum $\epsilon/MaxUtil$ for which the maximum Utilitarian Welfare allocation is in the ϵ-Core is 1.*

Proof. This proof uses the example of an instance from Table 4.12. In this instance, only 2 allocations exist, $A = (\emptyset, \{T_1\})$ and $A^* = (\emptyset, \emptyset)$. A is ϵ-dominated with coalition $\{N_2\}$ by A^* for any $\epsilon < X$. Since $MaxUtil = X + Y$ and X can be arbitrarily large, a maximum Utilitarian Welfare allocation can be outside an ϵ-Core for any $\epsilon/MaxUtil < 1$. Thus, the lower bound on the minimum $\epsilon/MaxUtil$ for which the maximum Utilitarian Welfare allocation is in the ϵ-Core is 1. □

Theorem 12. *The maximum Utilitarian Welfare allocation in the Core can have an arbitrarily worse Utilitarian Welfare than the maximum Utilitarian Welfare allocation.*

Proof. This proof uses the example of an instance from Table 4.12. In this instance, only 2 allocations exist, $A = (\emptyset, \{T_1\})$ and $A^* = (\emptyset, \emptyset)$. Maximizing the Utilitarian Welfare gives A, which gives a Utilitarian Welfare of Y. Since this allocation is dominated with coalition $\{N_2\}$ by allocation A^*, it is not in the Core. A^* is in the Core, so it is the maximum Utilitarian Welfare allocation in the Core, with a Utilitarian Welfare of 0. Since Y can be arbitrarily large, the maximum Utilitarian Welfare allocation in the Core can have an arbitrarily worse Utilitarian Welfare than the maximum Utilitarian Welfare allocation. □

4.5 Conclusion

To conclude, we have proven there does not always exist a solution in the Core. We might need to settle for a solution in some ϵ-Core. We know we can always find such a ϵ-Core for $\epsilon \geq MaxUtil/2$, and that we cannot guarantee a solution in the ϵ-Core for $\epsilon < MaxUtil/15$. Whether it is possible to guarantee solutions in the ϵ-Core for an ϵ between those values is a question for future work.

We have also concluded that the minimum $\epsilon/MaxUtil$ for which the maximum Utilitarian Welfare allocation is in the ϵ-Core can be arbitrarily close to 1, and that the Utilitarian Welfare of the maximum Utilitarian Welfare allocation in the Core can be arbitrarily worse than the Utilitarian Welfare of the maximum Utilitarian Welfare allocation.

Finally, we have concluded that maximizing Utilitarian Welfare can be done efficiently, but finding a solution in the Core cannot be done efficiently. Therefore, we will present a pseudo-polynomial algorithm and an approximation algorithm in Chapter 5.

Chapter 5

Algorithms

Now that we have shown that finding the Core is $NP - hard$, we know it is impossible to find a polynomial algorithm to find the Core. In this chapter, we propose two algorithms for finding the allocation in the Core with the maximum Utilitarian Welfare, and compare them to a brute force approach. One of the proposed algorithms is a pseudo-polynomial algorithm, the other is an approximation algorithm. Their advantages and limitations will be discussed, such as theoretical complexity and approximation bounds.

5.1 Brute force approach

A brute force approach to find the allocation in the Core with the maximum Utilitarian Welfare would be to check for each possible allocation whether it is in the Core. Since an allocation A is in the Core if there is no allocation that dominates it, we have to check that no allocation dominates A. Algorithm 3 shows this approach in an algorithm.

Algorithm 3 Brute force algorithm to find an allocation in the Core

```
MaxUWAllocation ← null
for all A do
    for all A* do
        if A* dominates A then
            Continue with the next A
        end if
    end for
    if UW(A) > UW(MaxUWAllocation) then
        MaxUWAllocation ← A
    end if
end for
return MaxUWAllocation
```

In an allocation, each task can be assigned to any player, or be left unassigned. This means there are $n+1$ choices for each task. Since there are t tasks, that makes $(n+1)^t$ possible allocations. Checking whether an allocation dominates another allocation can be done by checking each player's utility, which takes $O(n \cdot t)$. Since that is done in a nested for loop that loops over all possible alloctions, this brute force approach takes $O(n \cdot t \cdot (n+1)^{2 \cdot t})$ time.

5.2 Linear programming

Since the problem is to find an allocation that maximizes some function with certain constraints, it is intuitive to think about an integer linear programming solution. We find the allocation A that maximizes $UW(A) = \sum_{i \in N} u_i(A) = \sum_{i \in N} \left(\sum_{k \in N} \sum_{j \in A_k} g_{ij} - \sum_{j \in A_i} c_{ij} \right)$ given that A is in the Core. The constraint that A is in the Core can be formulated as $\cup_{i \in N} \left(u_i(A) \geq u_i(A^*) \cap A_i^* \neq \emptyset \right)$ for all A^*. Intuitively, for all allocations A^*, there has to be some player that does not prefer A^* but is assigned a task in A^*.

Since an integer linear program cannot handle OR-constraints, we have to rewrite the constraints with additional variables. This is shown in Algorithm 4.

Algorithm 4 Integer linear programming model

Maximize $\sum_{i \in N} \left(\sum_{k \in N} \sum_{j \in A_k} g_{ij} - \sum_{j \in A_i} c_{ij} \right)$ where:
for all A^* **do**
  for all $\{i \in N | A_i^* \neq \emptyset\}$ **do**
    $u_i(A) \geq u_i(A^*) \cdot x_i$
  end for
  $\sum_{\{i \in N | A_i^* \neq \emptyset\}} x_i \geq 1$
end for

The amount of possible A^* is $(n+1)^t$. The amount of constraints per A^* is $O(n)$. Therefore, the amount of constraints is $O((n+1)^{t+1})$. Since solving an integer linear program takes an exponential amount of time in itself, this will still be a pretty slow algorithm.

5.2.1 Reducing the amount of constraints

The linear program from before was very complex with its $O((n+1)^{t+1})$ constraints. However, a lot of these constraints are actually unnecessary. For example, if there is a constraint that $u_1(A) \geq 3 \cup u_2(A) \geq 5 \cup u_3(A) \geq 4$, then the constraint that $u_1(A) \geq 2 \cup u_2(A) \geq 3 \cup u_3(A) \geq 4$ is unnecessary because for any A where the first constraint is true, the second constraint is true as well (given that $A_1, A_2, A_3 \neq \emptyset$). Basically, we only need the Pareto front. Since it might be necessary if some $A_i = \emptyset$, we need to compute this Pareto front for every possible coalition.

Computing this Pareto front is a hard problem. However, given that n is constant, we can do it with a pseudo-polynomial algorithm, or with a polynomial approximation algorithm. Both algorithms consist of 2 parts. In the first part, a set of outcomes will be found. In the second part, these outcomes will be used as constraints in the discussed integer linear program to find the allocation in the Core with the maximum Utilitarian Welfare. In the pseudo-polynomial algorithm, the amount of constraints is dependent on the maximum gain and the amount of players. In the approximation algorithm, the amount of constraints is dependent on the precision of the approximation and the amount of players. Both algorithms are not dependent on the number of tasks, which allows the full algorithm to be (pseudo-)polynomial in the amount tasks.

Pseudo-polynomial algorithm

The first part of the pseudo-polynomial algorithm creates a set of outcomes, where an outcome is the utility of each player. It starts with the outcome where each player is assigned no task at all, which means each player gets 0 utility. Then, it iteratively adds all tasks. For each possible player to add the task to, it calculates the new utilities of all players if the task is added to that player based on all outcomes in the set, and adds that outcome if it is not weakly dominated by another already calculated outcome. This has to be done for each possible set of players. This is shown in Algorithm 5.

Since the utility any player can have in an outcome is bounded by the minimum utility a player can have and the maximum utility a player can have, the size of the set is bounded by W^n, where W is the difference between the minimum utility a player can gain and the maximum utility a player can gain.

Given that $\mathcal{P}(N)$ has size $O(2^n)$, T has size $O(t)$, S and U have size $O(W^n)$, C has size $O(n)$, adding an element to a list takes $O(1)$, and removing an element from a linked list takes $O(1)$ while looping through that list, the time complexity of this algorithm is $O(2^n \cdot (t \cdot (W^n \cdot n^2 + W^{n^2} \cdot n) + W^n)) = O(2^n \cdot t \cdot W^{2 \cdot n} \cdot n)$. That means that for a given n, this algorithm is pseudo-polynomial, which makes the problem weakly $NP - hard$.

Note that S and U are also bounded by $O((n+1)^t)$, as the amount of outcomes is bounded by the amount of allocations. This bounds the running time by $O(2^n \cdot t \cdot (n+1)^t \cdot n)$ as well. This is exponential in the amount of tasks, but constant in W. This bound will be relevant if W is large relative to t.

Approximation algorithm

If the maximum and minimum utilities are very high, it could be infeasible to find an allocation in the Core with the pseudo-polynomial algorithm. In that case, we might like to approximate the exact solution with an allocation in the ζ-Core. To guarantee a solution in the ζ-Core, we can replace dominates with ζ/t-dominates. This limits the size of S and U to $O((W \cdot t/\zeta)^n)$. This makes the time complexity of the algorithm $O(2^n \cdot t \cdot (W \cdot t/\zeta)^{2 \cdot n} \cdot n)$.

Algorithm 5 Pseudo-polynomial algorithm to find relevant outcomes

```
ParetoFront ← ∅
for all C ∈ 𝒫(N) do
    S ← {(0, ..., 0)}
    for all j ∈ T do
        U ← ∅
        for all s ∈ S do
            for all i ∈ C do
                o ← s
                for all k ∈ C do
                    o_k ← o_k + g_{k,j}
                end for
                o_i ← o_i − c_{i,j}
                Add o to U
            end for
        end for
        for all u ∈ U do
            for all s ∈ S do
                if s weakly dominates u then
                    Break
                end if
                if u weakly dominates s then
                    Remove s from S
                end if
                Add u to S
            end for
        end for
    end for
    Remove all s ∈ S where a player gains negative utility
    for all s ∈ S do
        Add s to ParetoFront
    end for
end for
return ParetoFront
```

Integer linear program

When (the approximation of) a Pareto Front is found, we can use the outcomes in them as constraints for the integer linear program. This is shown in Algorithm 6.

Algorithm 6 Find Allocation In The Core

Maximize $\sum_{i \in N} \left(\sum_{k \in N} \sum_{j \in A_k} g_{ij} - \sum_{j \in A_i} c_{ij} \right)$ where:
for all $p \in ParetoFront$ **do**
  for all $i \in C$ **do**
    $u_i(A) \geq p_i \cdot x_i$
  end for
  $\sum_{i \in C} x_i \geq 1$
end for
return A

Since the size of the Pareto Front is $O(W^n)$ for the pseudo-polynomial algorithm and $O((W/\zeta)^n)$ for the approximation algorithm, the running time is not dependent on the amount of tasks. Therefore, the k-player problem is polynomial.

Approximation guarantee

If the approximation algorithm gives an allocation A, we know for all $p \in ParetoFront$ that $A_i \geq p_i$ for some $i \in N$. Also, there does not exist an $A^* \notin ParetoFront$, where $A^*_i - \zeta > p_i$ for all $i \in N$ and all $p \in ParetoFront$, otherwise A^* would have been added to $ParetoFront$. Therefore, we know there does not exist an A^* that ζ-dominates A, so A is in the ζ-Core.

We also know that if an outcome is in the $ParetoFront$ of the approximation algorithm, it will either also be in the $ParetoFront$ of the pseudo-polynomial algorithm, or it will be dominated by an outcome in the $ParetoFront$ of the pseudo-polynomial algorithm. Therefore, any allocation that satisfies the constraints of the pseudo-polynomial algorithm also satisfies the constraints of the approximation algorithm, so the approximation algorithm always finds a maximum Utilitarian Welfare which is at least the maximum Utilitarian Welfare of the exact solution.

5.3 Conclusion

To conclude, we have proposed a pseudo-polynomial algorithm that finds the allocation in the Core with the maximum Utilitarian Welfare, and an approximation algorithm that finds an allocation in the ζ-Core with a Utilitarian Welfare of at least the maximum Utilitarian Welfare in the Core. Chapter 6 discusses the experimental results of these algorithms.

Chapter 6

Experiments

In Chapter 5, we discussed two algorithms to solve the problem. In this chapter, we report on the experiments with these algorithms. We aim to find out the feasibility of the algorithms. To do this, we discuss the running times and approximation ratios of the algorithms.

6.1 Experimental setup

To experimentally test the running time of the proposed algorithms, the algorithms were implemented and run with randomly generated instances. The algorithms were implemented in Java. Instances were generated with n players and t tasks where each value of G and C has a random value from a uniform distribution between (and including) 0 and an input variable W. The value of n, t and W was varied to check the feasibility for different applications of the problem. The tests were done on a Windows 10 (64 bit) machine with an Intel Core i7-8750H of 2.20 GHz with 16.0 GB RAM.

For the algorithms that use Linear Programming, the Gurobi solver was used.

6.2 The pseudo-polynomial algorithm

To value the running time of the pseudo-polynomial Linear Programming algorithm that uses the Pareto Front, it was compared to the running time of the Brute Force algorithm, and the Linear Programming algorithm that uses all allocations. Each of the algorithms was tested on the same 50 instances. The plot shows the mean running time of these runs and the 80% confidence intervals. In each graph, the line shows the mean running time of the runs on the 50 instances, and the area around the line contains the running time of all instances except the 5 with the lowest and the 5 with the highest running time. For each algorithm, when the running time becomes larger than 30 000 milliseconds, the algorithm was tested with only 1 instance. This is when the plot only

shows a line without the area around it. After the running time becomes larger than 100 000 milliseconds, experiments were stopped. Experiments were also stopped after a set value.

6.2.1 Varied amount of tasks

We would like to know how feasible the pseudo-polynomial algorithm is for different amount of tasks. Since the running time is bounded both by $O(2^n \cdot t \cdot W^{2 \cdot n} \cdot n)$ and by $O(2^n \cdot t \cdot (n+1)^t \cdot n)$, we expect that the running time will increase exponentially with a low t, and linearly when t becomes high enough such that the first bound becomes relevant. To test this, we evaluate the running time on 50 random instances for different values of t while keeping n and W constant. Figure 6.1 shows the results of these experiments.

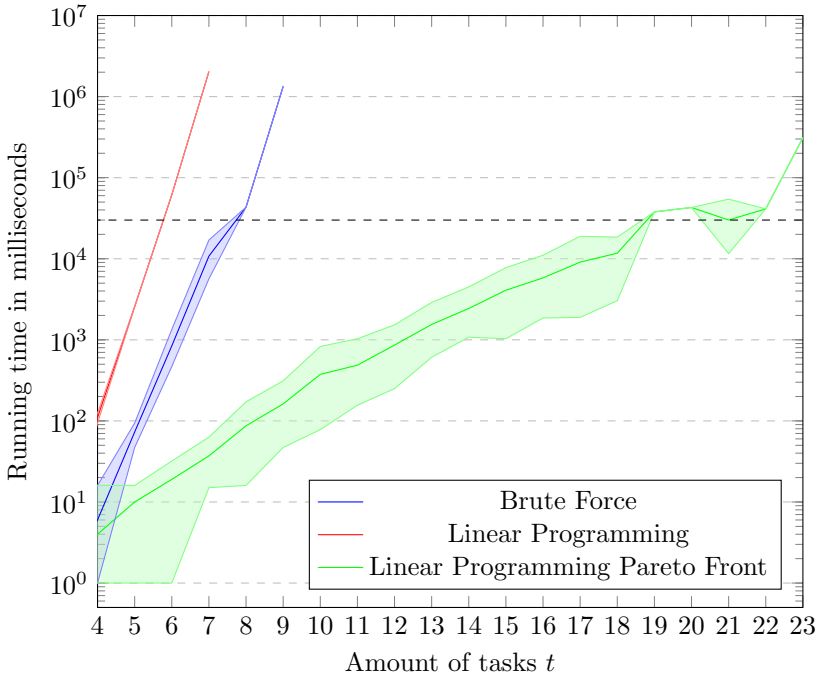

Figure 6.1: Running time of different algorithms for $n = 3$ and $W = 10$ (mean of 50 instances below dashed line, 1 instance above it)

As can be seen, the Brute Force algorithm and the Linear Programming algorithm become infeasible after a very limited amount of tasks. The Linear Programming algorithm using the Pareto Front does significantly better, but the growth still looks exponential. Possibly, even the highest tested values of t are not high enough that the linear bound becomes relevant. It could also be that it is hard to see with the noise in the data.

To be able to run with higher amounts of tasks, we make W dependent on t. We let W be $W = 100/t$. This has the additional advantage that we now expect a much slower growth, which is easier to see in a noisy plot. We use the same setup as before, except that for each value of W, we run the algorithm for a maximum of one value of t. Figure 6.2 shows the results of these experiments.

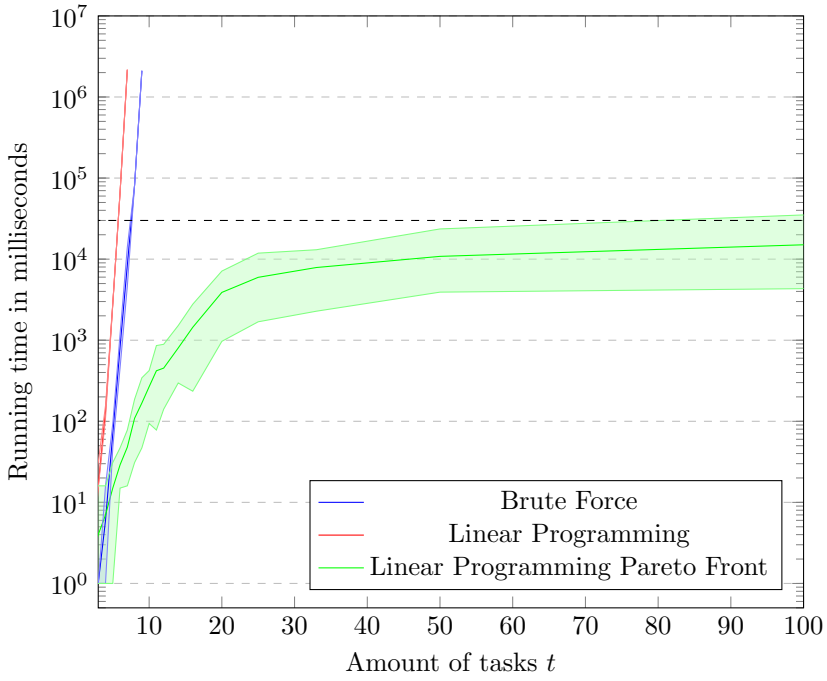

Figure 6.2: Running time of different algorithms for $n = 3$ and $W = 100/t$ (mean of 50 instances below dashed line, 1 instance above it)

As can be seen, the Brute Force algorithm and the Linear Programming algorithm have a very similar running time with a different value for W. The Brute Force algorithm runs in around 2 000 000 milliseconds with 9 tasks, while the Linear Programming algorithm runs in around 2 000 000 milliseconds with 7 tasks. For both algorithms, the value of W has little influence on the running time. The Linear Programming algorithm using the Pareto Front grows quickly with a low amount of tasks, and much slower after around 20 tasks. This can be explained by the the fact that the bound $O(2^n \cdot t \cdot W^{2 \cdot n} \cdot n)$ becomes more relevant and is less dependent on t. Clearly, W has a significant influence on the running time of this algorithm.

6.2.2 Varied value of W

We would also like to know how feasible the pseudo-polynomial algorithm is for different values of W. Since the running time is bounded both by $O(2^n \cdot t \cdot W^{2 \cdot n} \cdot n)$

and by $O(2^n \cdot t \cdot (n+1)^t \cdot n)$, we expect that the running time will increase linearly with a low W, and stay constant when W becomes high enough such that the second bound becomes relevant. To test this, we evaluate the running time on 50 random instances for different values of W while keeping n and t constant. Figure 6.3 shows the results of these experiments.

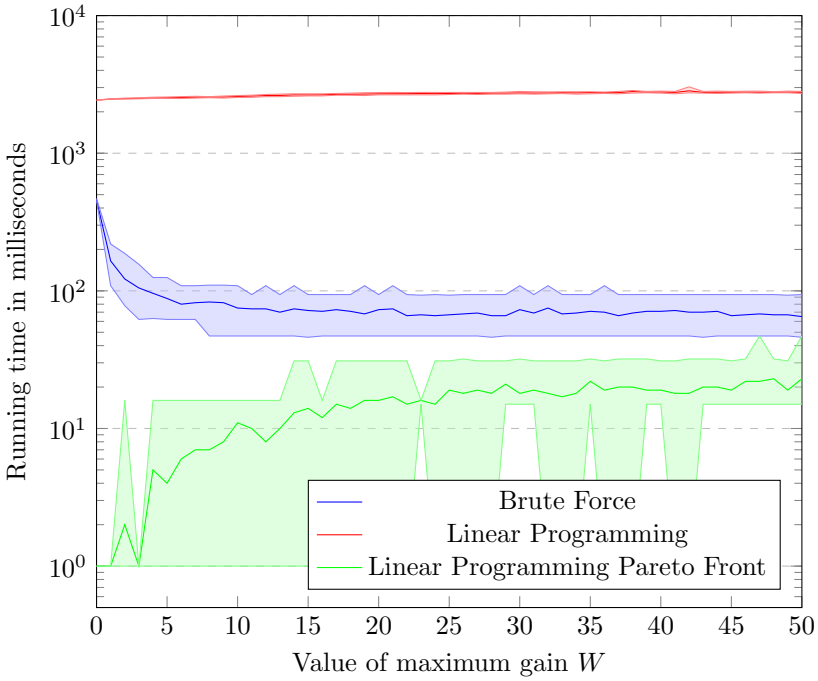

Figure 6.3: Running time of different algorithms for $n = 3$ and $t = 5$ (mean of 50 instances)

As can be seen, the Brute Force algorithm performed worse with a small W. This can be explained by the fact that with a small W, there is a very limited amount of variation in the value of the utility of the players. Because of this, more allocations have the same value. This might result in allocations being less likely to dominate other allocations. Since the Brute Force algorithm searches a dominating allocation for each possible allocation, it is slower in finding these for a low W, resulting in a higher running time.

The value of W has very little influence on the running time of the Linear Programming algorithm. Since the amount of allocations is only dependent on n and t, this is expected.

The running time of the Linear Programming algorithm that uses the Pareto Front increases with a higher value of W, but only until some value of W, after which it is constant. This is as expected.

6.2.3 Varied amount of players

Finally, we would like to know how feasible the pseudo-polynomial algorithm is for different amounts of players. Since the running time is bounded by n exponentially, we expect that the running time will increase exponentially. To test this, we evaluate the running time on 50 random instances for different values of n while keeping W and t constant. Figure 6.4 shows the results of these experiments.

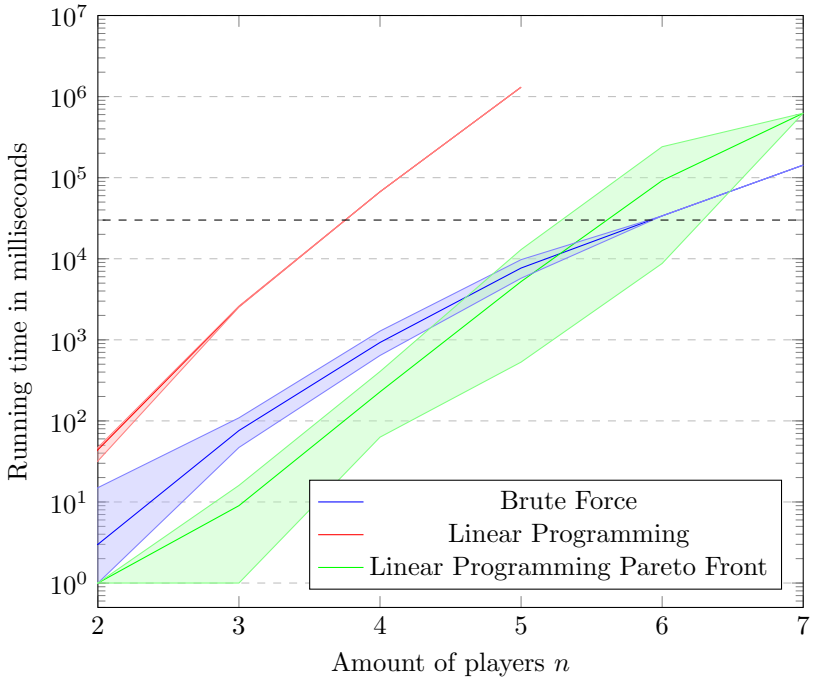

Figure 6.4: Running time of different algorithms for $t = 5$ and $W = 10$ (mean of 50 instances below dashed line, 1 instance above it)

As can be seen, all algorithms grow exponentially, which makes the amount of players for which the algorithms can feasibly compute the solution very limited. The Brute Force algorithm and the Linear Programming algorithm that uses the Pareto Front are significantly better than the Linear Programming algorithm. However, because of the high spread in the Linear Programming algorithm that uses the Pareto Front, it is unclear from these experiments which of those two works better.

6.3 The approximation algorithm

The approximation algorithm can be run with different values of the input parameter ζ. Recall that ζ/t was used to limit the size of the approximated Pareto Front. If an allocation was ζ/t-dominated by an allocation in the Pareto Front, it was removed from the Pareto Front. For a larger ζ, the running time becomes lower, but the guarantee we can give about the approximation becomes worse. Since we need to make a trade-off between running time and approximation guarantee, we would like to know the running time and approximation for different values of ζ. Since we are more interested in allocations that are almost in the Core, we vary ζ in a logarithmic scale. ζ/t is varied between 1 and 64. Note that therefore, the tested approximation guarantees ζ are dependent on t.

6.3.1 Running time

First, we would like to know how feasible the approximation algorithm is for different values of ζ. Since the running time is bounded by $O(2^n \cdot t \cdot (W \cdot t/\zeta)^{2 \cdot n} \cdot n)$, we expect the running time to scale inversely proportional to ζ. To test this, we evaluate the running time on 50 random instances for each value of ζ. Figure 6.5 shows the results of these experiments. Again, the line shows the mean running

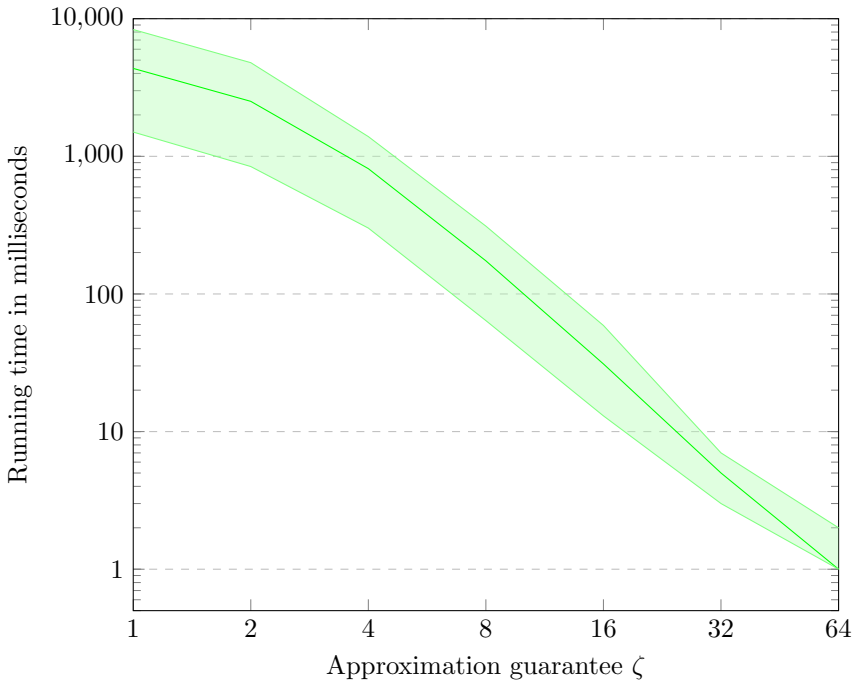

Figure 6.5: Running time of the approximation algorithm for $n = 3$, $t = 12$ and $W = 64$

time of the runs on the 50 instances, and the area around the line contains the running time of all instances except the 5 with the lowest and the 5 with the highest running time.

As can be seen, the running time decreases significantly with a higher ζ. This is as expected.

6.3.2 Approximation

Of course, we would also like to know how well the approximation algorithm approximates a strategyproof allocation. To test this, we run the algorithm on 1000 random instances for each value of ζ. We then calculate the minimum ϵ for which the found allocation is in the ϵ-Core. Figure 6.6 shows the results of these experiments. The bar plot shows for each value of ζ how many experiments found an allocation with a minimum ϵ in different ranges.

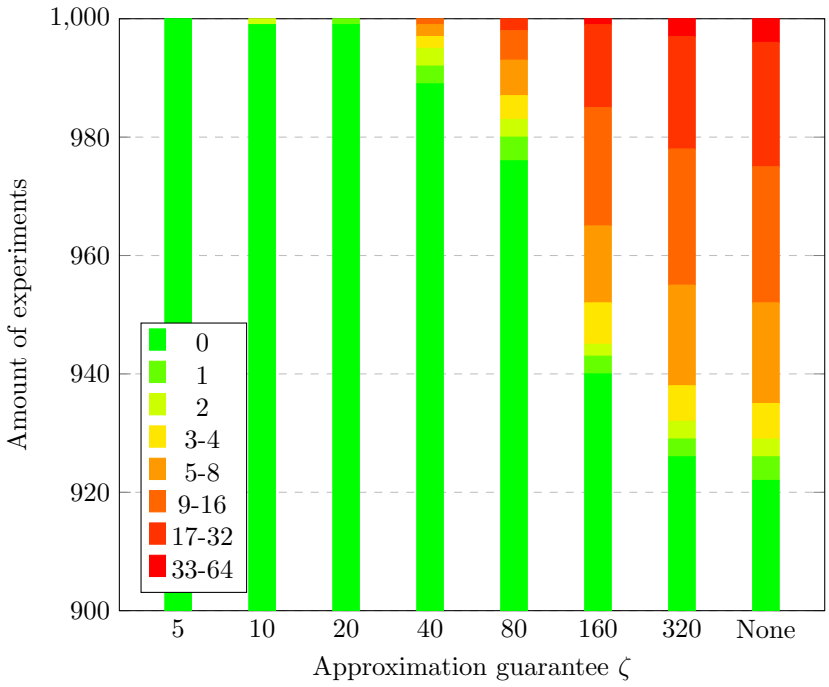

Figure 6.6: Approximation of the approximation algorithm for $n = 3$, $t = 5$ and $W = 64$

As can be seen, even though the algorithm does not guarantee a solution in the Core, it often finds an allocation in the Core anyway. In 92.2% of the instances, an allocation with the maximum Utilitarian Welfare is in the Core, so the algorithm will almost always find such an allocation irrespective of ζ. If the maximum Utilitarian Welfare allocation is not in the Core, the larger ζ, the

less often it will find an allocation in the Core. This is as expected. Of these 1000 experiments, the algorithm with $\zeta = 1$ found an allocation in the Core in all but one case, the algorithm with $\zeta = 64$ found an allocation in the Core in 926 cases. Also, if ζ is larger, the minimum ϵ for which the found allocation is in the ϵ-Core is clearly higher.

6.4 Conclusion

The experiments show that the Linear Programming algorithm using the Pareto Front has clearly lower running times than the Brute Force algorithm or the Linear Programming algorithm. They show that the running time does not scale well with the amount of players n, but that the algorithm is feasible if either the amount of tasks t or the value of W is limited.

The experiments also show that the running time can be decreased significantly by setting a higher approximation guarantee ζ. Even though the algorithm does not guarantee an allocation in the Core, it still finds an allocation in the Core in a lot of cases.

Chapter 7

Conclusion

In the previous chapters, we gave proofs and did experiments to show the limitations and possibilities for algorithms that solve the task distribution problem of this research. In this chapter, we use these proofs and experiments to argue when it is feasible to use the proposed algorithms for this problem. Additionally, we discuss what knowledge this research contributed, and how future work can use this research to gain more knowledge about this task distribution problem.

7.1 Conclusion

In this research, we studied a task distribution problem where multiple players have to distribute tasks among them. Each player values tasks being done, but has a cost to do a task himself. Since the players have different preferences, we would like to find an algorithm with an optimal distribution.

Ideally, this is an efficient algorithm that finds an allocation in the Core with a high Utilitarian Welfare. From this study, we can conclude that there exists an algorithm that finds the maximum Utilitarian Welfare (Theorem 1). Finding an allocation in the Core is the main problem.

First of all, we know that it is impossible for an algorithm to always find an allocation in the Core (Theorem 6). Additionally, even if the instance has a non-empty Core, requiring an allocation in the Core can give a Utilitarian Welfare that is infinitely worse than the allocation with the maximum Utilitarian Welfare (Theorem 12). Finally, it is an $NP - hard$ problem to find an allocation in the Core (Corollary 8.1).

However, we know there exists a pseudo-polynomial algorithm (Algorithm 5) which uses a Linear Program (Algorithm 6) to find an allocation in the Core if it exists. From the experiments, we can conclude that if the amount of players is low, and either the amount of tasks or the maximum utility a player can achieve is limited, this algorithm runs in a feasible amount of time. Also, there exists an approximation algorithm that finds such an allocation in a feasible amount of time if we scale the approximation guarantee with the maximum amount of

utility a player can achieve. For applications where these limitations are not a problem for the requirements, the proposed algorithms can be used to distribute a set of tasks in an (almost) strategyproof way.

7.2 Future work

In this research, we studied a problem in the context of fair division problems. To the best of our knowledge, this specific problem was not studied before. We proposed a model for this problem, studied its theoretical properties, proposed two algorithms to solve it, and did experiments to verify the performance of these algorithms.

As this is the first research for this specific problem, there is a lot of further research that can be done.

7.2.1 Properties of the problem

In this research, we proved that the Find-Core problem is $NP - hard$. With that knowledge, we know finding a polynomial algorithm to find an allocation in the Core is impossible. Future work could find the exact complexity class of the problem, possibly using similar reductions to the reductions used in this research. This would give us more knowledge about how feasible an algorithm that finds the Core can be.

In this research we proved that the minimum ϵ for which an ϵ-Core always exists lies between 1/15 and 1/2 of the maximum utility a player can achieve. With that knowledge, we know we can guarantee that there is an allocation in the $MaxUtility/2$-Core. Future work could find a tighter lower or upper bound for this minimum ϵ. A tighter upper bound would show that we can give a better guarantee, and a tighter lower bound would show that it is impossible to give a better guarantee than the value of that bound.

In this problem, we want to find an allocation that has a Utilitarian Welfare as high as possible, and is in the ϵ-Core for an ϵ as low as possible. These two optimization measures make this a multi-objective problem. In this research, we showed that maximizing one of the optimization measures cannot give any nontrivial guarantee for the other. With that knowledge, we know it is impossible to find an algorithm that maximizes one of the optimization measures, and gives some guarantee for the other. Future work could study whether it is possible to find an allocation which has some nontrivial suboptimal guarantee for both optimization measures. This might be more desirable in some applications than just optimizing one of the measures.

7.2.2 Algorithms

In this research, we studied linear programming as a solution technique. The proposed algorithms had a decent theoretical bound on the running time. Future

work could investigate the performance of other algorithms, because they might work better in practice. This includes heuristic approaches.

Specifically, we think evolutionary algorithms could work well for this problem. Since the valuation functions are additive, the tasks are independent of each other. Also, assigning a task from one player to another only changes the utilities of those players. This gives the problem a very structured relation between the allocations and the outcomes. Evolutionary algorithms typically work well for such structured problems.

7.2.3 Experiments

In this research, we did experiments with randomly generated data. These experiments give a good picture of how the running time is dependent on the input variables. They also show that the approximation algorithm finds a solution in the Core very often for applications where the gains and costs look random. Future work could additionally study how well the algorithm performs with other data.

Specifically, it could be interesting to look at data where the Core is empty, or consists of a very low amount of allocations. If the Core is small, that could result in the algorithm finding an allocation in the Core less often. These results could give a more complete picture of the performance of the algorithm.

Also, it could be interesting to look at data from practical applications. Experiments with real data might be able to predict the results better for those specific applications.

7.2.4 Optimization measures

In this research, we considered two optimization measures, the Utilitarian Welfare and the Core. We found possibilities and limitations for algorithms that maximize Utilitarian Welfare and find allocations in the Core. Future work could also study properties of this problem for other optimization measures. Chapter 2 already discusses the alternatives to Utilitarian Welfare, namely Egalitarian Welfare (Grant et al., 2010; Harsanyi, 1975; Myerson, 1981) and Nash Welfare (Kaneko & Nakamura, 1979). But more qualitative measures are possible too. Envy-freeness is a measure which is often used when talking about fairness. It means that each player likes their own allocated tasks better than the allocated tasks of any of the other players. The question is how that would exactly translate to this problem. For applications where those optimization measures are relevant, it could be interesting to see what limitations exist and what guarantees we could give about these individual measures, or a combination of them.

7.2.5 Variations in the model

Finally, there could be a variation on the used model. Additional assumptions might make the problem easier. It is possible that algorithms could use those

assumptions to find a solution faster. An example of such a variation is the ability for players to perform a task together, sharing the cost. This allows for an infinite amount of possible allocations, and the possible values the utility of a player can be becoming continuous rather than discrete. With this assumption, the proofs in this research do not hold, which means it is possible that there will always exist a Core, for example.

The model could also be more general. In this research, the valuation functions were assumed to be additive. Of course, it is possible that in an application, some player only values a certain task if another task is done too. In this research, such applications cannot be modeled. It would be interesting to see which results still hold under a more general model.

Of course, a lot of other variations of the model are possible too.

References

Airiau, S., & Endriss, U. (2014). Multiagent resource allocation with sharable items [Journal Article]. *Autonomous agents and multi-agent systems*, *28*(6), 956-985.

Aumann, R. J. (1961). The core of a cooperative game without side payments [Journal Article]. *Transactions of the American Mathematical Society*, *98*(3), 539-552.

Caragiannis, I., Kurokawa, D., Moulin, H., Procaccia, A. D., Shah, N., & Wang, J. (2019). The unreasonable fairness of maximum nash welfare [Journal Article]. *ACM Transactions on Economics and Computation (TEAC)*, *7*(3), 1-32.

Chalkiadakis, G., Elkind, E., & Wooldridge, M. (2011). Computational aspects of cooperative game theory [Journal Article]. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *5*(6), 1-168.

Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaitre, M., Maudet, N., ... Sousa, P. (2006). Issues in multiagent resource allocation [Journal Article].

Gillies, D. B. (1959). Solutions to general non-zero-sum games [Journal Article]. *Contributions to the Theory of Games*, *4*, 47-85.

Grant, S., Kajii, A., Polak, B., & Safra, Z. (2010). Generalized utilitarianism and harsanyi's impartial observer theorem [Journal Article]. *Econometrica*, *78*(6), 1939-1971.

Harsanyi, J. C. (1975). Nonlinear social welfare functions [Journal Article]. *Theory and Decision*, *6*(3), 311-332.

Kaneko, M., & Nakamura, K. (1979). The nash social welfare function [Journal Article]. *Econometrica: Journal of the Econometric Society*, 423-435.

Ma, J. (1994). Strategy-proofness and the strict core in a market with indivisibilities [Journal Article]. *International Journal of Game Theory*, *23*(1), 75-83.

Myerson, R. B. (1981). Utilitarianism, egalitarianism, and the timing effect in social choice problems [Journal Article]. *Econometrica: Journal of the Econometric Society*, 883-897.

Procaccia, A., Goldman, J., Shah, N., & Kurokawa, D. (2020). *Spliddit* [Web Page]. Retrieved from `spliddit.org`

Procaccia, A. D., & Wang, J. (2014). Fair enough: Guaranteeing approximate maximin shares [Conference Proceedings]. In *Proceedings of the fifteenth*

acm conference on economics and computation (p. 675-692).
Ramezani, S., & Endriss, U. (2009). Nash social welfare in multiagent resource allocation [Book Section]. In *Agent-mediated electronic commerce. designing trading strategies and mechanisms for electronic markets* (p. 117-131). Springer.
Shapley, L., & Scarf, H. (1974). On cores and indivisibility [Journal Article]. *Journal of mathematical economics*, *1*(1), 23-37.
Von Neumann, J., Morgenstern, O., & Kuhn, H. W. (1953). *Theory of games and economic behavior* [Book]. Princeton university press.
Weber, S., & Wiesmeth, H. (1991). Economic models of nato [Journal Article]. *Journal of Public Economics*, *46*(2), 181-197.