University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Piotr Skowron

Resource Allocation in Selfish and Cooperative Distributed Systems

PhD dissertation

Supervisors

dr hab. inż. Piotr Faliszewski
dr inż. Krzysztof Rządca

Institute of Informatics
University of Warsaw

September 2014

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

September 15, 2014 . . . .

date *Piotr Skowron*

Supervisor's declaration:
the dissertation is ready to be reviewed

September 15, 2014 . . . .

date *dr hab. inż. Piotr Faliszewski*

September 15, 2014 . . . .

date *dr inż. Krzysztof Rządca*

Abstract

In this dissertation we take an algorithmic view on resource allocation problems in distributed systems. We present a comprehensive perspective by studying a variety of distributed systems—from abstract models of generic distributed systems, through more specific and detailed models, to real distributed computer systems. These systems differ with respect to the nature of the resource allocation problems and with respect to the methodologies required to effectively solve them.

Effective resource allocation in distributed systems is a fundamental problem. Computer systems require good resource management mechanism to ensure expected functionality and expected quality of service. Even in our everyday life we often participate in resource allocation mechanisms. The examples range from cutting a cake at the birthday party to parliamentary elections and referendums (where the participating candidates can be viewed as resources).

We start our discussion from considering a general computational model that describes the problem of selecting a collective set of items for the shared use by a group of agents. This model is very general; it does not specify what is an agent and what is an item, and, thus, can be applied to many different scenarios. Indeed, we show that our model captures many real-life resource allocation problems. For instance, our algorithms for this general model can be applied to recommendation systems, e.g., to select a collection of movies for a plane, or to allocate students to sport classes based on their preferences. Our algorithms are also applicable to the problem of finding a proportional representation of a society in some collective decision-making body, i.e., to find winners in some modern parliamentary election systems. We analyze multiple variants of our general problem of selecting a collective set of items.

Next, we move to more specific models of computer systems. In these models we introduce several new elements such as jobs, processors, and the network. Each of these elements can be further described by a set of parameters. For instance, jobs have their release times, resource requirements, and durations; further, their durations might depend on the processors on which they are run; the processors might be identical or heterogeneous. The network connections can be described, e.g., by bandwidth and communication latencies. Consequently, in these models we focus on several variants of a more general problem. We ask how to schedule jobs to minimize their aggregated completion time, with specific variants of this question depending on the aggregation method and on the characteristics of the elements of the model. We establish computational complexity of variants of this scheduling problem and, in particular, we show effective algorithms optimizing jobs' schedules. We also provide analysis of other properties of our algorithms, such as their fault-tolerance and their game-theoretic stability.

In the last part of this dissertation we consider resource allocation problems in real implementations of complex distributed systems. We consider two storage-based systems: HYDRAstor, which is a commercial, distributed, scalable, high-performance, secondary storage system targeted for the enterprise market, and our prototype implementation of a P2P backup system. We explain how the design of resource allocation mechanisms for such complex systems is different from our previous approaches. In this part we present and discuss relatively more complex resource allocation mechanisms; these mechanisms consist of multiple elements and even of whole resource allocation subsystems. Further, they aim at achieving multiple (sometimes contradicting) goals.

Keywords: distributed systems, multi-agent systems, algorithms, complexity, approximation, game theory, social choice, cooperation, competition, proportional representation.

ACM clasification: CCS → Theory of computation → Design and analysis of algorithms
CCS → Computing methodologies → Artificial intelligence → Distributed artificial intelligence → Multi-agent systems
CCS → Computer systems organization → Architectures → Distributed architectures

Streszczenie

W poniższej rozprawie badamy algorytmy zarządzania zasobami w systemach rozproszonych. Przedstawiamy kompleksowe spojrzenie na tę tematykę: rozważamy różne systemy—od ogólnych, abstrakcyjnych modeli, przez bardziej konkretne, dedykowane modele, po rzeczywiste systemy rozproszone. Rozważane systemy różnią się specyfiką problemów zarządzania zasobami oraz metodologią, którą jest dla tych problemów najbardziej adekwatna.

Efektywne zarządzanie zasobami w systemach rozproszonych jest problemem o fundamentalnym znaczeniu. Systemy komputerowe wymagają dobrych mechanizmów zarządzania zasobami aby zapewnić odpowiednią jakość usług dla użytkowników. Również w naszym codziennym życiu często uczestniczymy w mechanizmach zarządzania zasobami. Przykłady takich mechanizmów to między innymi podział tortu na przyjęciu urodzinowym, czy referenda, a nawet wybory parlamentarne (w tym przypadku możemy utożsamiać kandydatów startujących w wyborach z zasobami).

W pierwszej części rozprawy rozważamy ogólny, abstrakcyjny model który opisuje problem wyboru podzbioru pewnych obiektów, które następnie będą współdzielone przez grupę użytkowników. Ten model jest bardzo ogólny ponieważ nie specyfikujemy kim (lub czym) dokładnie jest użytkownik i czym dokładnie są owe obiekty. W rezultacie, rozwiązania oparte o ten model możemy zaaplikować do wielu rzeczywistych problemów, takich jak przydział studentów, w oparciu o ich preferencje, do uniwersyteckich kursów, czy znajdowanie właściwych rekomendacji. Takie rekomendacje mogą dotyczyć, na przykład, wyboru zbioru filmów dostępnych na pokładzie samolotu. Nasze algorytmy znajdują takżę zastosowanie w znajdowaniu proporcjonalnej reprezentacji dla grupy ludzi, czyli np. aby znaleźć zwycięzców w niektórych nowoczesnych systemach wyborów parlamentarnych. W niniejszej rozprawie analizujemy wiele specyficznych wariantów tego ogólnego zagadnienia.

W drugiej częsci rozprawy rozważamy bardziej specyficzne modele opisujące systemy komputerowe. W tych modelach pojawiają się nowe elementy, takie jak zadania, procesory, czy sieć komputerowa. Każdy z tych elementów może być opisany przez zbiór parametrów: zadania mają swoje czasy powstania, wymagania zasobów, czy czasy wykonania. Długość trwania zadania może ponadto zależeć od rodzaju procesora na którym zadanie zostało uruchomione: procesory mogą być identyczne lub heterogeniczne. Połączenia sieciowe mogą być opisane przez przepustowość lub/i latencję komunikacji. Naturalnie w tych modelach zadajemy również bardziej specyficzne pytania. Pytamy jak uszeregować zadania, aby zminimalizować ich zagregowany czas zakonczenia. Specyficzne warianty tego pytania róznią się w zależności od metody agregacji oraz w zależności od cech charakterystycznych wybranych elementów modelu. W rozważanych modelach badamy złożoność obliczeniową różnych wariantów problemu szeregowania, w szczególności pokazując efektywne algorytmy do optymalizacji uszeregowania zadań. W tej części rozprawy analizujemy również inne cechy naszych algorytmów, takie jak odporność na błędy czy (teorio-growa) stabilność.

W ostatniej części rozprawy rozważamy problemy zarządzania zasobami w rzeczywistych, złożonych, komputerowych systemach rozproszonych. Rozważamy dwa rzeczywiste systemy przechowywania danych: HYDRAstor, który jest komercyjnym, rozproszonym, skalowalnym systemem przechowywania danych, oraz naszą prototypową implemenację systemu do tworzenia kopii zapasowych danych, opartego o architekturę P2P. Wyjaśniamy czym różni się projektowanie mechanizmów zarządzania zasobami dla takich systemów od poprzednio rozważanych przypadków. W tej części prezentujemy relatywnie bardziej złożone mechanizmy zarządzania zasobami: mechanizmy te składają się z wielu elementów, a nawet z wielu podsystemów zarządzania zasobami. Co więcej takie podsystemy mogą mieć czasami sprzeczne ze sobą cele.

Słowa kluczowe: systemy rozproszone, systemy wieloagentowe, algorytmy, złożoność, aproksymacja, teoria gier, wybór społeczny, kooperacja, konkurencja, proporcjonalna reprezentacja.

Klasyfikacja ACM: CCS → Teoria obliczeń → Projektowanie i analiza algorytmów
CCS → Metodologie obliczeniowe → Sztuczna inteligencja → Rozproszona sztuczna inteligencja → Systemy wieloagentowe
CCS → Zorganizowane systemy komputerowe → Architektury → Rozproszone architektury

Contents

Chapter 1

Introduction

In this dissertation we take an algorithmic view on resource allocation problems in distributed systems. We present a comprehensive perspective by studying a variety of distributed systems—from abstract models of generic distributed systems, through more specific and detailed models, to real distributed computer systems. These systems differ with respect to the nature of the resource allocation problems and with respect to the methodologies required to effectively solve them.

Effective resource management is a challenge of fundamental significance that we face in our everyday life. On one hand, we all need resources. We, as human beings, need biological resources for a healthy living; we, as the society, need natural and economic resources for a sustainable development; we, as organizations of various kinds, need human resources to reach our goals. On the other hand, resources are often limited and hard to access. For instance, some are accessible only by coordinated cooperative groups of people equipped with expensive hardware (the examples from our everyday life include mining or obtaining semi-manufactured resources as products of complex industrial processes). Consequently, people make continuous efforts to acquire and to use resources effectively.

This challenge of effective resource usage and management is also extremely pressing in the context of computer systems. Computer systems need resources to offer expected functionality, to offer quality of service, and to serve large numbers of users. The resources are, however, expensive. Some are accessible only to large cooperative coalitions of smaller systems. Such unique resources include, for example, (i) high computational power, (ii) geographically distributed servers, and (iii) large virtual disk space. Indeed, we already witness the phenomenon of cooperation in computer systems provoked by the desire to obtain these unique resources. There are organizations that consolidate their supercomputers into clouds to obtain greater computational power (e.g., PL-Grid [242], EGEE [93], DAS [76], GRID5000 [40]). There are vast content delivery networks designed to efficiently distribute web content around the world (e.g., Akamai [194,226]). Further, there are common people joining P2P storage systems to get access to geographically distributed disk space (e.g., Freenet [62], or previously Wuala [193]). Cooperation is also natural in the scientific community. For example, consider PlanetLab [58]—a system in which participating scientists from all over the world give access to their servers to the central administration unit. After providing their servers to the system, the scientists are allowed to run large-scale experiments using large, geographically disperse, infrastructure. Another example is BOINC [11]—a system that lets common people contribute the computational power of their desktop machines to research projects from many scientific areas. BOINC is an example showing how cooperation of common people allowed to create a distributed system with high computational power.

Above, we argued the importance and universality of resource management problems in general. We think, however, that effective resource management is especially imperative in *distributed systems*. Indeed, *distributed systems are ubiquitous*. For instance, all the examples given above concern resource management in distributed systems. The human body is a system consisting of multiple organs, or even billions of cells. The society is a distributed system consisting of the citizens. The universities, or research centers, are distributed systems gathering scientists. In fact, we often participate in resource management mechanisms in our everyday life, even though we do not often realize we do so. The examples range from cutting a cake at a birthday party (see [246] for a survey on the cake cutting problem), where a cake is a resource to be shared within a system of children playing at a party, through waiting in a queue to a post office [221], where a post office is a resource accessed by a system of customers, to parliamentary elections and referendums, where the participating candidates are resources to be selected for the system of citizens. Similarly, resource management is crucial for *distributed computer systems*, such as computational grids, content delivery networks, and P2P storage systems, composed of large numbers of computational units governed by multiple (virtual) organizations.

Both in our everyday life and in computer systems, we can observe two concepts aimed at making resource management more effective. First, there is a strong tendency to use resources wisely, i.e., not to waste resources. Second, we observe coordination and participation in joint efforts to acquire, maintain and use resources. Both concepts, optimization and cooperation, may require sophisticated resource management mechanisms.

The challenges in designing effective resource allocation mechanisms include, on one hand, designing accurate yet simple models of appropriate business processes and, on the other hand, designing algorithms that solve optimization tasks regarding these models. Taking into account the fact that the participating entities (people, organizations) may have contradicting goals, the challenge also includes designing incentive-compatible and fair mechanisms. The stability of organizationally distributed systems requires ensuring that each organization has an incentive to participate in the system. In particular, we would like the profits obtained by the system to be shared between the participating organizations in a fair and effective manner. Any system that fails to provide fair and well-optimized resource management will collapse either because it will not be competitive or because the organizations will simply have no desire to join it.

Addressing the aforementioned challenges is often hard. This is confirmed by theoretical studies that bring many evidences for the difficulty of resource allocation problems. For example, effective resource management may require solving computationally difficult, NP-hard, problems. This NP-hardness might seem as an invincible barrier, especially because distributed systems require mechanisms that scale well and that are able to handle large input data. Worse yet, NP-hardness of various resource allocation problems is not the only obstacle. For example, in the multi-agent systems, where the process of making decisions is distributed, the strategic behavior of participating agents may lead to suboptimal and socially undesired outcomes (for example, as in the prisoner's dilemma [254], the famous paradox). Furthermore, stable situations in which agents have no incentive to act to change the current state may not exist. Finally, real complex computer systems may require whole interacting subsystems of resource management mechanisms.

In this dissertation we take up the gauntlet and argue that in practice we can often solve resource allocation problems effectively and efficiently. On one hand, we provide proofs of theoretical difficulty of various resource allocation problems in distributed systems. On the other, we show effective, if not ideal, solutions that can be used in practice. In particular, we present the following techniques:

1. We show efficient algorithms giving very good approximation guarantees for the underlying computational problems.

2. We experimentally confirm that our algorithms very well approximate the optimal solutions.

3. We show how to apply game-theoretic solution concepts for resource management in various distributed systems.

4. We design new effective resource management mechanisms for some example real-life distributed systems (these mechanism include systems' architectures, algorithms, and communication protocols).

We present a comprehensive, multi-perspective approach to resource allocation problems in distributed systems. We start from an abstract high-level model describing the problem of selecting and allocating items to agents (Part I: Chapters 3–6). Next, we consider specific and more complex models of computer systems and related scheduling problems (Part II: Chapters 7–9). Finally, we analyze resource allocation problems based on real implementations in two real-life systems (Part III: Chapters 10 and 11). We show that changing the perspective from the general and relatively simple mathematical model to a specific and complex one exposes different challenges and different problems. In each case, however, we show how the resource allocation problems can be handled effectively and efficiently.

Below we give a more detailed overview of the further parts of this dissertation. Additionally, each part of this dissertation starts with an overview shortly describing the considered problems and summarizing our main contributions.

Part I. We start our analysis by considering a general problem of selecting a collective set of items ranked by the agents. Let us explain this allocation problem through an example. Consider a company that wants to provide free sport classes to its employees. There is a set of employees (who we refer to as the agents) and a set of sport classes (which we refer to as the items). The company, due to a limited budget, wants to select only K classes to be run, and then wants to assign the employees to the classes. Naturally, the company wants its employees to be as happy as possible with the provided classes.

This example is a specific case of the general problem of selecting a collective set of items for the agents. In Part I we explore its several variants, corresponding to different ways of measuring the satisfaction of the agents from the items. These variants address the following issues:

1. We address two ways in which the agents can express their preferences regarding each single item. On one hand, they can assign numerical values measuring their utilities from having particular items selected. For instance, an agent can assign utility 4 to item a_1, and utility 2 to item a_2, meaning that her level of satisfaction, according to some metric, from a_1 is twice as high as from a_2. On the other hand, they can express their preferences as rankings. For example, an agent might have preference ranking $a_1 \succ a_3 \succ \cdots \succ a_m$, meaning that for him or her a_1 is the most attractive item, a_3 is the second most attractive one, and so on, until a_m, which is the least appealing one.

2. We address different ways to measure the satisfaction of a single agent from groups of items. In the above example, since an agent is assigned to a single item, it is natural to assume that her satisfaction from the group of selected items is her satisfaction from the item to which she is assigned. We might think of other ways of measuring the satisfaction of an agent. For instance, in some scenarios it is reasonable to assume that the satisfaction of an agent from the group of items is the sum of her satisfactions from the individual items in that group. However, more complicated schemes exist too. For example, an agent might use the top preferred item certainly, the second one probably, the third one perhaps, etc., and thus her total satisfaction would be mostly influenced by her satisfaction from the top preferred item, significantly by her satisfaction from the second preferred item, slightly by her satisfaction from the third preferred item, and so on.

3. We address different ways to aggregate the satisfaction of multiple agents. For instance, we might want to maximize the sum of the utilities of the agents (the utilitarian approach) or to maximize the utility of the least satisfied agent (the egalitarian approach), or to use some yet other approach.

As we advocated before, this model captures some fundamental computational challenges from many real-life problems. Here, we only give several examples, and for the detailed discussion we refer the reader to Chapter 3. The algorithms for the problem of selecting a collective set of items can be used (i) in recommendation systems [188], (ii) for determining the proportional representation of the society in some collective body such as a parliament [47,210], (iii) for determining the optimal locations for certain facilities (for instance, to set the locations of the hospitals in a city) [57,145,269], and (iv) in the group activity selection problem [75] (for instance, to select some from the many options that the conference attendees have for a free afternoon).

We establish the computational complexity of the considered variants of the problem of selecting a collective set of items. We show polynomial-time, FPT, and exponential-time approximation algorithms. For some most interesting variants we additionally assess the quality of our algorithms through experiments, using real data describing peoples' preferences. We show that our algorithms, when evaluated on these data traces, approximate the optimal solutions significantly better than indicated by the theoretical (worst-case) guarantees. Finally, some of the most interesting variants of the problem, we show that the algorithms preserve their high quality of approximation even if we have incomplete agents' preferences, truncated to a certain number of the top positions.

Part II. In the second part of this dissertation (Chapters 7–9) we consider resource allocation problems in several more specific models. We abandon the general notion of an item and we introduce more specialized elements, such as machines with their processors, jobs that we intend to run on these machines, the durations of these jobs and the time moments they were created, the strategic interactions between the agents, an so on. These new elements allow us to consider more specific problems and their dedicated solutions.

In this part we focus on three example problems:

1. In Chapter 7 we study strategic interactions between teams of agents competing for an employment in a project. This chapter is an intermediate step in our analysis—on one hand we start from a generic model that describes general agents competing for an employment in a project and, on the other hand, we show that this generic model can be specialized with a particular scheduling model describing specific agents in a specific setting. Thus, our solutions from this chapter may be applied to various natural problems.

2. In Chapter 8 we consider a problem of finding a fair resource allocation mechanism in a distributed system consisting of multiple organizations sharing their infrastructures. This chapter concerns a concrete scheduling model in which we have a number of organizations, each owning a set of processors. The organizations merge their infrastructure and each organization can process its own jobs on any available processor. The jobs arrive continuously but neither their arrival pattern nor their durations are known in advance. The organizations want to have their jobs completed as quickly as possible, but in some busy periods there might not be enough processors to handle all incoming jobs immediately and some organizations must wait for free processors. We describe an algorithm that schedules the arriving jobs on the available processors in a way that is fair to the organizations.

3. In Chapter 9 we address load balancing problems in centrally-managed, geographically distributed systems. Similarly as in Chapter 8, we consider a concrete model of a distributed computer system. In our model there is a number of servers connected with a high-bandwidth network. The servers need to process large numbers of small user requests (such as web page requests) arriving continuously with unknown pattern. The servers can redirect requests to be handled on other servers, but each redirection causes a certain delay. We describe algorithms that efficiently compute which requests should be redirected to which servers to minimize the average user waiting time.

Since our goal is to give a comprehensive view on resource allocation problems in distributed systems, this part of the dissertation shows a diversified perspective on several aspects of resource management.

1. We show a diversified view on the agents behavior and interaction. Chapter 7 considers strategic agents and studies their competition. Chapter 8 examines the problem of finding a fair schedule from a cooperative game theory standpoint. Chapter 9 describes an organizationally centralized distributed system owned by a single entity.

2. We study different models of physical distribution of the system. Chapter 7 abstracts from the notion of a physical distribution. Chapter 8 considers a computational grid in which the communication latencies between the processors and the users are negligible when compared to the jobs' processing times. In Chapter 9 we consider a geographically distributed system in which the communication delay of the task contributes a significant part to their total completion time.

3. We use different techniques in analyzing our resource allocation mechanisms. In Chapter 7 we analyze several solution concepts from game theory. We prove the correctness of our algorithms by showing that the game-theoretic solutions have, somehow, natural structure. Consequently, we show simple and intuitive

exact algorithms. In Chapter 8 we analyze the parameterized complexity and approximability of the main discrete problem. In Chapter 9, on the other hand, we analyze the approximability of the problem in which input data is given in the form of continuous, fully divisible loads, and continuous functions.

Part III. In the two previous parts we studied relatively simple systems for which we could obtain theoretical models. The solutions obtained through analysis of these models can be, then, applied in practice. For instance, we are currently working on implementing our fair scheduling algorithms, described in Chapter 8, in CometCloud [65], an autonomic framework for running applications on supercomputers and in data centers. Some systems, however, are too complex to get an accurate mathematical model, suitable for a formal study. In this part we describe how to design resource management mechanisms for such complex systems.

There are many reasons for which the theoretical analysis of real-life complex systems is particularly difficult.

1. The quality of resource management mechanisms in some complex systems is affected by a number of factors that depend on each other, and, so, no single problem can be isolated and studied independently. For instance, let us consider the task of designing an effective backup system, while using only inexpensive and unreliable computers. Since we are forced to use low-end infrastructure, we cannot relay any important functionality of the system to a small group of computers. Indeed, such a design would be particularly vulnerable to failures. Consequently, we want to design a P2P system, i.e., a distributed system that consists of a network of computers, each having the same status and role.
   (a) To achieve good performance of such a system, we would aim at maximizing the concurrency of the backup operation and, so, we would aim to store our data on as many physical machines as possible. In our system, however, some machines may be less reliable than others and some machines may be often unavailable; using them as storage machines might result in serious problems, such as data unavailability or even data loss. Thus, in such a system, the requirements concerning reliability cannot be easily decoupled from the performance-based goals. In general, in real-life complex systems we often face multi-criteria optimization problems [91,95,167,309] in which single-criteria subproblems cannot be isolated, and thus the models get deeply complicated.
   (b) The daily availability of a machine might be a factor influencing whether such a machine can be used to store data. For instance, if a machine A is available every day between 8am and 1pm and machine B is available only between 2pm and 10pm, then A cannot directly send its data to B. On the other hand, if we could provide an effective asynchronous communication

between the machines, we might decide to use low-availability machines for storage as well. For instance, if there exists machine C that is usually available between 11am and 4pm, then A can send its data temporarily to C during the period of common availability; next, C can pass the data to B as soon as B rejoins the system. Thus, the existence of a mechanism providing asynchronous communication influences the main assumptions of a mechanism responsible for data distribution.

2. Real-life systems are often complex and they consist of many elements. Effective resource allocation requires managing many dependent and independent resources, and involves many mechanisms, such as network protocols, monitoring services, maintenance services, allocation algorithms in multiple software components, etc. Even if these mechanisms could be treated as black boxes and designed and studied independently from each other (which is very often impossible), it is particularly difficult to analyze the system and its properties as a whole.

3. Complex systems might have multiple contradicting goals. Let us, again, consider an example of a P2P backup system. To achieve resiliency to geographically-correlated disasters (such as earthquakes, floods, or fires) we would like to keep data in remote locations. This, however, stays in contradiction with the performance-oriented goals. It is not clear how to precisely describe, in a model, this trade-off between many contradicting goals.

In addition to the above discussion, we often are not aware which elements of the whole system are important and need to be taken into account during the design process. Consequently, it is not only hard to provide a suitable formal analysis of resource management mechanisms in real-life complex systems, but also it is hard to propose simplifications that would allow one to run simulations of some parts of the system. We can very often use wrong assumptions and miss the important factors that need to be taken into account, simply because we are not fully aware of them. Thus, we argue that in some real complex systems we require a methodology that focuses on emulation rather than on simulation or formal analysis. In such cases we need to perform the experiments on a real system, to confirm its desired properties.

In this part of the dissertation we address the issue of designing effective resource management mechanisms in real-life complex systems, where the formal methodology and simulations cannot be successfully applied.

In Chapter 10 we consider a real resource allocation problem that is taken from HYDRAstor [89,218], the commercial storage system developed by NEC Corporation [219]. In this system we address the problem of distributing resources between loads of various types, such as user reads and writes, reconstructions of missing parity data, data defragmentation tasks, and other background activities.

Since we are given an existing commercial system, our goal is to design a mechanism that is decoupled from the rest of the system. Thus, this mechanism cannot interfere with other parts of the system and, in particular, cannot influence the way in which data is distributed between physical nodes. We describe a fuzzy adaptive control mechanism for sharing resources among various types of highly-variable loads. This mechanism ensures proper division of the total system throughput between different categories of the tasks and maintains high resource utilization. Additionally, it does not influence other parts of the system.

In Chapter 11 we consider a different approach to designing resource management mechanisms. In this approach we do not want to provide a decoupled resource management mechanism, but instead we present a whole architecture for a P2P backup system that allows one to implement certain resource allocation mechanisms. For instance, this architecture supports data placement strategies that allow one to achieve certain goals, according to a given policy. These goals include reliability to geographically correlated disasters (placing data replicas far away from each other), performance of read/write operations (placing replicas on strong, non-overloaded servers), or putting small burden on the network (placing data replicas close to each other). We also present other mechanisms supporting effective resource management, such as mechanisms allowing for asynchronous communication.

To sum up, in this dissertation we show a comprehensive and multi-perspective view on resource management in distributed systems. We show how to effectively and efficiently solve the resource allocation problems. High-level contributions of this thesis are as follows:

1. We describe new models that capture resource allocation problems, in particular new classes of cooperative and strategic games (Chapters 3, 7, 8, and 9).

2. We analyze computational complexity of various resource allocation problems (Chapters 4–9).

3. We show algorithms (exact, approximate, fixed parameter tractable, heuristic, centralized, and distributed) for some fundamental problems regarding resource allocation in distributed systems (Chapters 4–11).

4. We show how to apply non-cooperative game-theoretic (Chapter 7) and cooperative game-theoretic (Chapter 8) solution concepts to ensure stability of resource allocation algorithms in some example distributed systems.

5. We show how to design resource allocation mechanisms in two real-life distributed systems—in HYDRAstor, a high-performance secondary-storage system aimed at the enterprise market (Chapter 10), and in our prototype implementation of a P2P backup system (Chapter 11).

6. We perform experimental evaluation of some of our mechanisms. We perform simulations to show good approximation properties of our algorithms (Chapters 5, 8, and 9), and their small convergence time (Chapter 9). We perform experiments on real systems to show that our algorithms are stable, that they result in high resource utilization and satisfy resource allocation objectives (Chapters 10, and 11).

We believe that three most valuable technical results presented in this thesis are the following.

1. In Chapter 5 we show several good approximation algorithms for winner determination under two appealing election systems—the Monroe and Chamberlin–Courant systems. Actually, we prove stronger results, giving guarantees on the satisfaction of agents, independently of the preference profile or the quality of the optimal solution. For instance, for the Polish parliamentary elections, where the the parliament consists of 460 members and the number of candidates is approximately equal to 6000, our algorithms guarantee that each voter is, on average, represented by a candidate that she prefers to 99% of the candidates (in case of Chamberlin–Courant system) and to 96% of the candidates (in case of Monroe system). We believe that our algorithms will, eventually, make it possible to use the two appealing election systems in practice.

2. In Chapter 6 we describe approximation algorithms that run in FPT time for the MaxCover problem. MaxCover is a very useful theoretic problem that finds applications in several fundamental resource allocation problems.

3. In Chapter 8 we present how to apply game theoretic solution concept to create a fair scheduling algorithm. Our algorithm operates without using the concept of money, which makes it particularly practical. We show several practical effective scheduling algorithms that give good fairness guarantees.

Our results have been published in the following conferences (6 of them are currently under review in various journals):

1. P. Skowron, K. Rzadca and A. Datta. People are Processors: Coalitional Auctions for Complex Projects (Extended Abstract). *In Proceedings of 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2014)* [282],

2. P. Skowron, P. Faliszewski and A. Slinko. Fully Proportional Representation as Resource Allocation: Approximability Results. *In Proceedings of The 2013 International Joint Conference on Artificial Intelligence (IJCAI-2013)* [276],

3. P. Skowron and K. Rzadca. Non-monetary fair scheduling: a cooperative game theory approach. *In Proceedings of 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA-2013)* [281],

4. P. Skowron, P. Faliszewski and A. Slinko. Achieving Fully Proportional Representation is Easy in Practice. *In Proceedings of 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2013)* [275],

5. P. Skowron, M. Biskup, L. Heldt and C. Dubnicki. Fuzzy adaptive control for heterogeneous tasks in high-performance storage systems. *In Proceedings of 6th Annual International Systems and Storage Conference (SYSTOR-2013)* [273],

6. P. Skowron and K. Rzadca. Fair Share Is Not Enough: Measuring Fairness in Scheduling with Cooperative Game Theory. *In Proceedings of 10th International Conference on Parallel Processing and Applied Mathematics (PPAM-2013)* [279],

7. P. Skowron and K. Rzadca. Network Delay-Aware Load Balancing in Selfish and Cooperative Distributed Systems. *In Proceedings of 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW-2013)* [280],

8. P. Skowron and K. Rzadca. Exploring heterogeneity of unreliable machines for P2P backup. *In Proceedings of The 2013 International Conference on High Performance Computing & Simulation (HPCS-2013)* [278].

The contents of Chapter 10 are patented (US Patent 2013 nr. 20130031563, and WO Patent 2012 nr. WO/2012/029259). Yet for the sake of clarity of the presentation we do not include these extended results in the main text. Our results have also been presented at a number of workshops, such as:

1. The Fourth Workshop on Cooperative Games in Multiagent Systems (CoopMAS). Saint Paul, USA, 2013 [276,281],

2. Forum Informatyki Teoretycznej FIT (An informal annual meeting of Polish researchers working on theoretical computer science) in Toruń, Poland in 2013 and in Jarnołtówek, Poland in 2014 [274–276],

3. 6th Multidisciplinary Workshop on Advances in Preference Handling (M-PREF-2012). Montpellier, France, 2012 [276],

4. New Challenges in Scheduling Theory Workshop (Fréjus, France, 2012 and Aussois, France, 2014) [280,281], and

5. The Workshop on Economic and Computational Aspects of Game Theory and Social Choice. Warsaw, Poland, 2014 [274,282].

Acknowledgements. The research presented in this thesis was supported by Polish National Science Centre grants Preludium UMO-2013/09/N/ST6/03661, Harmonia UMO-2012/06/M/ST1/00358, and Sonata UMO-2012/07/D/ST6/02440, by Foundation for Polish Science Homing Plus grant HOMING PLUS/2010-2/13 and by EU's Human Capital Program "National PhD Programme in Mathematical Sciences" carried out at the University of Warsaw.

Chapter 2

Preliminaries

In this chapter we introduce our notation and recall fundamental concepts from the cooperative and strategic game theory (Section 2.1), and from the (parameterized) complexity theory (Section 2.2). We conclude this chapter by giving a dictionary and a brief overview of the NP-hard problems used in our further discussions (Section 2.3).

2.1 Game Theory

Throughout this dissertation we often refer to *agents*. Agents are autonomous, intelligent, rational, and goal-oriented entities, which make decisions that affect the systems that we study. Agents can represent human beings, organizations, pieces of software, or intelligent computer systems, depending on the setting.

Game theory is a study of decision making by agents. Games formalize strategies the agents can choose from and the outcomes of using these strategies, i.e. the results of the players' chosen actions. In game theory, a solution concept is a formal rule that describes how rational agents would behave. It allows to predict the strategies of the agents and the results of the game.

In this section we review several solution concepts from non-cooperative and cooperative game theory. These concepts commonly assume that the agents are selfish and rational, i.e., that each agent aims at maximizing her own profit. The cooperative approach differs from the non-cooperative one in the way agents interact. In particular, the cooperative game theory considers scenarios where it is possible and profitable for agents to cooperate and to form binding agreements. Consequently, the solution concepts from the cooperative game theory aim at defining the ways in which the total profit of the cooperative coalition should be distributed among its participants to ensure fairness and stability of the formed coalition. The non-cooperative game theory, on the other hand, studies scenarios where agents make their decisions independently. Thus, in the non-cooperative view there is no concept of a binding agreement, and, so, the solution concepts aim at predicting agents' strategies (predicting how a game will be played) rather than describing conditions for profit distributions that guarantee stable agreements. For the more detailed discussion on the cooperative and non-cooperative game theoretic solution concepts we refer the reader to the book of Osborne and Rubinstein [231].

2.1.1 Non-Cooperative Game Theory

Formally, a game is defined by a set of agents, sets of their available strategies (also referred to as their actions), and a set of their payoff functions (which will be defined later). We usually denote the set of agents as $N = [n]$, where for each $n \in \mathbb{N}$, by $[n]$ we mean $\{1, \ldots, n\}$. The set of actions of agent i is denoted by S_i. A strategy profile $\vec{x}$ is a vector of actions of all agents; i.e., $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$, where $x_i \in S_i$ for every $i \in N$. By (x'_i, x_{-i}) we denote the strategy profile x after replacing the i-th agent's strategy x_i with x'_i. We define S as the set of all strategy profiles: $S = S_1 \times S_2 \times \cdots \times S_n$. The payoff function of agent i, $f_i : S \to \mathbb{R}$, defines for every strategy profile $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ the payoff that agent i receives, provided every agent $j \in N$ takes action x_j; intuitively, the agents choose their strategies to maximize their payoff.

There are many known solution concepts in the strategic game theory, some of which we briefly describe below.

Nash Equilibrium

Perhaps the most famous solution concept in game theory is the Nash equilibrium [217]. Intuitively, Nash Equilibrium describes a state in which no agent can benefit from changing her strategy. In other words, in Nash Equilibrium every agent is playing her best action, given the strategies of the others.

Definition 2.1. *A strategy profile $x^* \in S$ is a Nash Equilibrium if no unilateral deviation in strategy by any single agent is profitable for that agent. That is, $x^* \in S$ is a Nash Equilibrium if it holds that:*

$$\forall i, x_i \in S_i \colon f_i(x_i, x^*_{-i}) \leq f_i(x^*_i, x^*_{-i}).$$

Nash equilibria do not necessarily exist, and, indeed, this is the case for many natural games [231]. On the other hand, we can consider the concept of a Nash Equilibrium that is defined for *mixed strategies*, where players choose a probability distribution over their actions and are interested in maximizing their expected utility. It is natural to ask the question about the existence of Nash Equilibrium in mixed strategies. In 1950, Nash proved one of the most famous result in game theory, saying that under mild assumptions about the game, Nash equilibria in mixed strategies are guaranteed to exist [217].

Strong Nash Equilibrium

The notion of the Nash equilibrium implicitly assumes that agents cannot (or do not want to) communicate and, thus, that they make their decisions in isolation. In many real-life scenarios, however, this is not the case and the agents can coordinate their strategies. For such games, Aumann [15] proposed a refinement of the concept of a Nash Equilibrium known as the Strong Nash Equilibrium (SNE).

In the definition below we extend the notation (x'_i, x_{-i}) in a natural way so that for a given subset of agents $N' = \{i_1, \dots i_{|N'|}\} \subseteq N$ and a vector of their actions $x_{N'} = \langle x'_{i_1}, \dots x'_{i_{|N'|}} \rangle$, by $(x_{N'}, x_{-N'})$ we mean the strategy profile that we obtain after replacing for each $i \in N'$ the i-th agent's strategy x_i with x'_i.

Definition 2.2. *A strategy profile $x^* \in S$ is a* Strong Nash Equilibrium (SNE) *if no unilateral deviation in strategy by any* set of agents *is profitable for every deviating agent, i.e., x^* is an SNE if:*

$$\Big(\forall N' = \{i_1, \dots i_{|N'|}\} \subseteq N, x_{N'} = \langle x'_{i_1}, \dots x'_{i_{|N'|}} \rangle\Big)\Big(\exists i \in N'\Big) \colon f_i(x_{N'}, x^*_{-N'}) \leq f_i(x^*).$$

The concept of the SNE is very restrictive and there are relatively few games for which an SNE exists. Unfortunately, there is no result analogous to the existence of Nash Equilibria for the case of mixed strategies. For this reason, SNE has been criticized as being too strong. In effect, alternative, weaker, concepts, such as coalition-proof Nash equilibrium [23] and Coalitional Farsighted (Conservative) Stable Set [82], have been proposed. Since these two solution concepts are less common, we recall their definitions and provide a discussion of their applications in Chapter 7, which is the only place in this dissertation where we use them.

Pareto Efficiency

Pareto Efficiency [184], also referred to as Pareto Optimality, is another concept of stability. It defines a state in which we cannot improve the payoff of any agent without reducing the payoff of some other one. Thus, Pareto Efficiency also defines the concept of global optimality: non Pareto-efficient states can undoubtedly be improved upon with no harm to any agent.

Definition 2.3. *A strategy profile $x^* \in S$ is Pareto-efficient if there exists no profile $x \in S$ such that (i) every agent under x gets at least as good a payoff as under x^*, and (ii) there exists some agent that under x gets strictly better payoff than under x^*. Formally, x^* is Pareto-efficient if:*

$$\forall x \in S \colon \big(\exists i \in N \colon f_i(x) < f_i(x^*)\big) \ \textit{ or } \ \big(\forall i \in N \colon f_i(x) = f_i(x^*)\big).$$

The concept of Pareto Efficiency is also popular in the context of multi-criteria optimization [91,95,167]. If we want to optimize a function with several independent

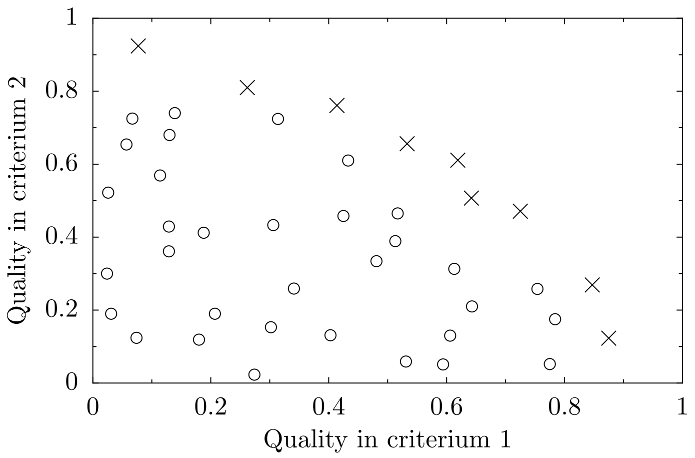

Figure 2.1: Graphical illustration of the Pareto optimal set in the context of bi-criteria optimization. The Pareto optimal set consist of the elements marked with the cross.

criteria, we ideally want to find a Pareto optimal solution, i.e., a solution for which there is no single criterion in which the optimization function can be improved without reducing its quality according to some other criterion. The concept of Pareto Efficiency is graphically illustrated in Figure 2.1.

2.1.2 Cooperative Game Theory

Cooperative game theory analyzes scenarios where agents work together to achieve some goal. In the cooperative game theory we often refer to the sets of agents as the *coalitions*. Formally, a cooperative game is defined by the set of agents N and the *characteristic function*: $v : 2^{\mathbb{N}} \to \mathbb{R}$ that describes how much payoff each coalition of agents $\mathcal{C} \subseteq N$ can get. We assume that the empty coalition (empty set of agents) cannot gain any payoff, $v(\emptyset) = 0$. Additionally, in the cooperative game theory the characteristic function is often assumed to satisfy superadditivity.

Definition 2.4. *A set function $v : 2^{\mathbb{N}} \to \mathbb{R}$ is* superadditive *if it satisfies the following condition:*

$$\textit{for every } S, T \subseteq N \textit{ such that } S \cap T = \emptyset, \textit{ we have } \quad v(S \cup T) \geq v(S) + v(T).$$

Alternatively, it is also often assumed that the characteristic function is cohesive.

Definition 2.5. *A set function $v : 2^{\mathbb{N}} \to \mathbb{R}$ is* cohesive *if for every collection $S_1, \ldots, S_k$ of pairwise disjoint subsets of N whose union is N, it holds that:*

$$\sum_{i=1}^{k} v(S_i) \leq v(N).$$

Intuitively, superadditivity and cohesiveness state that players can collectively achieve a higher value than in separated coalitions. Each of these conditions is, in some sense, a minimal natural requirement for agents to form a *grand coalition*, a coalition consisting of all agents. A stronger assumption to superadditivity, is convexity.

Definition 2.6. *A game is* convex *if* $v(\emptyset) = 0$ *and for every* $S, T \subseteq N$*, we have:*

$$v(S \cup T) + v(S \cap T) \geq v(S) + v(T).$$

Solution concepts in cooperative game theory define rational divisions of the total value of the grand coalition $v(N)$ between its members. Such divisions are often referred to as *payoff vectors*. Payoff vector $\vec{x}$ is a vector of n nonnegative values $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$, where the i-th value x_i denotes the payoff allocated to the i-th agent. There are many known solution concepts in the cooperative game theory [231] including, e.g., the stable set [212], the core [112], the kernel [77], the nucleolus [257], and the Shapley value [265]. Below, we briefly describe those three that we use in the further parts of this dissertation.

The Shapley Value

The Shapley value [265] is an established solution concept in the cooperative game theory, whose goal is to define a fair division of the total value of the coalition between its participants. The Shapley value of the player i in a coalitional game (N, v) is:

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \Big(v(S \cup \{i\}) - v(S) \Big).$$

We often refer to the value $(v(S \cup \{i\}) - v(S))$ as the *marginal contribution* of the i-th agent to the coalition S. Thus, the Shapley value of an agent i can be viewed as a weighted marginal contribution of i. This view can be formalized by the following alternative definition of the Shapley value [231].

Let $\mathcal{L}_N$ denote all possible orders over the set of agents N. Each order $\prec_N$ can be associated with a permutation of the set N, so $|\mathcal{L}_N| = |N|!$. For each order $\prec_N \in \mathcal{L}_N$ we define $\prec_N(i) = \{j \in N : j \prec_N i\}$ to be the set of all agents that precede i in the order $\prec_N$. The Shapley value can be expressed in the following form:

$$\phi_i(v) = \frac{1}{|N|!} \sum_{\prec_N \in \mathcal{L}_N} \Big(v\big(\prec_N(i) \cup \{i\}\big) - v\big(\prec_N(i)\big) \Big). \tag{2.1}$$

As we noted, this formulation has an interesting interpretation. The Shapley value, $\phi_i(v)$ is the expected marginal contribution of the agent i to the grand coalition, provided that we choose the order in which agents join uniformly at random.

The Shapley value is characterized by a number of desirable properties [129,305]. Below, we recall the first axiomatic characterization proposed by Shapley [231]: The

Shapley value is the unique map from the set of all games to payoff vectors (naturally, in this case $\langle \phi_1(v), \phi_2(v), \ldots, \phi_n(v) \rangle$ is a payoff vector) that satisfies the following four properties:

Efficiency. The total value $v(N)$ is distributed:

$$\sum_{i \in N} \phi_i(v) = v(N).$$

Symmetry. If the agents i and j have the same marginal contributions, they need to obtain the same profits:

$$\Big(\forall_{S \subset N : i,j \notin S}\ v(S \cup \{i\}) = v(S \cup \{j\})\Big) \Rightarrow \phi_i(v) = \phi_j(v).$$

Additivity. If we combine two games (N, v) and (N, w), into a new game $(N, v+w)$, defined as $(v + w)(S) = v(S) + w(S)$, then for every $i \in N$, we have:

$$\phi_i(v + w) = \phi_i(v) + \phi_i(w).$$

Dummy (Null player). An agent that does not increase the value of any coalition $S \subset N$ gets nothing:

$$\Big(\forall_{S \subset N} : v(S \cup \{i\}) = v(S)\Big) \Rightarrow \phi_i(v) = 0.$$

The Core

The core [112] is the concept from the cooperative game theory that is very closely related to that of Strong Nash Equilibrium from the non-cooperative game theory. The core describes the set of payoff vectors which are stable, in the sense that no coalition of agents can deviate so that all its members are better off.

Definition 2.7. *The core of the cooperative game (N, v) is the set of all payoff vectors, such that for each such a vector $\vec{x} = \langle x_i \rangle_{i \in N}$ it holds that (i) the total value of the coalition is distributed $\sum_i x_i = v(N)$, and (ii) there exists no coalition $S \subseteq N$, and no payoff vector $\langle y_i \rangle_{i \in N}$, such that $\sum_i y_i = v(N)$ and $y_i > x_i$, for all $i \in S$.*

Similarly to the Shapley value, the core can be characterized alternatively to Definition 2.7, by the two following axioms:

Efficiency: $\sum_i x_i = v(N)$.

Coalitional rationality: $\forall S \subseteq N\colon\ \sum_i x_i \geq v(S)$.

Consequently, the core is the set of vectors satisfying a system of linear inequalities, and, so, it is closed and convex.The core is always well-defined, but can be empty; emptiness of the core is analogous to the cases in non-cooperative games when Nash equilibria do not exist.

The Stable Set

The stable set [212] (also known as the von Neumann-Morgenstern stable set) was one of the first solution concepts proposed for the cooperative games. The stable set describes the set of all payoff vectors that dominate the payoff vectors from outside the set, and, at the same time, are not dominated by any other payoff vectors from the set.

Definition 2.8. *A payoff vector $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ is* dominated *by a payoff vector $\vec{y} = \langle y_1, y_2, \ldots, y_n \rangle$, when $\vec{y}$ is preferred to $\vec{x}$ by all the members of some coalition $\mathcal{C}$, i.e., if there exists a coalition $\mathcal{C} \neq \emptyset$ such that:*

1. *For each $i \in \mathcal{C}$ it holds that $y_i > x_i$, and*
2. *$\sum_{i \in \mathcal{C}} y_i \leq v(\mathcal{C})$.*

Definition 2.9. *A set S of payoff vectors is called a stable set if it satisfies two properties:*

1. *No payoff vector in S is dominated by another vector in S.*
2. *Each payoff vector outside S is dominated by some vector from S.*

The idea behind the stable set is the following. A coalition $\mathcal{C}$ is not satisfied with the payoff vector when it can propose an alternative division of the total value that is more profitable for all of its members. Such coalition may threat the stability of the grand coalition agreement by breaking out and implementing their proposed division without the other agents. However, coalition $\mathcal{C}$ will have a unilateral incentive to break out only if the newly proposed division is stable, as otherwise other coalitions may object to the proposal of $\mathcal{C}$, and consequently, after such a sequence of objections, some members of $\mathcal{C}$ may end up in a worse than initial state. If we select a payoff vector from the stable set, then no coalition will have such unilateral incentive to break out and implement an alternative payoff vector.

The stable set may not exist and if it exists it is usually not unique. There is an elegant connection between the stable set and the core. First, for each cooperative game its core is a subset of each of its stable sets (we recall that the stable set might not be unique). Second, if the core is exactly equal to some of the games' stable sets, then for such game the stable set is unique, and, so, the notions of the core and of the stable set are equivalent. Interestingly, for convex games there exists a unique stable set that coincides with the core; also, for these games the core contains the Shapley value.

2.2 Complexity and Algorithms

We assume familiarity with standard notions pertaining to algorithms and worst-case complexity theory (such as definitions of the decision and optimization problems, the classes P and NP, many-one reductions, NP-completeness, etc.). Below we review the most essential concepts regarding parameterized complexity and approximation algorithms.

2.2.1 Parametrized Computational Complexity

We sometimes use results from the theory of parameterized complexity developed by Downey and Fellows [87]. This theory allows one to single out a particular parameter of the problem, say K, and analyze its contribution to the overall complexity of the problem. An analogue of the class P here is the class FPT which consists of problems that can be solved in time $f(K)n^{O(1)}$, where n is the size of the input instance, and f is some computable function.

For example, consider the VertexCover problem in which we are given a graph and our goal is to find the smallest possible subset C of vertices such that each edge of the given graph is incident to some vertex from C. One of the natural parameters for VertexCover is the size of the optimal solution, $K = |C|$. VertexCover can be solved by an exhaustive search in time $O(2^K Kn)$. (The key observation to obtain the mentioned complexity is that each edge can be covered by only two vertices, and that in the optimal solution all edges need to be covered.) This algorithm has exponential running time, but the exponential part of its complexity depends only on the value of the parameter K. Thus, VertexCover for the parameter K is fixed parameter tractable (FPT).

From the point of view of parameterized complexity, FPT is seen as the class of tractable problems. There is also a whole hierarchy of hardness classes, FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots$ W[P] $\subseteq \cdots$. The standard definitions of W[1], W[2], ... are quite involved and so, instead of providing them here, we point the reader to appropriate overviews [87,102,222]. However, we can also define these classes through a notion of an appropriate reduction to their complete problems.

Definition 2.10 (Parametrized reduction [87]). *Let $\mathcal{P}$ and $\mathcal{P}'$ be two decision problems parameterized by natural number parameters $\mathcal{K}$ and $\mathcal{K}'$, respectively. We say that $\mathcal{P}$ reduces to $\mathcal{P}'$ through a parameterized reduction if there exist a mapping $F\colon \mathcal{P} \to \mathcal{P}'$ (computable in FPT time with respect to parameter $\mathcal{K}$) and two computable functions, $g\colon \mathbb{N} \to \mathbb{N}$ and $h : \mathbb{N} \to \mathbb{N}$, such that:*

1. *for each instance $(I, K) \in \mathcal{P}$ the answer to (I, K) is "yes" if and only if the answer to $F(I) = (I', K')$ is "yes",*

2. *K and K' are the values of the parameters $\mathcal{K}$ and $\mathcal{K}'$ respectively,*

3. *$|I'| \leq g(K)\mathrm{poly}(|I|)$, and*

4. *$K' \leq h(K)$.*

The parameterized reduction (sometimes also referred to as an FPT reduction) simultaneously preserves the instance size (point (iii) in the above definition) and the size of the parameter (point (iv)). The size of the instance I' does not have to be bounded by a polynomial of the size of I, but the exponential part of the relation between the sizes of I and I' must depend only on the value of the parameter.

W[1] is the class of all problems for which there is a parameterized reduction to the CLIQUE problem (with parameter K). In the CLIQUE problem we are given a graph G and we ask whether in G there exists a set S of K vertices such that every two vertices from S are connected. W[2] is the class of problems with parameterized reductions to SETCOVER (with parameter K). In the SETCOVER problem we are given a set of elements N and a set $\mathcal{F}$ of subsets of N. We ask whether there exist K subsets from $\mathcal{F}$ such that each element of N belongs to at least one of these subsets. We define and describe most relevant computational properties of the problems CLIQUE and SETCOVER in Section 2.3.

2.2.2 Approximation

Let $\mathcal{P}$ be an algorithmic problem where, given some instance I, the goal is to find a solution s that maximizes a certain function f. We call such problems maximization problems. Given an instance I of $\mathcal{P}$, we refer to the value $f(s)$ of an optimal solution s as $\mathrm{OPT}(I)$ (or, sometimes, simply as OPT if the instance I is clear from the context).

Definition 2.11 (Approximation algorithm)**.** *Let β, $0 < \beta \leq 1$, be some fixed constant. Let $\mathcal{A}$ be an algorithm, for a maximization problem $\mathcal{P}$, that given an instance I returns a solution $\mathcal{A}(I)$. $\mathcal{A}$ is called a β-approximation algorithm for the problem $\mathcal{P}$ if for every instance I of $\mathcal{P}$ it holds that $f(\mathcal{A}(I)) \geq \beta\mathrm{OPT}(I)$.*

Analogously, we define $\mathrm{OPT}(I)$ and the notion of a γ-approximation algorithm, $\gamma > 1$, for the case of minimization problems, where the task is to find a solution that minimizes a given goal function g. Given an instance I of such a minimization problem $\mathcal{P}'$, a γ-approximation algorithm is required to return a solution s' such that $g(s') \leq \gamma\mathrm{OPT}(I)$.

We are particularly interested in settings where it is possible to obtain arbitrarily good approximation algorithms.

Definition 2.12 (PTAS)**.** *A polynomial time approximation scheme (PTAS) for a maximization (minimization) problem $\mathcal{P}$ is an algorithm that for every $\epsilon > 0$ provides a polynomial $(1-\epsilon)$-approximation algorithm (a polynomial $(1+\epsilon)$-approximation algorithm) for $\mathcal{P}$.*

Definition 2.13 (FPTAS)**.** *A fully polynomial time approximation scheme (FPTAS) for a maximization (minimization) problem $\mathcal{P}$ is an algorithm that for every $\epsilon > 0$, and for every instance I of $\mathcal{P}$ provides an $(1-\epsilon)$-approximation (an $(1+\epsilon)$-approximation) solution for I in time polynomial in the instance size $|I|$ and the approximation parameter $1/\epsilon$.*

In our analysis we sometimes use the powerful result of Nemhauser et al. [220], which says that greedy algorithms achieve $1 - \frac{1}{e}$ approximation ratio when used to optimize nondecreasing submodular functions. Below, we explain what are the nondecreasing and submodular functions and recall the result of Nemhauser et al..

Definition 2.14 (Nondecreasing set function)**.** *A set function $z : 2^N \to \mathbb{R}$ is nonincreaing if for every $S_1 \subseteq S_2 \subseteq N$ we have $z(S_1) \leq z(S_2)$.*

Definition 2.15 (Submodular set function)**.** *Let $z : 2^N \to \mathbb{R}$ be a set function defined for each set $S \subseteq A$. We say that z is submodular if for every $S_1 \subseteq S_2 \subseteq N$ and every $x \in N \setminus S_2$ we have that:*

$$z(S_1 \cup \{x\}) - z(S_1) \geq z(S_2 \cup \{x\}) - z(S_2).$$

There are many other equivalent definitions of submodularity [220], but throughout this dissertation we will only use the definition given above.

Theorem 2.1 (Approximating submodular functions [220])**.** *Let $z : 2^N \to \mathbb{R}$ be a nondecreasing submodular set function. Consider the problem $\mathcal{P}$ of selecting $S \subseteq N$ such that $|S| = K$ and $z(S)$ is maximal. The greedy algorithm that starts from an empty solution $S = \emptyset$ and in each of K iterations adds to the solution the element x that maximizes $z(S \cup \{x\})$, is an $(1 - 1/e)$–approximation algorithm for $\mathcal{P}$.*

2.3 Overview of NP-hard Problems

Below we recall definitions of, and provide some background information for, some NP-hard problems used throughout our discussions.

We start from defining the SAT and 3-SAT problems. These are perhaps most fundamental problems known in the complexity theory. In particular, SAT is famous as the first known example of an NP-complete problem, as proved by Cook in 1971 [67] and, independently, by Levin in 1973 [174]. These results were a significant breakthrough in complexity theory. Below we provide definitions of the two problems—these definitions will be useful in our further discussions, but we will not use them explicitly in our proofs of hardness.

Definition 2.16. *In the* SAT *problem we are given a propositional formula, that is built from (i) variables, (ii) operators "and" (conjunction), "or" (disjunction), and "not" (negation), and (iii) parentheses. We ask whether there exists an assignment of the logical values "true", and "false" to the variables, that the given formula is true.*

Definition 2.17. *The* 3-SAT *problem is defined analogously to the* SAT *problem, with the single difference, that the propositional logic formula is given in the specific form: it is a conjunction of clauses, each clause being a disjunction of at most three literals, where a literal is either a variable or its negation.*

2.3.1 Covering Probelms

In many of our hardness proofs we use reductions from various covering problems, that is, different variants of the SetCover problem, defined below.

Definition 2.18 (SetCover)**.** *An instance I of* SetCover *consists of set $U = [n]$ (called the ground set), family $\mathcal{F} = \{F_1, F_2, \dots, F_m\}$ of subsets of U, and of a positive integer K. We ask if there exists a set $I \subseteq [m]$ such that $\|I\| \leq K$ and $\bigcup_{i \in I} F_i = U$.*

Set-Cover remains NP-complete even if we restrict each member of U to belong to at most two sets from $\mathcal{F}$. The simple greedy algorithm that in each step selects a set that covers the largest number of yet-uncovered elements gives approximation guarantee $O(\log(n))$ [60] (where n denotes the number of elements). This is the best possible approximation ratio that a polynomial-time algorithm may achieve [98]. If the frequency of the elements, defined as the number of sets an element can belong to, is bounded by a constant p, then an algorithm based on LP-relaxation gives approximation guarantee equal to p [293]. Set-Cover is also hard from the perspective of parameterized complexity. For the parameter K, denoting the number of sets in the optimal cover, it is W[2]-complete. (Indeed, in this thesis we define W[2] as the class of problems that FPT-reduce to Set-Cover.)

SetCover has several interesting variants.

Definition 2.19 (X3C)**.** Exact3SetCover *(*X3C*) is a variant of* Set-Cover *where $\|U\|$ is divisible by 3, each member of $\mathcal{F}$ has exactly three elements, and $K = \frac{\|U\|}{3}$.*

X3C remains NP-complete even if we additionally assume three sets from $\mathcal{F}$ [107].

In SetCover we minimize the number of sets that we have to pick to cover the whole ground set. Instead, we might want to maximize the number of ground set elements that we can cover using some K sets from $\mathcal{F}$. This is captured by the MaxCover problem.

Definition 2.20 (MaxCover)**.** *In the* MaxCover *problem we are given a set N of n elements, a family $\mathcal{S} = \{S_1, \dots, S_m\}$ of m subsets of N, and an integer K. The goal is to find a size-at-most-K subcollection of $\mathcal{S}$ that covers as many elements from N as possible.*

MaxCover is NP-hard by a simple reduction from SetCover. The greedy algorithm that in each of K iterations adds to the solution a set that covers most yet uncovered elements, achieves approximation ratio $(1 - 1/e)$ [135] (the same result can be also obtained by applying Theorem 2.1, and this is optimal unless P = NP [98]. Surprisingly (by comparison to SetCover), there are no known results on approximating the MaxCover problem with the bounded frequencies of the elements. From the point of view of the parameterized complexity, MaxCover with no bounds on frequencies of the elements is known to be W[1]-complete [126].

There is also an FPT algorithm for MaxCover, for parameter T, i.e., the number of elements to be covered, due to Bläser [32]. Establishing the parameterized complexity and parameterized approximability of MaxCover with frequencies bounded by a constant is a problem that we study in Chapter 6.

2.3.2 Graph Problems

In some hardness proofs it is convenient to take advantage of some additional structure, for instance to use graphs. A graph $G = (V, E)$ consists of a set of objects V (referred to as *vertices*) where some pairs of objects are connected by edges (with E denoting the set of all edges). In graph theory we distinguish *undirected* and *directed* graphs. In the undirected graphs the edges are symmetric, that is if a vertex v is connected by an edge e with a vertex u, it implies that u is also connected with v by the same edge e. Consequently, in undirected graphs an edge can be defined as a two-element subset of vertices. This is not the case for the directed graphs, where we can rather think of the edges as of the ordered pairs of vertices. In this dissertation we will mostly use undirected graphs. Further, we will always assume that there are no loops in the considered graphs, that is, that a vertex cannot be connected with itself by a single edge.

One of the basic notions regarding undirected graphs that we will further use is the *degree* of a vertex, which is the number of edges adjacent to this vertex. Similarly, for directed graphs one can define the *in-degree* and the *out-degree* of a vertex, as the number of edges pointing at the vertex and pointing away from the vertex, respectively.

We will be particularly interested in one specific class of undirected graphs—in bipartite graphs.

Definition 2.21. *An undirected graph $G = (W, E)$ is* bipartite *if its set of vertices W can be divided into two disjoined sets, U and V, such that every edge $e \in E$ connects some vertex from U with some vertex from V (there are no edges between any two vertices from U, and between any two vertices from V). Such bipartite graph will hereinafter be denoted as $G = (U \cup V, E)$.*

Definition 2.22. *A bipartite graph $G = (V \cup U, E)$ is* balanced *if $|V| = |U|$.*

For more details on the graph theory we refer the reader to the book of West [302].

Some graph problems are special cases of the covering problems. For instance, the VertexCover problem defined below is a special case of the SetCover problem.

Definition 2.23 (VertexCover)**.** *In the* VertexCover *problem we are given an undirected graph $G = (V, E)$, where $V = \{v_1, \ldots, v_m\}$ and $E = \{e_1, \ldots, e_n\}$, and a positive integer K. We ask if there is a set C of up to K vertices such that each edge is incident to at least one vertex from C.*

Indeed, if we identify the edges with the elements and the vertices with the sets (saying that a vertex "contains" its all incident edges), we see that VertexCover is a special case of SetCover with the frequencies of the elements equal to two: each edge is incident to exactly two vertices, and, so, it is contained by exactly two sets that correspond to these vertices. However, the two problems, VertexCover and SetCover with element frequencies equal to two, are not equivalent—in VertexCover every pair of vertices is connected by at most one edge. This means that for every pair of sets that correspond to two different vertices, these sets have at most one common element. In the frequency-bounded variant of SetCover, on the other hand, any two sets may have arbitrarily many common elements.

Analogously, the CubicVertexCover problem, defined below, is a special case of the X3C problem with the frequencies of the elements equal to two.

Definition 2.24 (CubicVertexCover [8])**.** *The* CubicVertexCover *problem is identical to the standard* VertexCover *problem, except that each vertex in the input graph has degree equal to three.*

Both VertexCover and CubicVertexCover are NP-hard [8,107].

Finally, the MaxVertexCover is the special case of the MaxCover with the frequencies of the elements equal to two (though, again, we stress that MaxCover with frequencies bounded by two is strictly more general than MaxVertexCover.).

Definition 2.25 (MaxVertexCover [8])**.** *In the* MaxVertexCover *problem we are given an undirected graph $G = (V, E)$, and two positive integers—K and T. We ask if there is a set C of up to K vertices such that there exists at least T edges, each incident to at least one vertex from C.*

To the best of our knowledge, the best polynomial-time approximation algorithm for MaxVertexCover is due to Ageev and Sviridenko [5], and achieves approximation ratio of $\frac{3}{4}$. However, in various settings, it is possible to achieve better results; we mention the papers of Han et al. [128] and of Galluccio and Nobili [106] as examples.

From the point of view of parameterized complexity, MaxVertexCover was first considered by Guo et al. [126], who have shown that it is W[1]-complete. The problem was also studied by Cai [39] who gave the currently best exact algorithm for it, and by Marx, who gave an FPT approximation scheme[1] [196].

In our reductions we sometimes also use some other graph problems, which do not have such a clear relation to the covering problems.

[1]The definition of an FPT approximation scheme is similar to the definition of a polynomial time approximation, with the difference that instead of requiring a polynomial time complexity of the algorithms we require them to run in FPT time. Thus, in case of a maximization problem $\mathcal{P}$, an FPT approximation scheme provides for every $\epsilon > 0$ a $(1 - \epsilon)$-approximation algorithm for $\mathcal{P}$, running in FPT time. FPT approximation schemes for minimization problems are defined analogously.

Definition 2.26 (Clique)**.** *In the* Clique *problem we are given an undirected graph $G = (V, E)$ and we ask whether there exists a subset $S \subseteq V$ of K vertices, such that all the vertices from S are connected in G.*

Definition 2.27 (Densest-k-Subgraph)**.** *In a* Densest-k-Subgraph *problem we are given an undirected graph $G = (V, E)$ and a positive integer K. We ask for a subgraph S with K vertices with the maximal number of edges.*

Densest-k-Subgraph is a generalization of the Clique problem, from which it follows that it is NP-hard. Furthermore, it seems that Densest-k-Subgraph is quite hard to approximate. Khot [164] ruled out the existence of a PTAS for the problem under standard complexity-theoretic assumptions, Bhaskara et al. [30] showed the polynomial integrality gap, Raghavendra and Steurer [249] and Alon et al. [9] proved that there is no polynomial-time approximation algorithm with a constant approximation ratio, but under somewhat non-standard assumptions. Finally, the best approximation algorithm for the problem that we know of, due to Bhaskara et al. [29], has approximation ratio $O(n^{1/4+\epsilon})$, where n is the number of vertices in the input graph.

Another variant of the Clique problem is the Maximum Edge Biclique Problem (MEBP) problem defined below. Here, however, the connection to the Clique problem is a bit less explicit and for the appropriate reduction we refer the reader to the paper of Petters [239].

Definition 2.28 (MEBP)**.** *In the* Maximum Edge Biclique Problem *(*MEBP*) we are given a balanced bipartite graph $(U \cup V, E)$ where $U \cup V$ is the set of vertices ($\|U\| = \|V\|$) and E is the set of edges (there are edges only between the vertices from U and V). We ask for a biclique (i.e., a subgraph S, such that every vertex from $U \cap S$ is connected with every vertex from $V \cap S$) with as many edges as possible.*

According to Feige and Kogan [99], there exists a constant c such that there is no polynomial $(2^{c\sqrt{\lg n}}/n)$-approximation algorithm for MEBP unless for some ϵ we have 3-SAT $\in$ DTIME$(2^{n^{3/4+\epsilon}})$. Currently it seems unlikely that such an algorithm for 3-SAT exists. For some of our arguments it will be more convenient to define and use the following variant of MEBP.

Definition 2.29 (MEBP-V)**.** *In* MEBP-V *we are given the same input as in* MEBP *and a positive integer K. We ask for a biclique S such that $\|S \cap V\| = K$ and S contains as many edges as possible.*

Lemma 2.2. *There exists a constant c such that there is no polynomial-time $(2^{c\sqrt{\lg n}}/n)$-approximation algorithm for* MEBP-V *unless for some ϵ we have* 3-SAT $\in$ DTIME$(2^{n^{3/4+\epsilon}})$.

Proof. For the sake of contradiction, let us assume that there exists a constant c and a polynomial-time $(2^{c\sqrt{\lg n}}/n)$-approximation algorithm $\mathcal{A}$ for MEBP-V. By running $\mathcal{A}$ for every value of K ranging from 1 to $\|V\|$, we obtain a polynomial-time $(2^{c\sqrt{\lg n}}/n)$-approximation algorithm for MEBP. This stays in contradiction with the result of Feige and Kogan [99]. $\square$

2.3.3 Partition and Packing Problems

In a few cases, for the sake of convenience, we also use reductions from some other problems.

Definition 2.30 (SubsetSum)**.** *In the* SubsetSum *problem we are given a set $S = \{x_1, x_2, \ldots, x_n\}$ of n integers and a value x, and we ask whether there exists a subset $S' \subseteq S$ such that $\sum_{x_i \in S'} x_i = x$.*

The SubsetSum problem is NP-hard, but it is often considered as one of the easiest NP-hard problems. If the values of the elements in the set S are bounded by a constant C, a simple dynamic program solves the problem in time $O(nC)$. Further, SubsetSum admits a simple FPTAS [216].

Definition 2.31 (BinPacking)**.** *In the* BinPacking *problem we are given a set $T = \{t_1, t_2, \ldots, t_q\}$ of q items and their sizes (the size of item t_i is denoted s_i), and a set N of n bins, each having capacity d. We ask whether it is possible to pack all the items into the bins.*

Definition 2.32 (UnaryBinPacking)**.** *This problem is identical to* BinPacking *except for the fact that all input parameters are encoded in unary.*

UnaryBinPacking is W[1]-hard [147] when parameterized by n, the number of bins. We use this result to show parameterized hardness of some problems for small values of the numeric parameters. W[1]-hardness of UnaryBinPacking is slightly alleviated by the result of Jansen et al. [147], who showed an additive 1-approximation algorithm running in FPT time even for BinPacking (thus, to pack the items this algorithm uses at most one more bin than the optimal algorithm). Interestingly, in terms of multiplicative approximation there is no PTAS for the problem under standard complexity assumptions.

Part I

Finding a Collective Set of Items: From Proportional Multirepresentation to Group Recommendation

Chapter 3

The Model and Its Applications

3.1 Overview

In this part of the dissertation we analyze the problem of selecting a "good" set of items that can be collectively used by a group of agents. We analyze the computational complexity of the problem, and we show effective algorithms for its different variants. Throughout this part we use the term "item" in its most general meaning, so that items may be movies, goods, candidates in political elections, etc. Under this interpretation, "selecting a set of items" may correspond to many real-life scenarios, ranging from selecting a set of movies for an airplane, through deciding which journals a university library should subscribe, to selecting a group of people (e.g., a parliament) to represent a society. The agents have their preferences over the items and, intuitively, the set of items is considered "good" if it results in a high satisfaction of the agents. We explore several natural ways of measuring the quality of a set of items and several natural ways in which the items can be shared among the agents. We will briefly describe our approaches in the further part of this overview.

A number of real-world problems consist of selecting a set of items for a group of agents to jointly use. Among many natural problems that our approach addresses, we can find the following ones.

1. *Selecting a set of activities.* Consider a conference where the organizers want to set up a number of activities for the participants, for a free afternoon. The organizers face the problem of selecting a set of good items: here the agents are the conference participants and the items correspond to the activities that the organizers consider for selection. Consequently, by saying that an agent i uses an item a we mean that i attends the activity a. Clearly, the conference participants can have diversified preferences and satisfying them all might not be possible. If the conference organizers decide to select a single activity, it is likely that many participants will be unhappy and will choose to stay in their hotel rooms. To avoid such a situation, the organizers may select more, say, $K \geq 2$ activities and allow the participants to choose the preferred one. This way it is likely

that the number of unhappy participants will decrease. However, for obvious reasons (e.g., the cost or logistical inconvenience), the organizers cannot decide to select too many activities; after the careful consideration they might decide on some particular value of K, the number of selected activities. Nevertheless, some choices of K activities are better than the others. The organizers would look for the best choice that would satisfy the participants most.

2. *Selecting a set of (sport/language) classes* for students. This example is similar in nature to the previous one. Here, the agents are the students and the items correspond to the (sport/language) classes. We say that a student i uses an item a if she is assigned to the class a. The university wants to select the set of K sport (or language) classes for the students. The selected classes are, then, available for the students' registration. However, in comparison with the previous example, this one exposes an additional issue. Here it is natural to assume that the classes have strict capacity restrictions: each item (each class) can fit only up to some maximum number of students (agents). Consequently, it may happen that some students will not be able to register for their most preferred classes. Thus, in addition to selecting a good set of classes (the classes most liked by the students), the university would look for a good assignment of the students to the selected classes.

3. *Selecting a set of movies for a plane.* In this example an airline wants to select a set of movies to provide for the passengers on the plane's entertainment system. Naturally, the airline would select such movies that would satisfy the passengers most. Thus, in this case, the agents are the plane passengers and the items correspond to the movies considered for selection. Consequently, using an item a corresponds to watching a movie a. Since every passenger is allowed to watch any available movie, in this example, in contrast to the previous one, the items have no capacities. This example is, however, different from the first one too: It is unnatural to assume that each passenger would see a single movie only (would use a single item). It is more likely that the passengers would derive their satisfaction based on various subsets of available movies.

4. *Finding a proportional representation* for a group of people. Let us consider elections in which we want to select a set of representatives (e.g., a parliament, or some other collective body) for a given society. At first sight, this example might look totally different from the previous ones. However, after a careful consideration we will see that selecting a good parliament is nothing different from selecting a good set of items. The agents are the voters and the items correspond to the candidates participating in the election. In this example, however, it is less clear how the preferences of the voters over the sets of candidates (i.e., over the possible parliaments) can look like. For instance, even though the voters can have their precise preferences over single

candidates, it is unclear how they can be extended to the preferences over the committees (parliaments). Two fully proportional representation systems, the Chamberlin–Courant system [47] and the Monroe system [210], address this issue and present the following appealing interpretation. If we want a parliament to proportionally represent the society, then for each voter there should be a member of the parliament that represents this voter well. When viewed from this perspective, we can say that an agent i (a voter i) uses an item a if she is represented in the collective body by a. This idea was first highlighted by Chamberlin and Courant [47] who proposed a novel election system in which the satisfaction of the voter is defined as the satisfaction from her best representative in the parliament. In this election system we pick a set of candidates that satisfies the voters most. Monroe suggested a similar idea, but in addition he required that each member of parliament represents roughly the same number of voters [210]. Here, we just accent the intuition behind the idea of viewing the elections systems as resource allocation and we present a more detailed discussion in Chapter 5.

These, of course, are just several characteristic examples, and our model, formally defined in the next section, captures fundamental computational challenges from many other real-life problems. The above examples are all similar in nature, but there are some substantial differences in ways in which items are shared between agents. We can distinguish the two following basic approaches:

The *disjunctive* approach. In this approach every agent is allowed to use only a single item from the selected set. This approach is natural, e.g., in the activity selection problem, where the activities take place at the same time. Consequently, each agent needs to choose which activity from the selected ones he or she wants to attend. If there are no further restrictions, then, naturally, every agent chooses her most preferred item. A special case of the *disjunctive* approach, the *capacitated disjunctive* approach, additionally assumes that every item has its capacity, i.e., the maximum number of agents that can comfortably use this item. This approach is natural, e.g., in the problem of selecting a set of (sport/language) classes, which usually have strict capacity restrictions.

The *conjunctive* approach. This approach is on the other extreme of the spectrum of possibilities. Here we assume that each agent uses *all* the selected items and that he derives equal parts of satisfactions from using each one of them. Thus, e.g., if the airline wants to select $K = 7$ movies, then in the conjunctive approach every agent would be interested in having *all* seven selected movies compatible with her preferences. For example, the agent's satisfaction from the set of items could be the sum of this agent's satisfactions from all the selected individual items. This approach is usually adequate if the number of selected items (e.g., the number of movies to be selected for the plane) is very small. Indeed, in the movie selection example it is hard to expect the passengers to get satisfaction from all the movies if there is, say, $K = 100$ of them available. They would not even have the possibility to see all of them during one flight. On the other hand, the conjunctive approach is natural if the selected items are in some way independent and do not exclude each other. For instance, if we select the set of K activities for a class of students, each taking place in a different month, then each student will be able to participate (and derive her utility) from all the selected activities.

The two basic approaches described above are on the extreme ends of a spectrum of possibilities. In many real-life examples one should expect much more complicated schemes in which the agents use (and get their satisfaction from) subsets of the selected items. For instance, in the movie selection example, a passenger watches her top preferred movie certainly, the second one probably, the third one perhaps, etc. Thus, we should expect that the total satisfaction of such a passenger would be mostly influenced by her satisfaction from the top preferred movie, less influenced by her satisfaction from the second preferred movie, even less by the third one, and so on. Similarly, if we consider the problem of selecting K journals for the university's library, then a reader will typically not be interested in having only a single favorite journal, nor in having all journals compatible with her preferences, but rather she will be most interested in having access to some T interesting journals. Even in the parliament, the voters might want to be represented by some top T members of the parliament rather than by a single person.

The above observation motivates us to propose a new approach. In this new approach the impact of each selected item on the satisfaction of an agent may depend on the *rank* of this item (from the agent's point of view) among the selected ones. Thus, in the movie selection example, the impact of the top preferred movie on the satisfaction of a passenger is greater than that of the second preferred movie, and so on. If we decided to remove the passenger's top preferred movie from the plane's entertainment system, then the impact of the second movie would increase (the passenger is more likely to watch the second movie, now that her favorite one is not available). Formally, this new approach is defined by a vector of weights, and thus we call it a *weighted approach*. The first weight quantifies the impact of the top preferred item from the selected ones on the agent's total satisfaction. The second weight quantifies the impact of the second preferred item, and so on. We give the precise formal definition of the weighted approach in the following section.

In addition to considering different ways in which the items can be used by the agents (the disjunctive, conjunctive, and the weighted approach), we also explore several ways of measuring the satisfaction of the agents from the selected items. There are two main ways in which the agents can express their preferences regarding the items (their satisfactions regarding the items): either the agents can express numerical utility values for the items, or they can rank the items from the most desirable one to the least desirable one. Intuitively, the numerical utility values quantify the levels of satisfaction of the agents from using particular items.

The preference rankings (also called the preference orders) carry less information than the numerical utilities. However, in many contexts we can only hope for the rankings, e.g., because it is too difficult for the agents to derive their exact numerical values for the utilities. In cases where we have only agents' preference rankings, but we need explicit numerical values that quantify their satisfactions from the items, we can use a positional scoring function, a function that for a given preference order allows one to derive numerical utilities.

Intuitively, a positional scoring function (a PSF) assigns a certain value of utility to an item ranked at a certain position. That is, the utility of an item depends solely on its position in the agent's preference order. There are several particularly popular positional scoring rules. For instance, the Borda count is a PSF which assumes that the utility associated with an item depends linearly on the position of that item in the ranking (with the least preferred item assigned the utility equal to 0): If there are m items in total, then the item ranked as i-th best has utility equal to $(m - i)$. Another example of a popular positional scoring function is k-approval, in which the first k items in the preference order get utility equal to 1, and all the remaining ones get the utility equal to 0. We note that the Borda count PSF and the k-approval PSF can be seen as two extremes. In the case of k-approval, the agents only have extreme views regarding the items (they like them or not). In contrast, in the case of the Borda count, they have a full linear spectrum of appreciation of the items.

Further, it is reasonable to assume that the formula for computing the satisfaction of a single agent from the group of items depends on the way in which the items are used by the agents. For instance, in the uncapacitated disjunctive approach, we naturally assume that the utility of an agent from the selected set of items is just her utility from the most preferred item in the selected set. In the capacitated disjunctive approach this should be the utility of the item that the agent is assigned to, and in the conjunctive approach, the sum of the agent's utilities over all the selected items.

As we already noted, we very often deal with more complex schemes than the disjunctive and the conjunctive one. For such cases we introduced the weighted approach. On one hand, the weighted approach can be viewed as a technique to describe how the items are used by the agents. On the other hand, it can be viewed as a formula for computing the utility of the agents from the sets of items. From the previous discussion we recall that the weighted approach is defined by the vector of weights. The i-th weight quantifies the impact of the utility of the i-th most preferred item in the set on the total utility of an agent. Thus, the utility of an agent from the set of items is just an ordered weighted average [309] of her utilities from the individual items in the set. Naturally, the weights used to compute the ordered weighted average of the agent's utilities are the same weights that we use to describe the weighted approach. Consequently, we will refer to the vector of weights that defines the weighted approach as the ordered weighted average vector (the OWA vector, sometimes also referred to as the OWA operator). We note that the weighted approach is a generalization of both the conjunctive and the disjunctive approach. Indeed, if the OWA vector has all weights equal, then the utility of each item has the same impact on the agent's total utility and such an OWA vector defines the conjunctive approach. If the OWA has only the first weight greater than 0, then only the most preferred item of a given agent affects her total utility. Consequently, such an OWA vector defines the disjunctive approach.

Finally, there are various ways to aggregate the satisfaction of the whole groups of agents, as well. For instance, we might want to maximize the sum of the utilities of the agents (the utilitarian approach) or to maximize the utility of the least satisfied agent (the egalitarian approach), or to use some yet other approach.

Apart from providing a formal specification of the new weighted model and providing concrete examples and settings where particular OWA operators are applicable, our goal in this part of the dissertation is to establish the computational complexity of the problem of selecting an optimal set of items. In our computational analysis we consider different ways in which an optimal set of items is defined. These different ways correspond to:

1. Different classes of the agents' utilities (e.g., explicitly given utility values, values derived through positional scoring rules, with the focus on Borda count and k-approval).

2. Different approaches to sharing the items (corresponding to different OWA operators).

3. Two different approaches to aggregating the agents' utilities: the egalitarian and the utilitarian one[1].

Our results show that in almost all cases (with the single natural exception of the conjunctive utilitarian approach), the considered computational problem is NP-hard. Nevertheless, we show many approaches that alleviate these hardness results. In particular, we consider high-quality polynomial-time and exponential-time approximation algorithms (for example, for some of our problems we show polynomial-time and FPT approximation schemes). For some cases we experimentally confirm the high quality of solutions found by our algorithms.

We believe that using approximation algorithms is justified for the considered applications. For example, if we want to select a set of movies played at the same time in a cinema, it is likely that an agent will enjoy watching a good movie even though is is not her absolutely most favorite one. Consider another similar example: Amazon.com may recommend you a book on gardening which may not be the best

[1]The egalitarian approach is usually much harder computationally. In Chapter 5 we show that all the considered egalitarian versions of the disjunctive variant of the problem are inapproximable. In further chapters we focus only on the utilitarian approach.

book for you on this topic, but still full of useful advice. For such situations, Herbert Simon [272] used the term 'satisficing,' instead of optimizing, to explain the behavior of decision makers under circumstances in which an optimal solution cannot be easily determined. On page 129 he wrote: "Evidently, organisms adapt well enough to *satisfice*; they do not, in general, 'optimize'." Effectively, what Simon says is that the use of approximation algorithms fits well with the human nature.

Below, we briefly summarize the remaining content of this part of the dissertation.

In the further part of this chapter we formalize our model. We specify several natural ways in which the items can be shared among the group of agents and several ways to measure the satisfaction of the agents. We formalize the computational problem of selecting a set of items. Next, we present our computational results in the three following chapters.

In Chapter 4 we focus on the weighted conjunctive approach. We show that the problem, and its (almost) every reasonable special case is NP-hard. In many contexts, however, we do not require perfect optimal solutions and losing only a small fraction of optimality is an affordable price. Motivated by this observation, we focus on the approximation algorithms for the problem. Unfortunately, we show that our problem in its full generality is hard to approximate. Next, we consider some more specific classes of OWA vectors, as well as some specific types of agents' utilities. The main message of our computational results in this chapter is that although our problem in general is hard, it has different approximation properties depending on the class of OWA vectors used and the nature of agents' utilities.

In Chapter 5, we consider the disjunctive and the capacitated disjunctive approaches to sharing the items.[2] Most of our results are given for the variant in which we get the utilities of the agents by applying the Borda count to their preference rankings. In this approach, the problem of finding the optimal selection of the items is both NP-hard [188,245], and hard from the perspective of the parameterized complexity theory [27]. These hardness results hold for every reasonable positional scoring function, both for the utilitarian and the egalitarian variant. Similarly as in Chapter 4, here we also study the approximability of this variant of the problem.

Apart from the several new hardness results, we find that the utilitarian version of the problem can be approximated with very high quality. Our algorithms give even better results when evaluated on real data describing peoples' preferences. We show that our algorithms preserve their high quality even if we have incomplete data in the form of preference orders truncated to the certain number of top positions.

In Chapter 6 we further explore the utilitarian version of the problem of selecting the set of items with the disjunctive approach to sharing the items, but for another measure of agents' satisfaction. Here we consider the approval utilities of the agents, i.e., the utilities coming from the set $\{0, 1\}$. One possible way to obtain these kind of utilities is by applying the k-approval PSF to the agents' preference rankings.

[2]For the capacitated disjunctive approach we analyze the specific case in which the capacities of the items are equal.

We observe that the disjunctive utilitarian version of the problem of selecting the set of items with the approval utilities is equivalent to the MaxCover problem. In MaxCover we are given a set N of elements and a set of subsets of N, with the goal of selecting K subsets to cover as many elements from N as possible. Indeed, we can see the equivalence of the problems by identifying the agents with the elements, the subsets with the items, and the relation "approves of an item" with "belongs to a subset". With this identification, selecting a set of items to maximize the number of satisfied agents (the agents that approve at least one item in the selected set) is equivalent to selecting a set of subsets to maximize the number of covered elements.

The MaxCover problem already received a lot of attention from the scientific community. It is known that there exists a polynomial $(1 - 1/e)$-approximation algorithm for the problem [136], and that, under standard complexity assumptions, there exists no polynomial algorithm with the better approximation guarantees [98]. Motivated by this known bound, we ask if there exist good exponential-time approximation algorithms. Indeed, we show that there exists a whole spectrum of the exponential approximation algorithms, with a trade-off between the computational complexity and the approximation ratio. These algorithms have better running time than the exact brute-force algorithm for MaxCover. At the same time, they achieve better approximation guarantees than $(1 - 1/e)$. We also show FPT approximation schemes for the variants of the MaxCover problem in which each element can belong to at most p sets (which corresponds to the requirement that each agent approves of at most p items). Naturally, all these results apply to the considered variant of the problem of selecting a set of items.

3.2 The Model

In this section we formally define our model and the problem of selecting a collective set of items. We first define basic notions such as items, utilities and preference orders. Then, we present the item selection problem in its most general variant. Finally, we discuss several interesting special cases of our problem.

3.2.1 Agents, Alternatives, and Utilities

We assume that there is a set $N = [n]$ of *agents* and a set $A = \{a_1, \dots a_m\}$ of *alternatives* (also referred to as *items*, or *candidates*).[3] For each agent $i \in N$ and for each alternative $a_j \in A$, we have an intrinsic utility (satisfaction) u_{i,a_j} that the agent i derives from a_j. A collection of the utility vectors of all the agents is called a *utility profile*.

[3]We will use the terms "alternative" and "item" interchangeably. We will more often us the term alternative in the context of the disjunctive approach, and we will use the term item in the general weighted approach. We will use the term "candidate" to refer to alternative in the context of elections.

Instead of providing numerical utilities, sometimes the agents express their preferences as *preference orders* (also referred to as *rankings*). A *preference order* $\succ$ is a strict linear order over A, i.e., a linear order of the form $a_{\pi(1)} \succ a_{\pi(2)} \succ \cdots \succ a_{\pi(m)}$ for some permutation π of $[m]$. From now on, we will write $\succ_i$ to denote the i-th agent's preference order. A collection $V = (\succ_1, \dots, \succ_n)$ of agents' preference orders is called a *preference profile.* For an alternative $a \in A$, by $\mathrm{pos}_i(a)$ we mean the position of a in the i'th agent's preference order. For example, if a is the most preferred alternative for i then $\mathrm{pos}_i(a) = 1$, and if a is the least preferred one then $\mathrm{pos}_i(a) = m$. Sometimes we will include subsets of the alternatives in the descriptions of preference orders. For example, if A is the set of alternatives and B is some nonempty strict subset of A, then by $B \succ A - B$ we mean that for the preference order $\succ$ all alternatives in B are preferred to those outside of B.

Most of our techniques require the agents to provide numerical utility values. One possible way of extracting numerical utilities from preference orders is to apply a positional scoring function to the preference profile. A *positional scoring function* (PSF) is a function $\alpha^m \colon [m] \to \mathbb{N}$. Intuitively, a PSF assigns to an alternative ranked in the position i, the utility value equal to $\alpha^m(i)$. Thus, the utility of an agent i from an alternative a solely depends on the position of a in the i's preference ranking: $u_{i,a_j} = \alpha^m(\mathrm{pos}_i(a))$.

Since we use positional scoring rules to derive the utilities of the agents, it is natural to consider non-increasing positional scoring functions. A PSF α^m is an *non-increasing positional scoring function* if for each $i, j \in [m]$, if $i < j$ then $\alpha^m(i) \geq \alpha^m(j)$[4].

Typically, we are interested in families of non-increasing positional scoring functions, $(\alpha^m)_{m=1}^{\infty}$, with one function for each possible number of candidates. In particular, we will be interested in the Borda count PSF family $\alpha^m_{\mathrm{B}}(i) = m - i$, and in the k-approval PSF family $\alpha^m_{\mathrm{A}}(i) = 1$ if $i \leq k$, and $\alpha^m_{\mathrm{A}}(i) = 0$ otherwise. We assume that our positional scoring functions are computable in polynomial time with respect to m.

3.2.2 Item-Selection Problem

In this section we formulate the problem of selecting a collective set of items in its most general form. In this general form we assume that the utility that each agent

[4]At this point we note that non-decreasing positional scoring functions have their applications as well. Instead of measuring the satisfactions of the agents from the items, we can measure the level of their unhappiness, their dissatisfaction. This way we can use an alternative optimization goal and instead of trying to maximize the agents' satisfaction, we can aim at minimizing their unhappiness. A non-decreasing positional scoring functions can be used to extract from the agents' preference orders concrete numerical values quantifying their dissatisfactions with the items. However, for the sake of consistency, we do not consider minimization of the dissatisfaction in this dissertation. For a discussion on approximating dissatisfaction of the agents in this context we refer a reader to our conference paper [277].

derives from a set of K items is an ordered weighted average [309] of this agent's intrinsic utilities for these items.[5]

A weighted ordered average (OWA) over K numbers is a function defined through a vector $\alpha^{(K)} = \langle \alpha_1, \ldots, \alpha_K \rangle$ of K (nonnegative) numbers[6] as follows: Let $\vec{x} = \langle x_1, \ldots, x_K \rangle$ be a vector of K numbers and let $\vec{x}^{\downarrow} = \langle x_1^{\downarrow}, \ldots, x_K^{\downarrow} \rangle$ be the nonincreasing rearrangement of $\vec{x}$, that is, $x_i^{\downarrow} = x_{\sigma(i)}$, where σ is any permutation of $\{1, \ldots, K\}$ such that $x_{\sigma(1)} \geq x_{\sigma(2)} \geq \ldots \geq x_{\sigma(K)}$. Then we set:

$$\mathrm{OWA}_{\alpha^{(K)}}(\vec{x}) = \sum_{i=1}^{K} \alpha_i x_i^{\downarrow}$$

To make the notation lighter, we write $\alpha^{(K)}(x_1, \ldots, x_K)$, instead of $\mathrm{OWA}_{\alpha^{(K)}}(x_1, \ldots, x_K)$.

We will provide a more detailed discussion of OWA operators useful in our context later; for the time being let us note that they can be used, for example, to express the arithmetic average (through the size-K vector $(\frac{1}{K}, \ldots, \frac{1}{K})$), the maximum and minimum operators (through vectors $(1, 0, \ldots, 0)$, and $(0, \ldots, 0, 1)$, respectively) and the median operator (through the vector that has 0s everywhere, except for the middle position, where it has 1).

Given the above setup, we formalize our problem of computing "the most satisfying set of K items" in the following way.

Definition 3.1. *In the* OWA-Winner *problem we are given a set $N = [n]$ of agents with utilities over m items (alternatives) from the set $A = \{a_1, \ldots, a_m\}$, a positive integer K ($K \leq m$), and a K-number OWA $\alpha^{(K)}$. The task is to compute a subset $W = \{w_1, \ldots, w_K\}$ of A such that $u_{\mathrm{ut}}^{\alpha^{(K)}}(W) = \sum_{i=1}^{n} \alpha^{(K)}(u_{i,w_1}, \ldots, u_{i,w_K})$ is maximal.*[7]

For a family $(\alpha^{(K)})_{K=1}^{\infty}$ of OWAs, we write α-OWA-Winner to denote the variant of the OWA-Winner problem where, for a given solution size K, we use OWA $\alpha^{(K)}$. From now on we will not mention the size of the OWA vector explicitly and it will always be clear from context. We implicitly assume that OWAs in our families are polynomial-time computable.

Finally, we will often speak of variants of OWA-Winner where agents' utilities are somehow restricted. In particular, by approval-based utilities we mean that each

[5] We note that this general form corresponds to the weighted approach to sharing items between agents, introduced and described in the previous section.

[6] The standard definition of OWAs assumes normalization, that is, $\sum_{i=1}^{K} \alpha_i = 1$. We do not make this assumption here for the sake of convenience; note that whether OWA vectors are normalized or not is irrelevant to all notions and results of this work.

[7] Formally, what we define here should be called the utilitarian OWA-Winner problem because we are interested in maximizing the total utility. It is also natural to consider egalitarian OWA-Winner problem, where we maximize the utility of the worst-off agent. However, the discussion on the egalitarian versions of our problem in this dissertation is limited to Chapter 5.

agent's utilities come from the set $\{0,1\}$, and by Borda-based utilities we mean the case where for each agent i the set of her utilities for all the items, that is, $\{u_{i,a_1}, \ldots, u_{i,a_m}\}$ is equal to $\{0, \ldots, m-1\}$.

Example 3.1. Let $n = 6$, $m = 6$, $K = 3$, $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, $\alpha = (2,1,0)$, and Borda-based utilities derived from the following rankings:

$$\begin{aligned} \texttt{3 agents:}\ & a_1 \succ a_2 \succ a_3 \succ a_5 \succ a_6 \succ a_4 \\ \texttt{2 agents:}\ & a_6 \succ a_1 \succ a_4 \succ a_3 \succ a_5 \succ a_2 \\ \texttt{1 agent:}\ & a_5 \succ a_4 \succ a_2 \succ a_3 \succ a_6 \succ a_1 \end{aligned}$$

Let us compute the score of $\{a_1, a_2, a_6\}$. The first three agents get utility $2 \cdot 5 + 4 = 14$ each, the next two get $2 \cdot 5 + 4 = 14$ each and the last one gets $2 \cdot 3 + 1 = 7$. Therefore the score of $\{a_1, a_2, a_6\}$ is $3 \cdot 14 + 2 \cdot 14 + 7 = 42 + 28 + 7 = 77$. It can be checked that this is the optimal set. (The next best ones are $\{a_1, a_2, a_4\}$, $\{a_1, a_2, a_5\}$ and $\{a_1, a_5, a_6\}$, all with score 75.) On the other hand, the rule defined by OWA $\alpha' = (1,1,1)$ would choose $\{a_1, a_2, a_3\}$, and the Chamberlin and Courant's rule (in our terms, the rule defined by OWA $\alpha'' = (1,0,0)$) would choose $\{a_1, a_5, a_6\}$.

3.2.3 Special Cases of the Problem

OWA-Winner is a remarkably general problem and we will usually focus on some special cases, for particular families of OWAs. For instance, for OWA $\langle 1, 0, \ldots, 0 \rangle$ we obtain the disjunctive version of the problem, in which the total utility of an agent is her utility from the most preferred selected item. This version of the problem has particularly many applications, and—thus—is of special interest: we study this variant in Chapters 5 and 6.

Below we give a catalog of other particularly useful OWA families (in the description below we take K to be the dimension of the vectors to which we apply a given OWA).

1. ***k*-median OWA.** For each $k \in \{1, \ldots, K\}$, $k\text{-med}^{(K)}$ is the OWA defined by the vector of $k-1$ zeros, followed by a single one, followed by $K-k$ zeros. It is easy to see that $k\text{-med}^{(K)}(x_1, \ldots, x_K)$ is the k-th largest number in the set $\{x_1, \ldots, x_K\}$ and is known as the k-median of $\vec{x}$. In particular, $1\text{-med}^{(K)}(\vec{x})$ is the maximum operator, $K\text{-med}^{(K)}(\vec{x})$ is the minimum operator, and if K is odd, $\frac{K+1}{2}\text{-med}^{(K)}(\vec{x})$ is the median operator.

2. ***k*-best OWA.** For each $k \in \{1, \ldots, K\}$, $k\text{-best}^{(K)}$ OWA is defined through the vector of k ones followed by $K-k$ zeros. That is, $k\text{-best}^{(K)}(\vec{x})$ is the sum of the top k values in $\vec{x}$ (with appropriate scaling, this means an arithmetic average of the top k numbers). $K\text{-best}_K^{(K)}$ is simply the sum of all the numbers in $\vec{x}$ (after scaling, the arithmetic average), and so it corresponds to the conjunctive

version of the problem. Naturally, 1-best$^{(K)}$ OWA corresponds to the disjunctive version of the problem.

3. **Arithmetic progression OWA.** These OWAs are defined through vectors of the form aprog[a]$^{(K)} = \langle a + (K-1)b, a + (K-2)b, \dots, a\rangle$, where $a \geq 0$ and $b > 0$. (One can easily check that the choice of b has no impact on the outcome of OWA-Winner; this is not the case for a, though.)

4. **Geometric progression OWA.** These OWAs are defined through vectors of the form gprog$[p]^{(K)} = \langle p^{K-1}, p^{K-2}, \dots, 1\rangle$, where $p > 1$.

5. **Hurwicz OWA.** For each λ, $0 \leq \lambda \leq 1$, this OWA is defined through vector $(\lambda, 0, \dots, 0, 1-\lambda)$.

Naturally, all sorts of middle-ground OWAs are possible between these particular cases, and can be tailored for specific applications. As our natural assumption is that highly ranked items have more impact than lower-ranked objects, we often make the assumption that OWA vectors are *nonincreasing*, that is, $\alpha_1 \geq \dots \geq \alpha_K$. While most OWA operators we consider are indeed nonincreasing, this is not the case for k-medians (except for 1-median) and Hurwicz (except for $\lambda = 1$).

3.3 Applications of the Model

In this section we give several different scenarios where our model is applicable. In contrast to the informal examples presented in the overview, here we give the examples of concrete values of the parameters (e.g., OWA vectors) for concrete applications.

3.3.1 The Disjunctive Approach

The disjunctive variant of OWA-Winner has notably many applications. In particular, the disjunctive variant of the OWA-Winner problem with the Borda utilities corresponds to the Chamberlin–Courant's voting rule for electing sets of representatives [47]. For this rule, voters (agents) have Borda utilities over a set of candidates and we wish to elect a K-member committee (for instance, a parliament), such that each voter is represented by one member of the committee. In other words, if we select K candidates, then a voter is "represented" by that selected candidate that she ranks highest among the chosen ones. The idea is that then, in the parliament, each selected candidate would have voting power proportional to the number of voters that she represents. It is easy to see that winner determination under Chamberlin–Courant's voting rule corresponds exactly to solving 1-best-OWA-Winner for the case of Borda utilities.

The Monroe's system [210], while not precisely a specific case of the OWA-Winner problem, is certainly related—it corresponds to the disjunctive capacitated version of the problem of selecting a collective set of items, where the capacities of the items are roughly equal to n/K (with n denoting the number of agents and K the size of the elected committee). Since in the Monore's system each committee member represents roughly the same number of voters, the committee itself does not need to run weighted voting when making decisions. Similarly as in the Chamberlin–Courant's system, in the Monroe's system we assume the Borda utilities of the agents.

More generally, the 1-best-OWA-Winner problem (with and without capacities) addresses the broad and important problem of finding a proportional representation for a group of agents.

The above connections show that, indeed, the complexity of the disjunctive variant of the OWA-Winner problem is interesting, can lead to progress in several other directions, and may have impact on other applications of artificial intelligence and computer science in general.

3.3.2 The Weighted Approach

Here we present three scenarios, with the common feature that they each focus on some form of uncertainty about the final outcome; the impact of a selected item is the probability that this item will be actually used by the agent.

Malfunctioning Items or Unavailable Candidates

In this model, we assume that, as in the disjunctive approach, each user only benefits from one item, but that the items may not be working properly: if we select (off-line) a set of items S, then (on-line) there will be a subset S^+ of items that can be used, and a set $S^- = S \setminus S^+$ of objects that are 'malfunctioning' or are 'unavailable' and cannot be used. For instance, items are radio channels that can be unreachable, or items are candidates running in an election and these candidates may finally decide to not take a position in the elected committee, or items are parking lots that are to be built but that can sometimes be full (see [187] for further examples of social choice with possibly unavailable candidates). Moreover, we have a prior probability distribution about the (un)availability of items: as in [187], we assume that each item is available with probability p (i.i.d.). The utility an agent gets from a set of selected items S is the value of the best available object in S, that is, of the best object in S^+. Therefore, it is the value of the item ranked in position i in S if the first $i-1$ items are unavailable and the ith item is available. The expected contribution of an item to the utility of a user is therefore proportional to $p(1-p)^{i-1}$, which leads to the OWA defined by $\alpha_i = p(1-p)^{i-1}$, which is a geometric progression with initial value p and coefficient $1-p$.

Uncertainty About the Number of Items Enjoyed by a User

We assume now that there is some uncertainty about the number of items that a user will enjoy. A first possible reason is that users may have a limited capacity to enjoy items. For instance, items are movies or books and each user has a time constraint that will prevent him or her from enjoying all selected items. A second possible reason is that users are reluctant to use items they don't like enough: they will watch only the films whose value reaches a given subjective threshold. We give here two possible models for the choice of the OWA vectors:

- We first assume that the probability that a user enjoys i items, for $0 \leq i \leq K$, is uniformly distributed, that is, a user will enjoy exactly her first i items in S with probability $P(i) = \frac{1}{K+1}$. Thus, she will enjoy the item ranked i if she enjoys at least i items, which occurs with probability $\frac{K-i+1}{K+1}$. This leads to the OWA vector defined by $\alpha_i = K - i + 1$ (we disregard the normalizing constant), which is an arithmetic progression.

- Second, we assume that the values given by each user to each item are distributed uniformly, i.i.d., on $[0, 1]$ and that each user uses only the items that have a value at least θ, where θ is a fixed (user-independent) threshold. Therefore, a user enjoys the item in S ranked in position i if he or she values at least i items at least θ, which occurs with probability $\sum_{j=i}^{K} \left(\begin{array}{c} K \\ i \end{array} \right) (1-\theta)^i \theta^{K-i}$, thus leading to the following OWA vector defined by $\alpha_i = \sum_{j=i}^{K} \left(\begin{array}{c} K \\ i \end{array} \right) (1 - \theta)^i \theta^{K-i}$. For instance, if $K = 4$ and $\theta = \frac{3}{4}$, the OWA (omitting the denominators) is $\alpha = (175, 67, 13, 1)$; for $K = 4$ and $\theta = \frac{1}{2}$, we get $\alpha = (14, 12, 5, 1)$; and for $K = 4$ and $\theta = \frac{1}{4}$, we get $\alpha = (252, 243, 189, 81)$.

Ignorance About Which Item Will Be Assigned to a User

We now assume that a matching mechanism will be used posterior to the selection of the K items. The matching mechanism used is not specified; it may also be a randomized mechanism.

If users have a *complete ignorance about the mechanism used*, then it makes sense to use known criteria for decision under complete uncertainty (see, e.g., the book of Luce and Raiffa [190]):

- the *Wald* criterion assumes that agents are extremely risk-averse, and corresponds to $\alpha = K\text{-med}^{(K)}$: we, therefore, seek to maximize $\sum_{i=1}^{n} \min_{w_j \in W} u_{i,w_j}$.

- the *Hurwicz* criterion is a linear combination between the worst and the best outcomes, and corresponds to $\alpha = (\lambda, 0, \ldots, 0, 1 - \lambda)$ for some fixed $\lambda \in (0, 1)$.

If users still have a complete ignorance about the mechanism used *except that they know that they are guaranteed to get one of their best i items*, then the Wald and Hurwicz criteria now lead, respectively, to the OWAs $\alpha = i\text{-med}^{(K)}$ and $\alpha = (\lambda, 0, \ldots, 0, 1-\lambda, 0, \ldots, 0)$, with $1-\lambda$ in position i.

If users know that the mechanism used is a random mechanism with a uniform distribution among the items ranked in positions 1 to i, then the choice of i-best OWA makes sense. More generally, the matching mechanism may assign items to agents with a probability that depends on the rank and that decreases when the rank increases.

3.3.3 The Conjunctive Approach

In the conjunctive approach each agent derives an equal part of her satisfaction from every selected item, and, thus, this variant corresponds to the K-best-OWA-Winner problem. The conjunctive approach also finds its application in several settings. For instance, in the context of parliamentary elections, if the utilities of the agents are derived by applying the Borda count PSF to the agents' preference rankings, the K-best-OWA-Winner problem corresponds to winner determination in the K-Borda rule.[8]

This variant is also applicable if the selected items are in some sense independent, and thus if the fact whether an agent is going to use a selected item does not depend on what other items have been selected (e.g., if we want to buy, for a group of agents, the tickets for 10 movies, each played in a different month).

This conjunctive variant of the problem of selecting a collective set of items is, however, computationally easy, and so its analysis in this dissertation is limited.

3.4 Related Work

Weighing intrinsic values by coefficients that are a function of their rank in a list is of course not new. Ordered Weighted Average operators have been used extensively in multicriteria decision making (MCDM), and to a lesser extent in social choice [151]. Also, studying *rank-dependent expected utility* (RDEU) [248] is a well-known research stream in decision theory, whose starting point is the construction of models that explain Allais' paradox: given a set of possible consequences of an act, the contribution of a possible consequence on the agent's RDEU is a function of its probability and of its rank in the list of consequences ordered by decreasing probability. While these three research streams use ranks to modify the contribution of a *criterion*, an *agent*, or a *possible consequence*, in our setting they modify the

[8]In the K-Borda rule we select as the winners the K items (candidates) with the highest Borda scores. The Borda score of an item (candidate) is the sum of the utilities assigned to this item by all agents, assuming that these utilities are obtained by applying the Borda count PSF to agents' preference rankings.

contribution of *items*, our final aim being to select an optimal set of items. Since we do not select criteria, agents or possible consequences, it is not obvious how our results can apply to these three aforementioned research fields.

Several known settings are recovered as particular cases of our general model. In particular, this applies to Chamberlin–Courant's [47] and Monroe's [210] election systems, and to (variants of) the budgeted social choice model [188]. Computational complexity of Chamberlin–Courant's and Monroe's schemes was first studied by Procaccia et al. [245]; the parameterized complexity of the problem was analyzed by Betzler et al. [27]. The first approximation algorithm was proposed by Lu and Boutilier [188]. These results on approximability were extended in this chapter and in Chapter 5.

It is quite natural to view finding proportional representation (and thus solving 1-best-OWA-Winner) as coalition structure generation problems in cooperative games [251–253]. Here the agents are the players of a certain cooperative game, and the value of each "coalition" of agents is the (dis)satisfaction they derive from being assigned to a given item. The goal is to partition the set of the voters into K disjoint coalitions (and assign distinct candidates to them) in a way that maximizes the sum of coalition values (or, in the egalitarian setting, the value of the worst off coalition). Formally, this is a very special case of coalition structure generation with externalities. Externalities come from the fact that no two coalitions can be assigned to the same candidate.

Group recommender systems (see, e.g., the work of O'Connor et al. [227] for one of the first approaches, and the surveys of Jameson and Smyth [146] and of Masthoff [198]) aim at recommending sets or sequences of items (such as a set of television programs or a sequence of songs) to a group of users, based on preferences of all group members. Two mainstream approaches exist (see [146]). The first one is based on the construction of an 'average user' whose preferences are built by aggregating the preferences of individuals in the group; next, the group of items that are most preferred by this average user is recommended. The second approach is based merging the recommendations made for individuals; in other words, the recommendation for the set of agents is the set consisting of all recommendations for the individuals. Unlike these, in our approach the decision of selecting items is performed for the whole sets of items simultaneously. This makes the problem more computationally intense, but allows to find better recommendations.

The facility location problem (FL) is closely related to 1-best-OWA-Winner. In FL, however, the goal is to minimize the dissatisfaction of the agents instead of maximizing their utility (satisfaction). Although, as far as exact solutions are concerned both formulations are equivalent, there is a significant difference in the quality of approximation. Some works focus on general dissatisfaction functions [101], but most of the results were established for dissatisfactions corresponding to the distances, and thus satisfying the triangle inequality [145,269]. Also, in FL the goal is to minimize the dissatisfaction of the worst-off agent (the egalitarian view). The utilitarian version of the problem is called K-MEDIAN [145]. The parameterized complexity of the problem was analyzed by Fellows and Fernau [101]. The approximation algorithms include [57,145,269]. Interestingly, a local-search algorithm (which, to the best of our knowledge, is the best known approximation algorithm for the capacitated version of FL [57]) is also a $\frac{1}{2}$-approximation algorithm for maximizing nondecreasing submodular functions [220], and thus for OWA-WINNER with non-decreasing utility functions. We conclude that it would be interesting to compare the algorithms for FL and K-MEDIAN with different algorithms for OWA-WINNER on real preference traces [200].

Chapter 4

Finding a Collective Set of Items: The Weighted Approach

In this chapter we study the problem of selecting a collective set of items, formally defined in Chapter 3. We consider the problem both in its full generality, and for various natural classes of the OWA vectors, and several types of agents' utilities.

We start our analysis by discussing worst-case results in Section 4.1; then we move on to approximability results, in Section 4.2 for the case of general utilities (but with some focus on approval-based ones) and in Section 4.3 for the case of Borda-based ones. We show that, in general, our problem is NP-hard, but that for the natural class of non-increasing OWA vectors the problem can be approximated with the ratio $(1 - 1/e)$; for other classes of OWA vectors good approximation algorithms are rare. Nevertheless, for the case of Borda-based utilities it is possible to obtain polynomial-time approximation schemes (PTASes) for a relatively large, interesting family of OWA vectors, including k-median and k-best (for a fixed value of k), and geometric progression OWA.

4.1 Computing Exact Solutions

In general, OWA-Winner is a rather difficult problem. However, as long as we seek a size-K winner set where K is a fixed constant, then the problem is in P.

Proposition 4.1. *For each fixed constant K (the size of the winner set),* OWA-Winner *is in* P.

Proof. For a profile with m items, there are only $O(m^K)$ sets of winners to try. We try them all and pick one that yields highest utility. □

Naturally, in practice the variant of the problem with fixed K has only limited applicability and throughout the rest of the chapter we focus on the α-OWA-Winner variant of the problem where K is given as part of the input and α represents a family of OWAs, one for each value of K. By results of Procaccia, Rosenschein and Zohar [245] and Lu and Boutilier [188], we know that the 1-best-OWA-WINNER problem is NP-hard both for approval and for Borda-based utilities (see also Chapters 5 and 6 for a detailed discussion). A simple reduction shows that this result carries over to each family of k-best OWAs and k-med OWAs, where k is a fixed positive integer, independent of the input.

Proposition 4.2. *For each fixed k,* k-best-OWA-WINNER *and* k-med-OWA-WINNER *are* NP*-complete, even if the utility profiles are restricted to be approval-based or Borda-based.*

Proof. Let k be a fixed constant. It is easy to see that k-best-OWA-WINNER and k-med-OWA-WINNER are both in NP. To show NP-hardness, we give reductions from 1-best-OWA-WINNER (either with approval-based utilities or with Borda-based utilities) to k-best-OWA-WINNER and to k-med-OWA-WINNER (with the same types of utilities).

Let I be an instance of 1-best-OWA-WINNER with n agents, m items, and where we seek a winner set of size K. We form an instance I' of k-best-OWA-WINNER that is identical to I except that: (1) We add $k-1$ special items $b_1, \dots, b_{k-1}$ such that under approval-based utilities each agent i has utility 1 for each item b_j, $1 \leq j \leq k-1$, and under Borda-based utilities, for item b_j, $1 \leq j \leq k-1$, each agent i has utility $m+j-1$. (2) We set the size of the desired winner set to be $K' = K + k - 1$. It is easy to see that if there is an optimal solution W' for I' that achieves some utility x, then there is a solution W'' for I' that uses all the $k-1$ items $b_1, \dots, b_{k-1}$ and also achieves utility x. Further, the set $W'' - \{b_1, \dots, b_{k-1}\}$ is an optimal solution for I and, for I, has utility $x - \sum_{i=1}^{n}\sum_{j=1}^{k-1} u_{i,b_j} = x - n\sum_{j=1}^{k-1} u_{1,b_j}$.

Analogous argument shows that 1-best-OWA-WINNER reduces to k-med-OWA-WINNER (also for approval-based and for Borda-based utilities). □

On the other hand, it is easy to note that for K-best OWA (that is, for the family of constant OWAs $\alpha = (1, \dots, 1)$) the problem is easy.

Proposition 4.3. K-best-OWA-WINNER *is in* P.

Proof. Let I be an input instance with m items and n agents, where we seek a winner set of size K. It suffices to compute for each item the total utility that all the agents would derive if this item were included in the winner set and return K items for which this value is highest. □

Indeed, if the agents' utilities are either approval-based or Borda-based, K-best-OWA-WINNER boils down to (polynomial-time) winner determination for K-best approval rule and for K-Borda rule [79], respectively (see also the work of Elkind et al. [96] for a general discussion of multiwinner rules). Given this result, it is quite interesting that already $(K-1)$-best-OWA-WINNER is NP-hard, both for approval-based and for Borda-based utilities.

Theorem 4.4. $(K-1)$-best-OWA-Winner *is* NP*-complete even for approval-based utilities.*

Proof. Membership in NP is clear. We show a reduction from the VertexCover problem. Let I be an instance of VertexCover with graph $G = (V, E)$, where $V = \{v_1, \ldots, v_m\}$ and $E = \{e_1, \ldots, e_n\}$, and with a positive integer K (without loss of generality, we assume that $K \geq 3$ and $K < m$). In I we ask if there is a set C of up to K vertices such that each edge is incident to at least one vertex from C.

We construct an instance I' of $(K-1)$-best-OWA-Winner in the following way. We let the set of items be $A = V$ and we form $2n$ agents, two for each edge. Specifically, if e_i is an edge connecting two vertices, call them $v_{i,1}$ and $v_{i,2}$, then we introduce two agents, e_i^1 and e_i^2, with the following utilities: e_i^1 has utility 1 for $v_{i,1}$ and for $v_{i,2}$, and has utility 0 for all the other items; e_i^2 has opposite utilities—it has utility 0 for $v_{i,1}$ and for $v_{i,2}$, and has utility 1 for all the remaining ones.

Let W be some set of K items (i.e., vertices) and consider the sum of the utilities derived by the two agents e_i^1 and e_i^2 from W under $(K-1)$-best-OWA. If neither $v_{i,1}$ nor $v_{i,2}$ belong to W, then the total utility of e_i^1 and e_i^2 is equal to $K-1$ (the former agent gets utility 0 and the latter one gets $K-1$). If only one of the items, i.e., either $v_{i,1}$ or $v_{i,2}$, belongs to W, then the total utility of e_i^1 and e_i^2 is equal to K (the former agent gets utility 1 and the latter one still gets $K-1$). Finally, if both items $v_{i,1}, v_{i,2}$ belong to W, then the total utility of e_i^1 and e_i^2 is also equal to K (the former gets utility 2 and the latter gets utility $K-2$). Thus the total utility of all agents is equal to $K \cdot n$ if and only if the answer to the instance I is "yes". This shows that the reduction is correct and, since the reduction is computable in polynomial time, the proof is complete. □

A variant of this result for Borda-based utilities follows by an application of a similar idea, but the restriction to Borda-based utilities requires a much more technical proof.

Theorem 4.5. $(K-1)$-best-OWA-Winner *is* NP*-hard even for Borda-based utilities.*

Proof. As before, it is clear that the problem is in NP, and we only show NP-hardness. We give a reduction from VertexCover. Let I be an instance of the VertexCover problem that consists of undirected graph $G = (V, E)$, where $V = \{v_1, \ldots, v_m\}$ and $E = \{e_1, \ldots, e_n\}$, and positive integer K (without loss of generality, we assume that $K \geq 3$).

From I, we construct an instance I' of $(K-1)$-best-OWA-Winner with Borda-based utilities as follows. We set

$$x = 4n(m+2)(K+4)$$

and we let the set of items be $A = V \cup \{d_1, d_2\} \cup H$, where $H = \{h_1, \ldots, h_x\}$ and $\{d_1, d_2\}$ are sets of dummy items that we need to build appropriate structure of the utility profile. To build the set of agents N, we set

$$y = (n(x+m+2)^2 + 1)$$

and we set $N = N_E \cup N_1 \cup \cdots \cup N_y$, where $N_E = \{e_1^1, e_1^2, \ldots, e_n^1, e_n^2\}$ contains pairs of agents that correspond to the edges of G, and $N_1, \ldots, N_y$ contain pairs of agents needed for the construction. Specifically, every set N_i, $1 \leq i \leq y$, consists of two agents, f_i^1 and f_i^2. We refer to the agents in the set $N_1 \cup \cdots \cup N_y$ as the "dummy agents." We describe agents' utilities through their preference orders (their utilities are derived using the Borda PSF).

The agents in the set N_E have the following preference orders. Let $e_i \in E$ be an edge of the graph that connects vertices $v_{i,1}$ and $v_{i,2}$. Agents e_i^1 and e_i^2 have preference orders:

$$e_i^1 : d_1 \succ d_2 \succ V - \{v_{i,1}, v_{i,2}\} \succ H \succ \{v_{i,1}, v_{i,2}\},$$
$$e_i^2 : d_1 \succ d_2 \succ \{v_{i,1}, v_{i,2}\} \succ H \succ V - \{v_{i,1}, v_{i,2}\}.$$

(Recall that when we put a set of items in a preference order, this means that this set can be replaced by these items in an arbitrary, easily computable, way.) Each agent f_i^1, $1 \leq i \leq y$, has the same, fixed, preference order:

$$f_i^1 : d_1 \succ v_1 \succ v_2 \cdots \succ v_m \succ d_2 \succ h_1 \cdots \succ h_x.$$

Similarly, each agent f_i^2, $1 \leq i \leq y$, has preference order:

$$f_i^2 : d_2 \succ v_m \succ v_{m-1} \cdots \succ v_1 \succ d_1 \succ h_1 \cdots \succ h_x.$$

Finally, in the instance I' we seek a set of winners of size $K+2$. This means that we use $(K+1)$-best-OWA to compute the aggregated utility that an agent derives from a set of winners.

This concludes the description of the reduction and it is clear that it is polynomial-time computable. Before we prove that it is correct, let us make several observations. Let W be some optimal solution for I'. We claim that W does not contain any of the items from H. For the sake of contradiction, assume that some $h \in H$ belongs to W. Since d_1 and d_2 are ranked ahead of h in every preference order (and in some preference orders d_1 is first and d_2 is second, so their utility cannot be ignored by the $(K+1)$-best-OWA), we infer that d_1 and d_2 must belong to W as well (otherwise we would obtain higher utility by replacing h with one of d_1 and d_2 in W). Let v be some item from V that does not belong to W. If we replace h with v in W then the total utility of the dummy agents increases by at least $2y$. Why is this so? Consider some pair N_i, $1 \leq i \leq y$ of dummy agents. Item h is either the lowest ranked member of W for both f_i^1 and f_i^2 or for neither. We consider these cases:

- ***h* is the lowest-ranked winner for both the agents in *N_i*.** Replacing h with v means that either some other member h' of $H \cap W$ becomes the lowest ranked winner for both f_i^1 and f_i^2, or d_2 becomes the lowest ranked winner for f_i^1 and d_1 becomes the lowest ranked winner for f_i^2. In either case, both f_i^1 and f_i^2 obtain utility higher by at least one from v than from the item that became the new lowest-ranked winner. Thus, the total utility yielded by these two agents increases by at least two.

- ***h* is not the lowest-ranked winner for either agent in N_i.** In this case, since both agents rank v higher than h and replacing h with v does not change the lowest-ranked winner for either of the agents, their total utility also increases at least by two.

Since there are y pairs of agents, the total utility increases by at least $2y$. Since the total utility of the agents from N_E is lower than $2n(x+m+2)^2 < 2y$, we see that after the change the total utility of all the agents increases. Thus, we get a contradiction and we conclude that W does not contain any of the agents from H.

Next, we claim that both d_1 and d_2 belong to W. We give a detailed argument for d_1 only; the case of d_2 is analogous. For the sake of contradiction, assume that d_1 does not belong to W. Let v_k be an item from W such that for each v_j, $j < k$, v_j does not belong to W. By our assumptions, for each agent f_i^2, $1 \leq i \leq y$, v_k is the lowest-ranked winner from W. Thus, if we replace v_k with d_1 in W, then the utility of each agent f_i^2 will not change, whereas the utility of each agent f_i^1 will increase. Further, the utility of each agent from N_E will increase. Thus, by replacing v_k with d_1, we can increase the total utility of the agents. We reach a contradiction and we conclude that d_1 must have been a member of W. An analogous argument shows that d_2 belongs to W as well.

As the result of the above reasoning, we infer that each set of winners consists of d_1, d_2, and K items from V. Whenever both d_1 and d_2 are included in the set of winners and neither item from H is, the total utility of the dummy agents is the same, irrespective which items from V are selected. With these observations, we now show that the answer for the input VertexCover instance is “yes” if and only if there is a size-$(K+2)$ winner set for I' that for agents in the set N_E yields total utility at least $nx(K+4)$.

($\Rightarrow$) Let us assume that there exists a cover C for I, that is, a set C of K vertices such that each edge is incident to at least one vertex from C. We show that winner set $W = C \cup \{d_1, d_2\}$ gives total utility of every two agents e_i^1 and e_i^2, $1 \leq i \leq n$, equal to at least $x(K+4)$. Pick some arbitrary i, $1 \leq i \leq n$, and let $v_{i,1}$ and $v_{i,2}$ be the two vertices connected by edge e_i. If both $v_{i,1}$ and $v_{i,2}$ belong to C, then e_i^2 obtains utility at least x for each item in $\{v_{i,1}, v_{i,2}, d_1, d_2\}$ (at least utility $4x$ in total). On the other hand, e_i^1 obtains utility at least x for each item in $W - \{v_{i,1}, v_{i,2}\}$. This gives utility at least Kx. Altogether, both agents get utility at least $x(K+4)$. If only one of the items $v_{i,1}$ and $v_{i,2}$, say $v_{i,1}$, belongs to C, then e_i^2 obtains utility at least $3x$ (at least x for every item from $\{v_{i,1}, d_1, d_2\}$), and e_i^1 obtains utility at least $(K+1)x$ (at least $2x$ from items d_1 and d_2, and at least $(K-1)x$ from the $K-1$ members of C

that e_i^1 ranks on the top positions). Again, both agents get utility at least $x(K+4)$. Thus the total utility of the agents in N_E in the optimal solution must be at least $nx(K+4)$.

($\Leftarrow$) Assume that W is some optimal solution for I' and that for the agents in N_E it yields utility at least $nx(K+4)$. By previous discussion, we know that W contains d_1, d_2, and K members of V. We set $C = W \setminus \{d_1, d_2\}$. Let us fix some arbitrary i, $1 \leq i \leq n$. Let $v_{i,1}$ and $v_{i,2}$ be the two vertices connected by edge e_i. We observe that under W, the total utility of agents e_i^1 and e_i^2 is at most $(x+m+2)(K+4)+mK$. To see this, let z be the number of items from $\{v_{i,1}, v_{i,2}\}$ that are included in C and note that (1) for the upper bound we can disregard the OWA that we use, (2) there are $x+m+2$ items and so we can upper-bound the utility derived from each item by $x+m+2$, (3) altogether, the items from W are ranked on at most $K+2-z$ top-$(m+2)$ positions by e_i^1 (we upper-bound their total utility by $(K+2-z)(x+m+2)$) and at most $2+z$ top-$(m+2)$ positions by e_i^2 (we upper-bound their total utility by $(2+z)(x+m+2)$), and (4) the items from W are ranked on at most z bottom-m positions by e_i^1 (we upper-bound their total utility by zm) and on $K-z$ bottom-m positions by e_i^2 (we upper-bound their total utility by $(K-z)m$). When we sum up these upper bounds, we get $(x+m+2)(K+4)+mK$. However, for our argument we also need an upper bound on the total utility of e_i^1 and e_i^2 under the assumption that neither $v_{i,1}$ nor $v_{i,2}$ belongs to C. In this case, the upper bound is $(x+m+2)(K+3)+mK$. We obtain it in the same way as the previous bound, except that we note that due to our $(K+1)$-best-OWA, the utility derived by e_i^1 can take into account at most $K+1$ agents from the top-$(m+2)$ positions of the preference order of e_i^1.

Based on these upper bounds, we will now show that if the total utility derived from W by the agents in N_E is at least $nx(K+4)$, then C must correspond to a cover of all the edges of G. To this end, consider a situation where there is at least one edge e_i such that neither of the vertices that it connects belongs to C. By using our upper bounds, in this case the total utility of the agents from N_E can be at most:

$$\begin{aligned}
&(K+3)(x+m+2) + (n-1)(K+4)(x+m+2) + nmK \\
&= (x+m+2)(K+3+(n-1)(K+4)) + nmK \\
&= (x+m+2)(n(K+4)-1) + nmK \\
&= xn(K+4) + n(m+2)(K+4) - (x+m+2) + nmK \\
&= xn(K+4) + 0.25x - (x+m+2) + nmK \\
&< xn(K+4)
\end{aligned}$$

(The last two lines follow directly from the definition of x.) So, from the assumption that C is not a solution for I, we obtain that the total utility of the agents in N_E must be lower than $nx(K+4)$, which contradicts our assumption. Thus C is a correct solution for I and, so, I is a yes-instance of VertexCover. This completes the proof. □

Using a proof that combines the ideas of the proofs of Theorems 4.2 and 4.4, we show that indeed OWA-WINNER is NP-hard for a large class of natural OWAs. This time, for the sake of simplicity, we give a proof for the approval-based utilities only.

Theorem 4.6. *Fix an OWA family α, such that there exists p such that for every K we have $\alpha_p^{(K)} > \alpha_{p+1}^{(K)}$; α-OWA-WINNER is* NP*-hard for approval-based utilities.*

Proof. We give a reduction from the CUBICVERTEXCOVER problem. Let I be an instance of CUBICVERTEXCOVER with graph $G = (V, E)$, where $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$, and positive integer K. Each vertex in G has degree exactly equal to three. W.l.o.g., we assume that $n > 3$. We ask if there is a set C of up to K vertices such that each edge is incident to at least one vertex from C.

We construct an instance I' of α-OWA-WINNER. In I' we set $N = E$ (the agents correspond to the edges), $A = V \cup \{b_1, b_2, \dots b_{p-1}\}$ (there are $(p-1)$ dummy items; other items correspond to the vertices), and we seek a collection of items of size $K + p - 1$. Each agent e_i, $e_i \in E$, has utility 1 exactly for all the dummy items and for two vertices that e_i connects and for each of the dummy items (for the remaining items e_i has utility 0). In effect, each agent has utility 1 for exactly $p + 1$ items.

We claim that I is a yes-instance of CUBICVERTEXCOVER if and only if there exists a solution for I' with the total utility at least $n\sum_{i=1}^{p}\alpha_i + (3K - n)\alpha_{p+1}$.

($\Rightarrow$) If there is a vertex cover C of size K for G, then by selecting the items $W = C \cup \{b_1, b_2, \dots b_{p-1}\}$ we obtain the required utility of the agents. Indeed, for every agent e_i there are at least p items in W for which i gives value 1 (the $p - 1$ dummy items and at least one vertex incident to e_i). These items contribute the value $n\sum_{i=1}^{p}\alpha_i$ to the total agents' utility. Additionally, since every non-dummy item has value 1 for exactly 3 agents, and since every agent has at most $(p+1)$ items with value 1, there are exactly $(3K-n)$ agents that have exactly $(p+1)$ items in W with values 1. Thus, these $(3K - n)$ agents (thanks these items that they rank at the $(p + 1)$'th position) additionally contribute $(3K - n)\alpha_{p+1}$ to the total utility. Altogether, the agents' utility is $n\sum_{i=1}^{p}\alpha_i + (3K - n)\alpha_{p+1}$, as claimed.

($\Leftarrow$) Let us assume that there is a set of $(K + p - 1)$ items with total utility at least $n\sum_{i=1}^{p}\alpha_i + (3K - n)\alpha_{p+1}$. In I' we have $(p - 1)$ items that have value 1 for each of the n agents, and every other item has value 1 for exactly 3 agents. Thus the sum of the utilities of $K + p - 1$ items (without applying the OWA operator yet) is at most $(p-1)n + 3K = pn + (3K - n)$. In effect, the total utility of the agents (now applying the OWA operator) is $n\sum_{i=1}^{p}\alpha_i + (3K - n)\alpha_{p+1}$ only if for each agent e_i the solution contains p items with utility 1. Since there are only $p - 1$ dummy items, it means that for each agent e_i there is a vertex v_j in the solution such that e_j is incident to v_j. That is, I is a yes-instance of CUBICVERTEXCOVER. □

The above theorem applies directly, for example, to the families of geometric progression OWAs and arithmetic progression OWAs.

Corollary 4.7. *The problems* gprog[p]-OWA-Winner *(for each $p > 1$) and* aprog[a]-OWA-Winner *(for each $a > 0$) are NP-complete.*

In fact, the following theorem (whose proof builds upon the above constructions) shows an even stronger NP-hardness result.

Theorem 4.8. *Fix an OWA family α, such that for every K, $\alpha^{(K)}$ is nonincreasing and nonconstant; α-*OWA-Winner *is* NP*-hard for approval-based utilities.*

Proof sketch. We give a reduction from CubicVertexCover. Let I be an instance of CubicVertexCover with graph $G = (V, E)$, where $V = \{v_1, \ldots, v_m\}$ and $E = \{e_1, \ldots, e_n\}$, and with positive integer K.

Now let us consider $\alpha^{(2K)}$; since $\alpha^{(2K)}$ is nonincreasing and nonconstant, one of the two following conditions must hold.

1. There exists $p \leq K$ such that $\alpha_p^{(2K)} > \alpha_{p+1}^{(2K)}$.

2. There exists $p > K$ such that $\alpha_p^{(2K)} > \alpha_{p+1}^{(2K)}$, and for every $p \leq K$, we have $\alpha_p^{(2K)} = \alpha_{p+1}^{(2K)}$.

If (1) is the case then we use a reduction similar to that in the proof of Theorem 4.6. The only difference is that apart from the set D_1 of $(p-1)$ dummy items (ranked first by all agents), we introduce the set D_2 of $(2K-p+1)$ dummy items and $(2K-p+1)$ sets $N_1, N_2, \ldots, N_{2K-p+1}$, each consisting of $2n$ dummy agents. The dummy items from D_2 are introduced only to fill-up the solution up to $2K$ members. The dummy agents from N_i have utility 1 for each of the items from D_1 and for the i'th item from D_2 (they have utility 0 for all the other items). This is to enforce that the items from D_2 are selected in the optimal solution. The further part of the reduction is as in the proof of Theorem 4.6.

If (2) is the case, then we use a reduction similar to that in the proof of Theorem 4.4. We let the set of items be $A = V \cup D_1 \cup D_2$, where D_1, $|D_1| = p+1-K$, and D_2, $|D_2| = 2K-p-1$ are sets of dummy items that we need for our construction. Similarly as in the proof of Theorem 4.4, for each edge $e_i \in E$ we introduce two agents e_i^1 and e_i^2. Here, however, we additionally need the set F of $(2n+1)$ dummy agents. Each dummy agent from F assigns utility 1 to each dummy item from D_2 and utility 0 to the remaining items—consequently, since $|F| > 2n$, each dummy item from D_2 must be selected to every optimal solution. Further, each non-dummy agent assigns utility 1 to each dummy agent from D_1—this way we ensure that every item from D_1 must be selected to every optimal solution. Finally, the utilities of the non-dummy agents for the non-dummy items are defined exactly as in the proof of Theorem 4.4. This ensures that the optimal solution, apart from D_1 and D_2, will contain the non-dummy items that correspond to the vertices from the optimal vertex cover. □

By the above discussion, we conjecture that the family of constant OWAs, that is, the family of K-best OWAs, is the only natural family for which α-OWA-Winner is in P. We leave this conjecture as a natural follow-up question.

Even though almost all the practical variants of OWA-Winner are computationally hard, we still might be in a position where it is necessary to obtain an exact solution for a given OWA-Winner instance and the brute-force algorithm from Proposition 4.1 is too slow. In such a case, it might be possible to use an integer linear programming (ILP) formulation of the problem, given below. We believe that this ILP formulation is interesting in its own right and, in particular, that it is interesting future work to experimentally assess the size of instances for which it yields solutions in reasonable amount of time.

Theorem 4.9. OWA-Winner *reduces to computing a solution for the following integer linear program.*

$$\begin{aligned}
&\text{minimize } \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k} \\
&\text{subject to:} \\
&\text{(a)} : \sum_{i=1}^{m} x_i = K \\
&\text{(b)} : x_{i,j,k} \le x_j && , i \in [n]; j,k \in [K] \\
&\text{(c)} : \sum_{j=1}^{m} x_{i,j,k} = 1 && , i \in [n]; k \in [K] \\
&\text{(d)} : \sum_{k=1}^{K} x_{i,j,k} = 1 && , i \in [n]; j \in [m] \\
&\text{(e)} : \sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \ge \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)} && , i \in [n]; k \in [K-1] \\
&\text{(f)} : x_{i,j,k} \in \{0,1\} && , i \in [n]; j,k \in [K] \\
&\text{(g)} : x_j \in \{0,1\} && , j \in [m]
\end{aligned}$$

Proof. Consider an input instance with n agents $N = [n]$ and m items $A = \{a_1, \ldots, a_m\}$, where we seek a winner set of size K, under OWA $\alpha = (\alpha_1, \ldots, \alpha_K)$. For each $i \in N$, $a_j \in A$, we write u_{i,a_j} to denote the utility that agent i derives from item a_j.

We form an instance of ILP with the following variables: (1) For each $i \in N$, $j \in [m]$, and $k \in [K]$, there is an indicator variable $x_{i,j,k}$ (intuitively, we interpret $x_{i,j,k} = 1$ to mean that for agent i, item a_j is the k-th most preferred one among those selected for the solution). (2) For each $j \in [m]$, there is an indicator variable x_j (intuitively, we interpret $x_j = 1$ to mean that a_j is included in the solution). Given these variables (and assuming that we enforce their intuitive meaning), the goal of our ILP is to maximize the function $\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K}\alpha_k u_{i,a_j} x_{i,j,k}$.

We require that our variables are indeed indicator variables and, thus, take values from the set $\{0,1\}$ only (constraints (f) and (g)). We require that the variables of the form $x_{i,j,k}$ are internally consistent. (constraint (c) says that each agent ranks only one of the candidates from the solution as k-th best, constraint (d) say that there is no agent i and item a_j such that i views a_j as ranked on two different positions among the items from the solution.) Then, we require that variables of the form $x_{i,j,k}$ are consistent with those of the form x_j (constraint (b)) and that exactly K items are selected for the solution (constraint (a)).

Our final constraint, constraint (e), requires that variables $x_{i,j,k}$ sort the items from the solution in the order of descending utility values (separately for each agent, of course). We mention that constraint (e) is necessary only for the case of OWAs α that are not-nonincreasing. For a nonincreasing α, an optimal solution for our ILP already ensures the correct "sorting" (otherwise our goal function would not be maximized). $\square$

4.2 (In)Approximability Results: General Utilities and Approval Utilities

The OWA-Winner problem is particularly well-suited for applications that involve recommendation systems (see, e.g., the work of Lu and Boutilier [188] for a discussion of 1-best-OWA-Winner in this context and examples in Chapter 3). For recommendation systems it often suffices to find good approximate solutions instead of perfect, exact ones, especially if we only have estimates of agents' utilities. It turns out that the quality of the approximate solutions that we can produce for OWA-Winner very strongly depends on both the properties of the particular family of OWAs used and on the nature of agents' utilities.

First, we show that as long as our OWA is nonincreasing, a simple greedy algorithm achieves $\left(1-\frac{1}{e}\right)$ approximation ratio. This result follows by showing that for a nonincreasing OWA α, the function u_{ut}^{α} (recall Definition 3.1) is submodular and nondecreasing, and by applying the famous result of Nemhauser et al. [220] (we recall the definition of submodularity and the theorem of Nemhauser et al. in Section 2.2.2).

Lemma 4.10. *Let I be an instance of* OWA-Winner *with a nonincreasing OWA α. The function u_{ut}^{α} is submodular and nondecreasing.*

Proof. Let I be an instance of OWA-Winner with agent set $N=[n]$, item set $A=\{a_1,\dots,a_m\}$, desired solution size K, and OWA $\alpha=\langle\alpha_1,\dots,\alpha_K\rangle$. For each agent $i\in N$ and each item $a_j\in A$, u_{i,a_j} is a nonnegative utility that i derives from a_j.

```
$W \leftarrow \emptyset$ ;
for $\ell \leftarrow 1$ to $K$ do
    $score \leftarrow \{\}$ ;
    foreach $a \in A \setminus W$ do
        $s \leftarrow 0$;
        foreach $i \in N$ do
            $p \leftarrow \|\{w \in W : u_{i,w} \leq u_{i,a}\| + 1$ ;
            $s \leftarrow s + u_{i,a} \cdot \alpha_p$ ;
        $score[a] \leftarrow s$
    $W \leftarrow W \cup \{\mathrm{argmax}_{a \in A \setminus W} score[a]\}$;
```

Figure 4.1: The greedy algorithm (Greedy) for finding the utilitarian set of K winners.

Since all the utilities and all the entries of the OWA vector are nonnegative, we note that u_{ut}^{α} is nondecreasing. To show submodularity, we decompose u_{ut}^{α} as follows:

$$u_{\mathrm{ut}}^{\alpha}(W) = \sum_{\ell=1}^{K-1} (\alpha_\ell - \alpha_{\ell+1}) u_{\mathrm{ut}}^{\ell\text{-best-OWA}}(W) + \alpha_K u_{\mathrm{ut}}^{K\text{-best-OWA}}(W)$$

For each $W \subseteq A$, $i \in N$ and $\ell \in [m]$, let $\mathrm{Top}(W, i, \ell)$ be the set of those ℓ items from W whose utility, from the point of view of agent i, is highest (we break ties in an arbitrary way). Since nonnegative linear combinations of submodular functions are submodular, it suffices to prove that for each $i \in N$ and each $\ell \in [m]$, function $u_i^{\ell}(W) = \sum_{w \in \mathrm{Top}(W,i,\ell)} u_{i,w}$ is submodular.

To show submodularity of u_i^{ℓ}, consider two sets, W and W', $W \subseteq W' \subseteq A$, and some $a \in A \setminus W'$. We claim that:

$$u_i^{\ell}(W \cup \{a\}) - u_i^{\ell}(W) \geq u_i^{\ell}(W' \cup \{a\}) - u_i^{\ell}(W'). \tag{4.1}$$

Let u_W and $u_{W'}$ denote the utilities that the i-th agent has for the ℓ-th best items from W and W', respectively (or 0 if a given set has fewer than ℓ elements). Of course, $u_{W'} \geq u_W$. Let u_a denote i-th agent's utility for a. We consider two cases. If $u_a \leq u_W$, then both sides of (4.1) have value 0. Otherwise:

$$u_i^{\ell}(W' \cup \{a\}) - u_i^{\ell}(W') = \max(u_a - u_{W'}, 0)$$
$$u_i^{\ell}(W \cup \{a\}) - u_i^{\ell}(W) = u_a - u_W,$$

which proves (4.1) and completes the proof. □

Theorem 4.11. *For a nonincreasing OWA α, the algorithm* Greedy *from Figure 4.1 is a polynomial time $(1 - 1/e)$-approximation algorithm for the problem of finding the utilitarian set of K winners.*

Proof. The thesis follows from the results of Nemhauser et al. [220] on approximating nondecreasing submodular functions (The Nemhauser's theorem is recalled as Theorem 2.1 in Section 2.2.2). □

Is a $(1 - \frac{1}{e})$-approximation algorithm a good result? After all, $1 - \frac{1}{e} \approx 0.63$ and so the algorithm guarantees only about 63% of the maximum possible satisfaction for the agents. Irrespective if one views it as sufficient or not, this is the best possible approximation ratio of a polynomial-time algorithm for (unrestricted) OWA-WINNER with a nonincreasing OWA. The reason is that 1-best-OWA-Winner with approval-based utilities is, in essence, another name for the MAXCOVER problem, and if P $\neq$ NP, then $(1 - \frac{1}{e})$ is the approximation upper bound for MAXCOVER [98]. We omit the exact details of the connection between MAXCOVER and 1-best-OWA-Winner and instead we point the readers to Chapter 6, which discusses this issue in detail.

For OWAs that are not nonincreasing, it seems that we cannot even hope for such a $(1 - \frac{1}{e})$-approximation algorithm. Such OWAs yield utility functions that are not necessarily submodular. For example, this is the case for 2-med OWA.

Example 4.12. Let us consider a single agent, two sets of items $W = \{c, d\}$ and $W' = \{b, c, d\}$ (of course $W \subset W'$), and 2-med-OWA α. The utilities of the agent over the items a, b, c, and d are equal to 10, 9, 2, and 1, respectively. We get:

$$u^{\alpha}_{\mathrm{ut}}(W \cup \{a\}) - u^{\alpha}_{\mathrm{ut}}(W) = 2 - 1 = 1, \quad u^{\alpha}_{\mathrm{ut}}(W' \cup \{a\}) - u^{\alpha}_{\mathrm{ut}}(W') = 9 - 2 = 7.$$

That is, u^{α}_{ut} is not submodular. Indeed, this example works even for approval-based utilities: it suffices to set the utilities for a and b to be 1, and for c and d to be 0.

It is quite plausible that there are no constant-factor approximation algorithms for many not-nonincreasing OWAs. As an example, let us consider the case of families of OWAs whose first entries are zero (but that, nonetheless, have a nonzero entry at a sufficiently early position). If there were a good approximation algorithm for winner determination under such OWAs, then there would be a good approximation algorithm for the DENSEST-K-SUBGRAPH problem, which seems unlikely (for the definition and the discussion on the DENSEST-K-SUBGRAPH problem we refer the reader to Section 2.3).

Theorem 4.13. *Fix some integer ℓ, $\ell \geq 2$. Let α be a family of OWAs such that each OWA in the family (for at least ℓ numbers) has 0s on positions 1 through $\ell - 1$, and has a nonzero value on the ℓ'th position. If there is a polynomial-time $x(n)$-approximation algorithm for α-*OWA-WINNER *then there is a polynomial-time $x(n)$-approximation algorithm for the* DENSEST-K-SUBGRAPH *problem.*

We should mention that Theorem 4.13 holds for a somewhat more general class of OWAs than stated explicitly. The proof relies on the fact that the first entry of the OWA is zero and that after the first non-zero entry of the OWA there are still

$K-1$ positions, where K is the parameter from the input of the DENSEST-K-SUBSET instance.

Proof of Theorem 4.13. Let I be an instance of the DENSEST-K-SUBGRAPH problem with graph $G = (V, E)$ and positive integer K. In I we ask for a subgraph with K vertices with the maximal number of edges.

From I we construct an instance I' of α-OWA-WINNER, where the set of agents N is E, the set of items is $A = V \cup \{d_1, \dots, d_{\ell-2}\}$ (or V if $\ell = 2$), and we seek a winner set of size $K + \ell - 2$. Agents utilities are set as follows: For each agent e and each item d_j, $1 \le j \le \ell - 2$, the utility of e for d_j is 1. If e is an edge in G that connects vertices u and v, then agent e's utility for u and v is 1 and is 0 for the remaining items from V.

It is easy too see that the items $d_1, \dots, d_{\ell-2}$ all belong to every optimal solution for I'. It is also easy to see that in each optimal solution the utility of each agent e is nonzero (and exactly equal to α_ℓ, the ℓ-th entry of the OWA α used) if and only if both items corresponding to the vertices connected by e are included in the solution. Thus the total utility of every optimal solution for I' is equal to α_ℓ times the number of edges that connect any two vertices corresponding to the items from the solution.

Let $\mathcal{A}$ be a polynomial-time $x(n)$-approximation algorithm for α-OWA-WINNER. If $\mathcal{A}$, returns a solution S for I' with none-zero utility, then the items $d_1, \dots, d_{\ell-2}$ all belong to S. Let us take the vertices corresponding to the items $S \setminus \{d_1, \dots, d_{\ell-2}\}$. The number of the edges connecting these vertices is equal to the total utility of S divided by α_ℓ. Thus, from $x(n)$-approximate solution for I' we can extract an $x(n)$-approximate solution for I. This completes the proof. □

As a further evidence that OWAs that are not-nonincreasing are particularly hard to deal with from the point of view of approximation algorithms, we show that for an extreme example of an OWA family, i.e., for the K-med OWAs, there is a very strong hardness-of-approximation result.

Theorem 4.14. *There exists a constant c such that there is no polynomial-time $(2^{c\sqrt{\lg n}}/n)$-approximation algorithm for K-med-OWA-WINNER unless for some ϵ we have 3-SAT $\in DTIME(2^{n^{3/4+\epsilon}})$.*

Proof. Let us assume that there is a constant c and a polynomial-time $(2^{c\sqrt{\lg n}}/n)$-approximation algorithm $\mathcal{A}$ for K-med-OWA-WINNER. We will show that we can use $\mathcal{A}$ to solve instances of MEBP-V with the same approximation ratio (For the definition and discussion on the complexity of MEBP-V we refer the reader to Section 2.3.). By Lemma 2.2, this will prove our theorem.

Let I be an instance of MEBP-V with bipartite graph $G = (U \cup V, E)$ and positive integer K. In I we ask for a biclique S such that $\|S \cap V\| = K$ and S contains as many edges as possible.

From I we construct an instance I' of K-med-OWA-WINNER in the following way. We let the set of agents N be U, the set of items A be V, and we seek a winner set of size K. The utility of agent u from item v is equal to 1 if and only if u and v are connected in G; otherwise it is 0. Now we note that there is a one-to-one correspondence between the solutions for I and for I'. Let S be a solution for I with x edges: $S \cap V$ is also a solution for I' with the utility at least equal to x/K. Let S be a solution for I' with the utility x. All the agents from U with non-zero utilities, together with S, form a biclique with Kx edges. Thus, from the $(2^{c\sqrt{\lg n}}/n)$-approximation solution for I' we can extract a $(2^{c\sqrt{\lg n}}/n)$-approximation solution for I. This completes the proof. $\square$

As a corollary of the above proof, we also have that Hurwicz[λ]-OWA-Winner is NP-hard (through an almost identical proof, but with a certain dummy candidate, that gets utility 1 from everyone, added, and with the size of the winner set extended by 1).

Corollary 4.15. Hurwicz[λ]-OWA-Winner *is* NP-*hard*

Interestingly, even though Hurwicz[λ] OWA is not nonincreasing, we do show an approximation algorithm for it with a constant approximation ratio. This shows that, indeed, even for not-nonincreasing OWAs, sometimes some form of approximation result is possible (though we will comment on the value of this approximation later).

Proposition 4.16. *Let $\mathcal{A}$ be a β-approximation algorithm for* 1-best-OWA-Winner*. $\mathcal{A}$ is a $\lambda\beta$-approximation algorithm for* Hurwicz[λ]-OWA-Winner*.*

Proof. Let us consider some instance I^H of Hurwicz[λ]-OWA-Winner, where the goal is to pick a set of K items. We construct an instance I^1 that is identical to I^H, but for the 1-best-OWA, and we run algorithm $\mathcal{A}$ on I^1. The algorithm outputs some set $W = \{w_1, \ldots, w_K\}$ (a β-approximate solution for I^1). We claim that W is a $\lambda\beta$-approximate solution for I^H.

Let $W^H = \{w_1^H, \ldots, w_K^H\}$ be an optimal solution for I^H and let $W^1 = \{w_1^1, \ldots, w_K^1\}$ be an optimal solution for I^1. We first note that the following holds (recall the $\vec{x}^{\downarrow}$ notation for sorted sequences from page 40):

$$u_{ut}^{\text{Hurwicz}[\lambda]}(W^H) = \sum_{i=1}^{n} \left(\lambda u_{i,w_1^H}^{\downarrow} + (1-\lambda)u_{i,w_K^H}^{\downarrow}\right) \le \sum_{i=1}^{n} u_{i,w_1^H}^{\downarrow} \le \sum_{i=1}^{n} u_{i,w_1^1}^{\downarrow} = u_{ut}^{\text{1-best}}(W^1).$$

In effect, we have that $u_{ut}^{\text{1-best}}(W^1) \ge u_{ut}^{\text{Hurwicz}[\lambda]}(W^H)$. Now, it is easy to verify that for W (or, in fact, for any set of K items) it holds that:

$$u_{ut}^{\text{Hurwicz}[\lambda]}(W) = \sum_{i=1}^{n} \left(\lambda u_{i,w_1}^{\downarrow} + (1-\lambda)u_{i,w_K}^{\downarrow}\right) \ge \lambda \sum_{i=1}^{n} u_{i,w_1}^{\downarrow} = \lambda u_{ut}^{\text{1-best}}(W).$$

Finally, combining these two inequalities and the fact that W is a β-approximate solution for 1-bestOWA-Winner, we get:

$$u_{ut}^{\text{Hurwicz}[\lambda]}(W) \ge \lambda u_{ut}^{\text{1-best}}(W) \ge \lambda\beta u_{ut}^{\text{1-best}}(W^1) \ge \lambda\beta u_{ut}^{\text{Hurwicz}[\lambda]}(W^H).$$

This completes the proof. $\square$

By using the algorithm GREEDY from Figure 4.1 we get the following corollary.

Corollary 4.17. *There is an algorithm that for* Hurwicz[λ]-OWA-WINNER *with no restrictions on the utility functions achieves approximation ratio* $\lambda(1 - \frac{1}{e})$.

Returning to nonincreasing OWAs, we can even show an example of a PTAS for OWA-WINNER for a certain family OWAs. However, to defeat the relation to the MAXCOVER problem, these OWAs need to be of a very special form: they need to be as similar to the K-best OWA as possible.

Theorem 4.18. *Consider a nonincreasing OWA α, $\alpha = \langle \alpha_1, \dots, \alpha_K \rangle$. Let I be an instance for α-*OWA-WINNER *(where we seek a winner set of size K). An optimal solution for the same instance but with K-*best-*OWA is a $(\sum_{i=1}^{K} \alpha_i)/(K\alpha_1)$-approximate solution for I.*

Proof. Let I be an instance of α-OWA-WINNER described in the statement of the theorem, let W^* be one of its optimal solution, and let W be an optimal solution for the same instance, but with α replaced with the K-best-OWA. Note that W is also an optimal solution for the K-number constant OWA $\beta = \langle \alpha_1, \dots, \alpha_1 \rangle$. We claim that the following inequalities hold (u^{α}_{ut} is defined with respect to the instance I and u^{β}_{ut} is defined with respect to instance I with β as the OWA):

$$u^{\alpha}_{ut}(W) \geq \frac{\sum_{i=1}^{K} \alpha_i}{K\alpha_1} u^{\beta}_{ut}(W) \geq \frac{\sum_{i=1}^{K} \alpha_i}{K\alpha_1} u^{\beta}_{ut}(W^*) \geq \frac{\sum_{i=1}^{K} \alpha_i}{K\alpha_1} u^{\alpha}_{ut}(W^*),$$

The second inequality holds because W is an optimal solution for I with OWA β. To see why the first and the third inequalities hold, let us focus on some agent i. The third inequality is simpler and so we prove it first.

Let $u^*_1, \dots, u^*_k$ be the utilities, in the nonincreasing order, that agent i has for the items in W^*. Thus the utility that i gets from W^* under α is $\sum_{i=1}^{K} \alpha_i u^*_i$. Since for each i, $1 \leq i \leq K$, we have $\alpha_i \leq \alpha_1$, i's utility under α is smaller or equal than i's utility under β, $\sum_{i=1}^{K} \alpha_1 u^*_i$.

We now prove the first inequality. Let $u_1, \dots, u_K$ be the utilities, in the nonincreasing order, that agent i has for the items in W. Our goal is to show that:

$$\begin{aligned}\alpha_1 u_1 + \cdots + \alpha_K u_K &\geq \frac{\sum_{i=1}^{K} \alpha_i}{K\alpha_1} \alpha_1 u_1 + \cdots + \frac{\sum_{i=1}^{K} \alpha_i}{K\alpha_1} \alpha_1 u_K \\ &= \frac{\sum_{i=1}^{K} \alpha_i}{K} u_1 + \cdots + \frac{\sum_{i=1}^{K} \alpha_i}{K} u_K.\end{aligned}$$

This inequality is equivalent to

$$K\alpha_1 u_1 + \cdots + K\alpha_K u_K \geq \sum_{i=1}^{K} \alpha_i u_1 + \cdots + \sum_{i=1}^{K} \alpha_i u_K,$$

which itself is equivalent to

$$u_1(K\alpha_1 - \sum_{i=1}^{K} \alpha_i) + \cdots + u_K(K\alpha_K - \sum_{i=1}^{K} \alpha_i) \geq 0.$$

We can rewrite the left-hand side of this inequality as:

$$(u_1 - u_2)(K\alpha_1 - \sum_{i=1}^{K} \alpha_i) + (u_2 - u_3)(K\alpha_1 + K\alpha_2 - 2\sum_{i=1}^{K} \alpha_i) + \cdots + \\ + (u_{K-1} - u_K)(\sum_{j=1}^{K-1} K\alpha_j - (K-1)\sum_{i=1}^{K} \alpha_i) + u_K(\sum_{j=1}^{K} K\alpha_j - K\sum_{i=1}^{K} \alpha_i).$$

We claim that each summand in this expression is nonnegative. Since $u_1, \dots, u_K$ is a nonincreasing sequence of nonnegative utilities, we have that for each j, $1 \leq j \leq K-1$, $u_j - u_{j+1}$ is nonnegative, and so is u_K. Now fix some t, $1 \leq t \leq K$. We have:

$$\sum_{j=1}^{t} K\alpha_j - t\sum_{i=1}^{K} \alpha_i = \sum_{j=1}^{t}(K-t)\alpha_j - t\sum_{i=t+1}^{K} \alpha_i \\ \geq t(K-t)\alpha_t - t\sum_{i=t+1}^{K} \alpha_i \geq t(K-t)\alpha_t - t(K-t)\alpha_t = 0$$

This completes the proof. □

As a consequence of this theorem, we immediately get the following result.

Theorem 4.19. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function computable in polynomial-time with respect to the value of its argument, such that $f(K)$ is $o(K)$. There is a PTAS for $(K - f(K))$-best-OWA-WINNER.*

Proof. Let us fix some ϵ, $0 < \epsilon < 1$. We give a polynomial time ϵ-approximation algorithm for $(K - f(K))$-best-OWA-WINNER. Since $f(K)$ is $o(K)$, there is some value K_ϵ such that for each $K \geq K_\epsilon$ it holds that $\frac{K-f(K)}{K} \geq \epsilon$. If for our input instance we are to find a winner set of size K, $K \geq K_\epsilon$, then we simply run the polynomial-time algorithm for K-best-OWA. Otherwise, we seek a winner set of size at most K_ϵ and we try all subsets of items of size K. Since, in this case, K is bounded by a constant, our algorithm runs in polynomial time. □

Both Corollary 4.17 and Theorem 4.19 have a bitter-sweet taste. In essence, they say that instead of using a particular OWA family (either Hurwicz[λ] or $(K - f(K))$-best OWA), we might as well use a different, simpler one (1-best OWA or K-best OWA). If one wanted Hurwicz[λ] OWA or $(K-f(K))$-best OWA for some very important reason, then these algorithms are insufficient (though, one could interpret them as suggesting that such a very important reason is unlikely).[1]

Still, Theorem 4.19 is a very interesting result when contrasted with Theorem 4.13. Theorem 4.19 says that there is a PTAS for α-OWA-WINNER for OWA family $\langle 1, \dots, 1, 0 \rangle$, whereas Theorem 4.13 suggests that it is unlikely that there is a

[1]This comment applies to Hurwicz[λ] for large values of λ only.

```
Notation:
Φ ← a map defining the number of free slots per agent. Initially for each agent $i$ we
have $\Phi[i] = \ell$.
1
2 $x \leftarrow m\mathrm{W}\left(\frac{K}{\ell}\right)\frac{\ell}{K}$;
3 $S \leftarrow \emptyset$;
4 for $i \leftarrow 1$ to $K$ do
5     $a \leftarrow \mathrm{argmax}_{a \in A \setminus S} \|\{j \mid \Phi(j) > 0 \land pos_j(a) \geq x\}\|$;
6     foreach $j \in \{j \mid \Phi(j) > 0\}$ do
7         if $pos_j(a) \geq x$ then
8             $\Phi[j] \leftarrow \Phi[j] - 1$;
9     $S \leftarrow S \cup \{a\}$;
10 return $S$
```

Figure 4.2: The algorithm PosBoundAndGreedy for nonincreasing OWAs where at most first ℓ entries are nonzero, for the case of Borda-based utilities.

constant-factor approximation algorithm for α-OWA-Winner with OWA family $\langle 0, 1, \ldots, 1\rangle$. Even though these two OWA families seem very similar, the fact that one is nonincreasing and the other one is not makes a huge difference in terms of approximability of our problem.

We note that all the results from this section apply both to the general and to the approval based case. Indeed, in the proofs of our positive results we did not make any assumptions regarding agents' utilities. We also presented the proofs of hardness in their more restricted form, i.e., for a specific case of approval utilities (less general utilities give a more general theorem).

4.3 Polynomial Time Approximation Schemes: Borda Utilities

We now focus on OWA-Winner with Borda-based utilities. In this case the difference between nonincreasing OWAs and those that are not nonincreasing is much less pronounced than in the general case (or, in the approval-based case) and very strong approximation algorithms exist. Indeed, we show PTASes for many variants of α-OWA-Winner with Borda-based utilities.

We start by discussing the algorithm PosBoundAndGreedy, presented in Figure 4.2. The algorithm PosBoundAndGreedy works for nonincreasing OWAs where only some initial ℓ positions are nonzero. By $\mathrm{W}(\cdot)$ we mean Lambert's W function, that is, a function that for each $x \in \mathbb{R}_+$ satisfies the equation $x = \mathrm{W}(x)e^{\mathrm{W}(x)}$ (and, thus, $\mathrm{W}(x)$ is $O(\log(x))$). The idea behind the algorithm is as follows: It proceeds in K iterations (where K is the winner-set size that we seek) and in each iteration it introduces one new item into the winner set. For each agent it considers only the top $x = m\mathrm{W}\left(\frac{K}{\ell}\right)\frac{\ell}{K}$ items with highest utilities and in a given iteration it greedily picks an item a that maximizes the number of agents that (1) rank a among items with highest x utilities, and (2) still have "free slots" (an agent has a free slot if among the so-far-selected winners less than ℓ have utilities among the x highest ones for this agent). Below we give an example of how the algorithm works.

Example 4.20. Consider the utility profile with $n = 10$ agents, $m = 8$ items $a_1, a_2, \dots, a_8$, and the utilities extracted from the following preference orders using Borda PSF.

$$\begin{aligned}
\texttt{agent}_1 &: a_1 \succ a_2 \succ a_3 \succ a_4 \succ a_5 \succ a_6 \succ a_7 \succ a_8 \\
\texttt{agent}_2 &: a_2 \succ a_1 \succ a_4 \succ a_3 \succ a_7 \succ a_6 \succ a_5 \succ a_8 \\
\texttt{agent}_3 &: a_1 \succ a_3 \succ a_7 \succ a_4 \succ a_6 \succ a_5 \succ a_8 \succ a_2 \\
\texttt{agent}_4 &: a_2 \succ a_1 \succ a_4 \succ a_5 \succ a_3 \succ a_8 \succ a_7 \succ a_6 \\
\texttt{agent}_5 &: a_1 \succ a_6 \succ a_5 \succ a_8 \succ a_7 \succ a_4 \succ a_2 \succ a_3 \\
\texttt{agent}_6 &: a_3 \succ a_2 \succ a_1 \succ a_5 \succ a_4 \succ a_7 \succ a_8 \succ a_6 \\
\texttt{agent}_7 &: a_1 \succ a_2 \succ a_6 \succ a_4 \succ a_7 \succ a_6 \succ a_8 \succ a_3 \\
\texttt{agent}_8 &: a_1 \succ a_3 \succ a_5 \succ a_7 \succ a_8 \succ a_6 \succ a_4 \succ a_2 \\
\texttt{agent}_9 &: a_1 \succ a_4 \succ a_3 \succ a_5 \succ a_2 \succ a_6 \succ a_7 \succ a_8 \\
\texttt{agent}_{10} &: a_8 \succ a_6 \succ a_3 \succ a_4 \succ a_5 \succ a_2 \succ a_1 \succ a_7
\end{aligned}$$

So, for example, $\texttt{agent}_1$ has utility 7 for a_1, utility 6 for a_2, 5 for a_3, and so on. We take $K = 4$ and we are using the 2-best OWA, so that $\ell = 2$. Consequently, the algorithm will consider only the first $x = \lceil m\mathrm{W}\left(\frac{K}{\ell}\right)\frac{\ell}{K}\rceil = 4$ positions of the agents' rankings ($\mathrm{W}(2) \approx 0.8562$). Initially, each agent has $\ell = 2$ free slots. In the first iteration, the algorithm selects an item that is most frequent among the first 4 positions of the agents' rankings, that is a_1. Every agent except for $\texttt{agent}_{10}$ ranks a_1 among the first 4 positions, so after the first iteration every agent except for $\texttt{agent}_{10}$ is left with one free slot; $\texttt{agent}_{10}$ still has 2 free slots. In the second iteration, the algorithm selects the most frequent item (excluding a_1), that is either a_3 or a_4. Ties are broken arbitrarily, so let us assume that it picks a_3. Every agent that ranks a_3 among the first 4 positions is assigned a_3 and loses one free slot. Now the agents $\texttt{agent}_4$, $\texttt{agent}_5$, $\texttt{agent}_7$, and $\texttt{agent}_{10}$ have one free slot left and the remaining ones have no free slots. In the third iteration, the algorithm considers only the 4 agents with free slots. The two most frequent items that are ranked among the first 4 positions by these 4 agents are a_4 and a_6; let us assume the algorithm picks a_4. After this iteration only $\texttt{agent}_5$ has a free slot. Since $K = 4$, the algorithm is allowed to pick one more item ranked among the first 4 positions by $\texttt{agent}_5$, which is either of a_5, a_6, and a_8. Say, the algorithm picks a_5.

Theorem 4.21. *Fix a positive integer ℓ and let α be a nonincreasing OWA where at most first ℓ entries are nonzero. Given an instance I of α-OWA-WINNER, the algorithm* POSBOUNDANDGREEDY *from Figure 4.2 computes a $\left(1 - \frac{2\mathrm{W}(K/\ell)}{K/\ell}\right)$-approximate solution for I in polynomial-time.*

Proof. Consider an instance I of α-OWA-WINNER, with n agents, m items, and where we seek a winner set of size K. Let $x = m\mathrm{W}\left(\frac{K}{\ell}\right)\frac{\ell}{K}$. Since we use an OWA where an agent's total utility from a winner set W depends on this agent's utilities for the top ℓ items from W (from this agent's point of view), we introduce the notion of free slots for each agent. Initially, each agent has ℓ free slots. Whenever an agent j has a free slot and the algorithm selects an item a such that from j's perspective a is among x items with highest utilities, we say that a starts occupying one free slot of i. Consequently, after such item is selected, i has one free slot less.

Let n_i denote the total number of free slots of all the agents after the i-th iteration of the algorithm; $n_0 = \ell n$. We will show by induction that $n_i \leq \ell n\left(1 - \frac{x}{\ell m}\right)^i$. Indeed, the inequality is true for $i = 0$. Let us assume that it is true for some i: $n_i \leq \ell n\left(1 - \frac{x}{\ell m}\right)^i$. Let F_i denote the set of agents that have free slots after iteration i. There are at least $\frac{n_i}{\ell}$ such agents. For $j \in F_i$, let $S(j)$ be the number of j's top-x items that were not included in the solution yet. If $j \in F_i$ has s free slots, then $S(j) = (x - \ell + s)$. Thus we have that $\sum_{j \in F_i} S(j) \geq n_i + (x - \ell)\frac{n_i}{\ell} = \frac{n_i x}{\ell}$. By the pigeonhole principle, there exists an item that is among top-x items for at least $\frac{n_i x}{\ell m}$ agents from F_i. Thus, after the $(i+1)$-th iteration of the algorithm, the total number of free slots is at most:

$$n_{i+1} \leq n_i - \frac{n_i x}{\ell m} = n_i\left(1 - \frac{x}{\ell m}\right) \leq \ell n\left(1 - \frac{x}{\ell m}\right)^{(i+1)}$$

In effect, at the end of the algorithm the total number of free slots is at most:

$$\begin{aligned} n_K &\leq \ell n\left(1 - \frac{x}{\ell m}\right)^K \leq \ell n\left(1 - \frac{x}{\ell m}\right)^K = \ell n\left(1 - \frac{\mathrm{W}\left(\frac{K}{\ell}\right)}{K}\right)^K \\ &= \ell n\left(\frac{1}{e}\right)^{\mathrm{W}(K/\ell)} = \frac{\ell n \mathrm{W}(K/\ell)}{K/\ell}. \end{aligned}$$

The number of occupied slots at the end of the algorithm is, thus, at least equal to $\left(\ell n - \frac{\ell n\mathrm{W}(K/\ell)}{K/\ell}\right)$. Every item that occupies an agent's slot has utility for this agent greater or equal to $\left(m - \frac{m\mathrm{W}(K/\ell)}{K/\ell}\right)$. Now, we will assess the OWA coefficients for the utilities of the items from the solution. If for some agent i the utility of an item a ($u_{i,a}$) is taken with the coefficient α_p ($p > 1$), then in the solution there must exist an item a' such that $u_{i,a'} \geq u_{i,a}$ and such that $u_{i,a'}$ is taken with coefficient α_{p-1}. Thus, there must exist at least $\frac{1}{\ell}\left(\ell n - \frac{\ell n\mathrm{W}(K/\ell)}{K/\ell}\right)$ occurrences of the items with the utilities greater than $\left(m - \frac{m\mathrm{W}(K/\ell)}{K/\ell}\right)$ and such that these utilities are taken with the coefficient α_1. By repeating the same reasoning for the remaining occurrences of the items from the solution, we get that the total utility of the agents is lower-bounded by:

$$\begin{aligned}\left(\ell n - \frac{\ell n\mathrm{W}(K/\ell)}{K/\ell}\right)\left(m - \frac{m\mathrm{W}(K/\ell)}{K/\ell}\right)\frac{1}{\ell}\sum_{i=1}^{\ell}\alpha_i &\geq \ell nm\left(1 - \frac{\mathrm{W}(K/\ell)}{K/\ell}\right)^2\frac{1}{\ell}\sum_{i=1}^{\ell}\alpha_i \\ &\geq nm\left(1 - \frac{2\mathrm{W}(K/\ell)}{K/\ell}\right)\sum_{i=1}^{\ell}\alpha_i.\end{aligned}$$

Since the total utility of all the agents can be upper-bounded by $nm\sum_{i=1}^{\ell}\alpha_i$, we get the desired approximation ratio. □

Theorem 4.22. *Fix a value ℓ and let α be a family of nonincreasing OWAs that have nonzero values on top ℓ positions only. There is a PTAS for* α-OWA-WINNER *for the case of Borda-based utilities.*

Proof. For every ϵ we show a polynomial algorithm with approximation ratio $(1-\epsilon)$. Consider some ϵ, $0 \leq \epsilon \leq 1$. There exists a value K_ϵ such that for each $K > K_\epsilon$ it holds that $\frac{2\mathrm{W}(K/\ell)}{K/\ell} < \epsilon$. For each instance I of α-OWA-WINNER where we seek winner set of size at least K_ϵ, we run the algorithm POSBOUNDANDGREEDY from Figure 4.2. For the remaining cases, where the winner-set size is bounded by a constant, we use a brute-force algorithm. □

Using the above result, we can also obtain a PTAS for OWA-WINNER for geometric progression OWAs, for the case of Borda utilities. This is quite a useful result since some of our scenarios from Chapter 3 yield OWAs of this form.

Corollary 4.23. *Fix a value $p > 1$. There is a PTAS for* gprog[p]-OWA-WINNER *for the case of Borda-based utilities.*

Proof. Our goal is to show an algorithm that for a given value ϵ, $\epsilon > 0$, in polynomial time outputs a $(1-\epsilon)$-approximate solution for gprog[p]-OWA-WINNER. Let us fix the value of such ϵ. The idea of our proof is to truncate the vector describing gprog[p] OWA to consider only some ℓ nonzero items on the top, where ℓ depends on ϵ only, and to run the algorithm from Theorem 4.22.

For a given number t, let S_t be the sum of the first t coefficients of gprog[p]. We have: $S_t = \mathrm{gprog}[p]_t + \mathrm{gprog}[p]_{t-1} + \cdots + \mathrm{gprog}[p]_1 = p^{K-t} + p^{K-(t-1)} + \cdots + p^{K-1} = p^{K-t}\frac{p^t-1}{p-1}$. We fix $\ell = \lceil \log_p(\frac{2}{\epsilon}) \rceil$. Now, consider the ratio $r = S_\ell/S_K$:

$$r = \frac{S_\ell}{S_K} = p^{K-\ell}\frac{p^\ell - 1}{p^K - 1} > p^{K-\ell}\frac{p^\ell - 1}{p^K} = 1 - \frac{1}{p^\ell} \geq 1 - \frac{1}{p^{\log_p(\frac{2}{\epsilon})}} = 1 - \frac{\epsilon}{2}.$$

Intuitively, the above inequality says that $1-\frac{\epsilon}{2}$ fraction of the total weight of gprog$[p]$ OWA is concentrated in its first ℓ coefficients.

Let gprog$[p]_{|\ell}$ denote the OWA obtained from gprog$[p]$ by replacing all coefficients with indices greater than ℓ with 0. Let $\mathcal{A}$ be a $(1-\frac{\epsilon}{2})$-approximation algorithm for gprog$[p]_{|\ell}$-OWA-Winner. From Theorem 4.22 we know that such an algorithm exists. It is easy to see that $\mathcal{A}$ is a $(1-\epsilon)$-approximation algorithm for gprog$[p]$-OWA-Winner. Indeed, the utility under gprog$[p]_{|\ell}$ for every K-element set W is close to the utility of W under gprog$[p]$ (recall the $\vec{x}^{\downarrow}$ notation from page 40 for sorted sequences; the inequality in the second line holds because for each i we have $\sum_{g=1}^{\ell} \mathrm{gprog}[p]_g u^{\downarrow}_{i,w_h} \leq \sum_{j=1}^{\ell} \mathrm{gprog}[p]_j u^{\downarrow}_{i,w_j}$):

$$
\begin{aligned}
u_{ut}^{\mathrm{gprog}[p]}(W) &= \sum_{i=1}^{n}\sum_{j=1}^{K} \mathrm{gprog}[p]_j u^{\downarrow}_{i,w_j} \\
&\leq \sum_{i=1}^{n}\left(\sum_{j=1}^{\ell} \mathrm{gprog}[p]_j u^{\downarrow}_{i,w_j} + \sum_{h=\ell+1}^{K} \mathrm{gprog}[p]_h \frac{\sum_{j=1}^{\ell} \mathrm{gprog}[p]_j u^{\downarrow}_{i,w_j}}{\sum_{g=1}^{\ell} \mathrm{gprog}[p]_g}\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{\ell} \mathrm{gprog}[p]_j u^{\downarrow}_{i,w_j}\left(1+\frac{\sum_{h=\ell+1}^{K} \mathrm{gprog}[p]_h}{\sum_{g=1}^{\ell} \mathrm{gprog}[p]_g}\right) \\
&\leq \sum_{i=1}^{n}\sum_{j=1}^{\ell} \mathrm{gprog}[p]_j u^{\downarrow}_{i,w_j}\left(1+\frac{\epsilon}{2}\right) = \left(1+\frac{\epsilon}{2}\right) u_{ut}^{\mathrm{gprog}[p]_{|\ell}}(W).
\end{aligned}
$$

From which we get that for every W:

$$u_{ut}^{\mathrm{gprog}[p]_{|\ell}}(W) \geq (1-\frac{\epsilon}{2}) u_{ut}^{\mathrm{gprog}[p]}(W).$$

This completes the proof because algorithm $\mathcal{A}$ returns a $(1-\frac{\epsilon}{2})$-approximate solution for gprog$[p]_{|\ell}$-OWA-Winner and $(1-\frac{\epsilon}{2})(1-\frac{\epsilon}{2}) \geq 1-\epsilon$. □

Also, by Proposition 4.16 and Theorem 4.22 we get the following result for the Hurwicz$[\lambda]$-OWA-Winner problem.

Corollary 4.24. *For each positive ϵ, there is an algorithm that for* Hurwicz$[\lambda]$-OWA-Winner *for the case of Borda-based utilities achieves approximation ration $\lambda(1-\epsilon)$.*

Interestingly, Theorem 4.22 can be generalized to the case of arbitrary OWAs (not necessarily nonincreasing) that have nonzero entries among the top ℓ positions only (where ℓ is a constant.[2]) The idea of our algorithm is very similar to that presented as the algorithm PosBoundAndGreedy in Figure 4.2, but this time we take more

[2]However, if one reads our proof carefully, it is clear that it generalizes to some values of ℓ that depend on K, but which are sufficiently small; e.g., it works for $\ell = K^{\frac{1}{3}}$).

care in choosing the winning items, so that we ensure that a large collection of voters ranks at least ℓ winners on positions with high utility values. Specifically, we use the following lemma, which is a direct consequence of Theorem 4.21.

Lemma 4.25. *Consider a set N of n agents and a set A of m items, where the agents rank the items from the most preferred ones to the least preferred ones. Let K, ℓ, and t be some positive integers such that $K \leq m$, $\ell \leq K$, and $t \leq \ell$. Let $x = \frac{m\mathrm{W}(K/\ell)}{K/\ell}$. There is a polynomial-time algorithm that finds a collection C of up to K/ℓ items such that there are at least $n(1 - \frac{\mathrm{W}(K/\ell)}{K/\ell})$ agents that each rank at least one member of C between positions $(t-1)x+1$ and tx.*

Proof. To see that this lemma holds, it suffices to analyze the proof of Theorem 4.21 for $\ell = 1$ and note that the proof works equally well irrespectively of whether we consider the positions 1 through x, or $x+1$ through $2x$, or any other segment of x positions in the agents' preference orders. □

Theorem 4.26. *Fix a positive integer ℓ and let α be a family of OWAs that have nonzero entries on top ℓ positions only. There is a polynomial-time $(1 - \frac{2\ell\mathrm{W}(K/\ell)}{K/\ell})$-approximation algorithm for α-OWA-Winner for the case of Borda-based utilities.*

Proof. Consider an input instance I of α-OWA-Winner with the set $N = [n]$ of agents, with the set A of m items, and where we seek winner set of size K. We consider the agents' utilities to be represented by preference orders over A (recall the discussion of Borda-based utilities in Chapter 3 and the fact that we can decode these preference orders from the Borda-based utilities of the agents). Let $\alpha = \langle \alpha_1, \ldots, \alpha_\ell, 0, \ldots, 0 \rangle$ be the OWA used in this instance. We set $x = \frac{m\mathrm{W}(K/\ell)}{K/\ell}$.

Our algorithm proceeds in ℓ iterations. We set $N^{(0)} = N$ and $n^{(0)} = n$. In the i-th iteration, $1 \leq i \leq \ell$, the algorithm operates as follows: Using the algorithm from Lemma 4.25, we find a set $A^{(i)}$ of up to K/ℓ items such that at least $n^{(i-1)}(1 - \frac{\mathrm{W}(K/\ell)}{K/\ell})$ of the agents from the set $N^{(i-1)}$ each rank at least one of these items among positions $(i-1)x+1, \ldots, ix$ of their preference orders. We let $N^{(i)}$ be the set of these agents and we set $n(i) = \|N^{(i)}\|$. Finally, we set $W = \bigcup_{i=1}^{\ell} A^{(i)}$ and return W as the set of winners (it is easy to see that W contains at most K items; if K contains fewer than K items then we supplement it with $K - \|W\|$ arbitrarily chosen ones).

By the construction of our algorithm, each of the agents from the set $N^{(\ell)}$ ranks at least ℓ items from the set W on positions no worse than ℓx. Thus the total utility that the agents from the set N derive from the solution W is at least:

$$n^{(\ell)} \left(\sum_{i=1}^{\ell} \alpha_i \right) (m - x\ell).$$

This is so, because for each i, $1 \leq i \leq \ell$, each of the agents in the set $N^{(\ell)}$ derives utility at least $\alpha_i(m - x\ell)$ from the agent that he ranks as i'th best among the items from W.

By construction of our algorithm, we have:

$$n^{(\ell)} \geq n\left(1 - \frac{\mathrm{W}(K/\ell)}{K/\ell}\right)^{\ell} \geq n\left(1 - \frac{\ell\mathrm{W}(K/\ell)}{K/\ell}\right).$$

Thus, the total utility obtained by the agents is at least:

$$\begin{aligned}
u^{\alpha}_{ut}(W) &\geq n\left(1 - \frac{\ell\mathrm{W}(K/\ell)}{K/\ell}\right)\left(\sum_{i=1}^{\ell}\alpha_i\right)(m - x\ell) \\
&= n\left(1 - \frac{\ell\mathrm{W}(K/\ell)}{K/\ell}\right)\left(\sum_{i=1}^{\ell}\alpha_i\right)\left(m - \ell\frac{m\mathrm{W}(K/\ell)}{K/\ell}\right) \\
&= nm\left(\sum_{i=1}^{\ell}\alpha_i\right)\left(1 - \frac{\ell\mathrm{W}(K/\ell)}{K/\ell}\right)\left(1 - \frac{\ell\mathrm{W}(K/\ell)}{K/\ell}\right) \\
&\geq nm\left(\sum_{i=1}^{\ell}\alpha_i\right)\left(1 - \frac{2\ell\mathrm{W}(K/\ell)}{K/\ell}\right)
\end{aligned}$$

Now, since the maximum possible total utility of all the agents is upper-bounded by $nm(\sum_{i=1}^{\ell}\alpha_i)$, we have that our algorithm has approximation ratio $(1 - \frac{2\ell\mathrm{W}(K/\ell)}{K/\ell})$. It is clear that it runs in polynomial time, and so the proof is complete. □

Based on Theorem 4.26, we can immediately generalize Theorem 4.22.

Theorem 4.27. *Fix a value ℓ and let α be a family of OWAs that have nonzero values on top ℓ positions only. There is a PTAS for α-OWA-WINNER for the case of Borda-based utilities.*

It is interesting to compare Theorems 4.21 and 4.26. Even though the latter one covers a larger set of cases, the algorithm it implies achieves a notably weaker approximation ratio (even if still sufficient to obtain a PTAS). This shows that OWAs that are not nonincreasing are harder to deal with even for Borda-based utilities (even if the difference is much smaller than in the general case). Theorems 4.26 and 4.27 are also quite interesting in conjunction with Theorem 4.13. In particular, they show that it seems impossible to generalize Theorem 4.13 to the case of Borda-based utilities. It might be surprising at first, because it was possible to generalize Theorem 4.4 to the case of Borda utilities (as Theorem 4.5) and, indeed, the main ideas of these proofs are similar.

It is also interesting to note that there is nothing in the proofs of Theorems 4.21 and 4.26 that would stop us from using them for the case where ℓ depends on K. In particular, Theorem 4.21 still gives an approximation ratio higher than $1 - \frac{1}{e}$ of the greedy algorithm even for $\frac{K}{10}$-best OWA. For the case, of Theorem 4.26, it stays useful (i.e., still yields even a PTAS) for values of ℓ in $O(K^{0.5-\epsilon})$, for each positive ϵ (the same is true for Theorem 4.21 as long as $\ell(K)$ is such that $\frac{\mathrm{W}(K/\ell(K))}{K/\ell(K)}$ goes to 0 when K goes to ∞).

4.4 Summary

We have investigated the computational feasibility of the problem of selecting a collective set of items, depending on the various assumptions we make about the agents' utilities and the choice of the OWA vector. Table 4.1 gives a summary of our results. We note that many of these results are directly related to the OWA families that appear in the settings from Chapter 3, that were our motivating force.

Some of our results appear negative, while some others (especially in the case of Borda utilities) are on the positive side. However, the way the results should be interpreted depends on the application domain.

Table 4.1: Summary of our results for the OWA families from Chapter 3. For each OWA family we provide four entries: In the first row (for a given OWA family) we give its worst case complexity (in the general case and in the Borda utilities case), and in the second row we list the best known approximation result (in the general case and in the Borda utilities case). We write K to mean the cardinality of the winner set that we seek. In the "References" column we point to the respective result in this chapter/literature (for negative results we indicate simplest type of utilities where it holds, for positive results the most general type of utilities where it holds). For approximation: DkS-bounded and MEBP-bounded mean, respectively, inapproximability results derived from the Densest-k-Subgraph problem and from the Maximum Edge Biclique Problem

OWA family	general utilities	Borda utilities	References
k-median (k fixed)	NP-hard	NP-hard	Prop. 4.2 (approval and Borda)
	DkS-bounded	PTAS	Thm. 4.13 (approval) and 4.27 (Borda)
K-median	NP-hard	NP-hard	Thm. 4.4 (approval) and 4.5 (Borda)
	MEBP-bounded	?	Thm. 4.14 (approval), open (Borda)
1-best	NP-hard	NP-hard	literature [188,245]
	$(1-\frac{1}{e})$-approx.	PTAS	literature [188], Thm. 4.22 (Borda)
k-best (k fixed)	NP-hard	NP-hard	Prop. 4.2 (approval and Borda)
	$(1-\frac{1}{e})$-approx.	PTAS	Thm. 4.11 (general) and 4.22 (Borda)
$(K-1)$-best	NP-hard	NP-hard	Thm. 4.4 (approval) and 4.5 (Borda)
	PTAS	PTAS	Thm. 4.18 (general)
K-best	P	P	folk result
arithm. progression	NP-hard	?	Corol.4.7 (approval), open (Borda)
	$(1-\frac{1}{e})$-approx.	$(1-\frac{1}{e})$-approx.	Thm. 4.11 (general)
geom. progression	NP-hard	?	Corollary 4.7 (approval), open (Borda)
	$(1-\frac{1}{e})$-approx.	PTAS	Thm 4.11 (general), Corol. 4.23 (Borda)
Hurwicz[λ]	NP-hard	?	Corol. 4.15 (approval)
	$\lambda(1-\frac{1}{e})$-approx.	$\lambda(1-\epsilon)$-approx.	Corollary 4.17 (general)
		for each $\epsilon > 0$	Corollary 4.24 (Borda)

.

The application of our approximation algorithms in high-stake domains, such as political elections is addressed in Chapter 5. In low-stake applications domains (which can include some committee elections, and of course group recommender systems), we are often not interested in the exact optimal solution, and usually the solutions with sufficiently good quality are acceptable. Consequently, applying our approximation algorithms to these settings seems perfectly reasonable.

Our research leads to many open problems. In particular, one might want to strengthen our approximation algorithms, provide algorithms for more general cases, provide more inapproximability results. Among these problems, a particularly interesting one regards the approximability of OWA-WINNER for the arithmetic progression family of OWAs. For this case, our set of results is very limited. In particular, can one provide a PTAS for arithmetic-progression OWAs under Borda-based utilities? Can one do so for $\frac{K}{2}$-best OWAs/K-median OWAs?

Chapter 5

Proportional Representation as Resource Allocation: Approximability

5.1 Introduction

In this chapter we consider the problem of selecting a collective set of items in the disjunctive model (for the formal definition of the problem we refer the reader to Chapter 3). This means that we consider the problem of selecting K items and assigning each agent to exactly one of them, to maximize agents' satisfaction. Naturally, the satisfaction of an agent from the set of selected items is simply her satisfaction from the item that she is assigned to. We consider the pure disjunctive variant of the model, where the satisfaction of the agent from the set of items is just her satisfaction from the most preferred one that is available, and a variant in which each selected item is assigned to roughly the same number of agents (which is a special case of the capacitated disjunctive version of the problem of selecting a collective set of items).

Further, we consider the case where the utilities of the agents are derived from their preference rankings by applying a positional scoring function (PSF), mainly focusing on the Borda count PSF.

These variants of our problem were first considered by Chamberlin–Courant [47] and Monroe [210] in the context of selecting a set of candidates in elections. When choosing a K-member committee, the Monroe and Chamberlin–Courant rules work as follows. We assume that m candidates participate in the election and that the society consists of n voters, who each rank the candidates, expressing their preferences about who they would like to see as their representative in the committee. For each voter these election systems assign a single candidate as their representative, respecting the following rules:

(a) altogether exactly K candidates are assigned to the voters. For the Monroe rule, each candidate is assigned either to about $\frac{n}{K}$ voters or to none; for the Chamberlin–Courant rule there is no such restriction and each committee member may be representing a different number of voters (the committee should take this into account in its operation, e.g., by means of weighted voting).

(b) the candidates are selected and assigned to the voters optimally, maximizing the total (societal) satisfaction.

The total satisfaction is calculated on the basis of individual satisfactions. We assume that there is a function $\alpha\colon \mathbb{N} \to \mathbb{N}$ such that $\alpha(i)$ measures how well a voter is represented by the candidate that this voter ranks as i'th best. The function α is the same for each voter. We can view α as a *satisfaction function*, and so it should be non-increasing. For example, it is typical to use the Borda count scoring function which is defined as $\alpha_{\mathrm{B}}^{m} = m - i$.

The two election systems, Chamberlin–Courant's and Monroe's, fit naturally into the framework of selecting a collective set of items in the disjunctive model. We can identify candidates with alternatives (items)[1] and voters with agents. With this correspondence, saying that a voter i is represented by a candidate a is analogous to saying that an agent i is assigned an item a. When viewed from this perspective, maximizing the satisfaction of the voters from their representatives in the Chamberlin–Courant rule corresponds to maximizing the satisfaction of the agents from the items in the disjunctive variant of the problem of selecting a collective set of items. Similarly, the problem of selecting winners in the Monroe system can be viewed as a special case of the disjunctive capacitated variant of the problem of selecting a collective set of items.

Paying a tribute to the origin of the considered problems, we will refer to the disjunctive variant of the model for selecting a collective set of items as to the Chamberlin–Courant's case, and to the disjunctive variant in which each selected item must be assigned to the roughly the same number of agents, as to the Monroe's case.

As we mentioned in Chapter 3.3.1, the two considered problems have multiple applications, however considering these problems in the context of elections is particularly appealing. The Monroe and Chamberlin–Courant rules create a useful connection between the voters and their representatives that makes it possible to achieve both candidates' accountability to the voters and proportional representation of voters' views. Among common voting rules, the Monroe and Chamberlin–Courant rules seem to be quite unique in having *both* these properties.[2] For example, First Past the Post system (where the voters are partitioned into districts with a separate

[1]To emphasize the fact that each agent can be assigned to (and derive her utility from) a single item only, throughout this chapter we will often refer to the items as to the alternatives. Sometimes, to emphasize the context of elections, we will use the term 'candidate' instead of 'alternative'.

[2]We stress, however, that we understand these properties in intuitive terms and that we are not making any formal claims here. However, we do point the reader to the recent comparison of multiwinner voting rules of Elkind et al. [96] for some interesting discussion of good properties of these rules.

single-winner Plurality election in each) can give very disproportionate results (forcing some of the voters to be represented by candidates they dislike). On the other side of the spectrum are the party-list systems, which achieve very good proportionality, but in which the voters vote for the parties, based on these votes the parties receive numbers of seats in the parliament, and then the parties distribute the seats among their members (usually following publicly available lists of the parties' candidates). This makes the elected candidates feel more accountable to apparatchiks of their parties than to the voters. Somewhere between the First Past the Post system and the party-list systems, we have the single transferable vote rule (STV).

Since it is NP-hard to determine winners in the Monroe and Chamberlin–Courant rules [27,188,245,283], the goal of this chapter is to provide effective approximation algorithms for the problem of finding winners under the Monroe and Chamberlin–Courant rules.

The use of approximation algorithms for Chamberlin–Courant's and Monroe's rules in real-life applications requires some discussion. For example, their use is naturally justified in the context of recommendation systems or other resource allocation variants described in Chapter 3.1. Here the strive for optimality is not crucial since a good but not optimal selection is still very useful and nobody would object if we replaced the optimal selection with an approximate one (given that the optimal one is hard to calculate).

On the other hand, the use of approximation algorithms in elections requires some care. It is conceivable that the electoral commission finds an allocation of voters to candidates with a certain value of satisfaction and one of the parties participating in the election finds an allocation with a better one. This can lead to a political deadlock. There are two ways of avoiding this. Firstly, an approximation algorithm can be fixed by law. In such a case, it becomes an acting voting rule and a new way to measure fairness in the society. Secondly, an electoral commission may calculate the allocation, but also publish the raw data and issue a call for submissions. If, within the period specified by law, nobody can produce a better allocation, then the committee goes ahead and announces the result. If someone produces a better allocation, then the electoral commission uses the latter one.

The use of approximation algorithms is even more natural in elections with partial ballots. Indeed, even if we use an exact algorithm to calculate the winners, the results will be approximate anyway since the voters provide us with approximations of their real preferences rather than the exact ones.

Our Results

We consider both utilitarian and egalitarian variant of the Chamberlin–Courant and Monroe rules. In the utilitarian variant the assignment should maximize the total satisfaction calculated as the sum of the voters' individual satisfactions with their representatives. In the egalitarian variant, the assignment should maximize the total satisfaction calculated as the satisfaction of the worst-off voter.

We obtain the following results:

1. For the egalitarian cases, the Monroe and Chamberlin–Courant rules are hard to approximate up to any constant factor (see Theorems 5.1 and 5.3).

2. For the utilitarian framework we show the following. For the Monroe rule with the Borda scoring function we give a $(0.715-\epsilon)$-approximation algorithm (Theorem 5.8; often, the ratio is much better; see Section 5.4). In case of an arbitrary positional scoring function we give a $(1-\frac{1}{e})$-approximation algorithm (Theorem 5.9). We recall that as the corollary of Theorem 4.22 from Chapter 4 we have a polynomial-time approximation scheme for the Chamberlin–Courant rule with the Borda scoring function.

3. We provide empirical evaluation of our algorithms for the utilitarian framework, both on synthetic and on real-life data. This evaluation shows that in practice our best algorithms achieve approximation ratios at least 0.9, and even better results are typical (see Section 5.5).

4. We show that our algorithms work very well in the setting where voters do not necessarily rank all the candidates, but rather provide the truncated ballots in which they rank several most preferred candidates only (usually at least three). We provide theoretical guarantees on the performance of our algorithms (Propositions 5.6 and 5.12) as well as empirical evaluation (see Section 5.5.4).

Our results show that, as long as one is willing to accept approximate solutions, it is possible to use the utilitarian variants of the Monroe and Chamberlin–Courant rules in practice. This view is justified both from the theoretical and from the empirical point of view. Due to our negative results, we did not perform empirical evaluation for the egalitarian variants of the rules, but we believe that this is an interesting future research direction.

For the approximability results for winner determination under Chamberlin–Courant's and Monroe's rules, but with k-approval positional scoring function used instead of the Borda one, we refer the reader to Chapter 6.

5.2 Preliminaries

In this section we first recall the definition of the positional scoring rules. Next, we recall the definition of the capacitated disjunctive variant of the problem of selecting the collective set of items, which was first introduced in Chapter 3.1, and finally, we discuss which restrictions of this problem correspond to the winner determination problem for the Monroe and Chamberlin–Courant voting rules. For the definitions of the basic notions such as preference orders and positional scoring rules we refer the reader to Chapter 3.

Positional scoring rules. A *positional scoring function* (PSF) is a function $\alpha^m\colon [m] \to \mathbb{N}$. A PSF α^m is a *decreasing positional scoring function* (DPSF) if for each $i, j \in [m]$, if $i < j$ then $\alpha^m(i) > \alpha^m(j)$. Intuitively, a DPSF γ^m measures an agent's satisfaction and we assume that for each DPSF γ^m it holds that $\gamma^m(m) = 0$ (an agent is completely not satisfied being assigned her least desired alternative). Sometimes we write α instead of α^m, when it cannot lead to a confusion.

We will often speak of families α of DPSFs of the form $\alpha = (\alpha^m)_{m\in\mathbb{N}}$, where α^m is a PSF on $[m]$, such that for a family of DPSFs it holds that $\alpha^{m+1}(i+1) = \alpha^m(i)$ for all $m \in \mathbb{N}$ and $i \in [m]$. In other words, we build our families of DPSFs by prepending values to functions with smaller domains. To simplify notation, we will refer to such families of DPSFs as *normal* DPSFs. We assume that each function α^m from a family can be computed in polynomial time with respect to m. Indeed, we are particularly interested in the Borda family defined by $\alpha^m_{\mathrm{B}}(i) = m - i$.

Assignment functions. We recall that N denotes the set of agents (voters), and A denotes the set of alternatives (candidates). In this chapter we will additionally use the notion of the capacity of an alternative—for each alternative $a \in A$, its capacity cap_a denotes the maximum number of agents that can be assigned to a. A K-*assignment function* is any function $\Phi\colon N \to A$, such that $|\Phi(N)| \leq K$ (that is, it matches agents to at most K alternatives), and such that for every alternative $a \in A$ we have that $|\Phi^{-1}(a)| \leq \mathrm{cap}_a$ (i.e., the number of agents assigned to a does not exceed a's capacity cap_a).

We will also consider partial assignment functions. A partial K-assignment function is defined in the same way as a regular one, except that it may assign a null alternative, $\bot$, to some of the agents. It is convenient to think that for each agent i we have $\mathrm{pos}_i(\bot) = m$. (We recall that $\mathrm{pos}_i(a)$ denote the position of the alternative a in i's preference order.) In general, it might be the case that a partial K-assignment function cannot be extended to a regular one. This may happen, for example, if the partial assignment function uses K alternatives whose capacities sum to less than the total number of voters. However, in the context of Chamberlin–Courant and Monroe rules it is always possible to extend a partial K-assignment function to a regular one.

Given a normal DPSF α, we may consider the following two functions, each assigning a positive integer to any assignment Φ:

$$\ell_1^\alpha(\Phi) = \sum_{i=1}^{n} \alpha(\mathrm{pos}_i(\Phi(i))),$$
$$\ell_\infty^\alpha(\Phi) = \max\nolimits_{i=1}^{n} \alpha(\mathrm{pos}_i(\Phi(i))).$$

These functions are built from individual satisfaction functions, so that they can measure the quality of the assignment for the whole society. In the utilitarian framework the first one can be viewed as a *total (societal) satisfaction function*. The second one can be used as a total satisfaction functions in the egalitarian framework. We will omit the word total if no confusion may arise.

For each subset of the alternatives $S \subseteq A$ such that $|S| \leq K$, we denote as Φ_α^S the partial K-assignment that assigns agents only to the alternatives from S and such that Φ_α^S maximizes the utilitarian satisfaction $\ell_1^\alpha(\Phi_\alpha^S)$. (We introduce this notation only for the utilitarian satisfaction-based setting because it is useful to express appropriate algorithms for this case; for other settings we have hardness results only and this notation would not be useful.)

The Capacitated Disjunctive Selection of the Collective Set of Items. Let us now recall the definition of the capacitated disjunctive variant of the problem of selecting the collective set of items problem, which forms the base of our study. This problem stipulates finding an optimal K-assignment function, where the optimality is relative to one of the total satisfaction functions that we have just introduced. In this problem we additionally assume that the utilities of the agents are obtained from their rankings, by applying a positional scoring function. For the sake of the clarity and brevity of the notation we will refer to this base problem as to the ASSIGNMENT problem.

Definition 5.1. *[The Capacitated Disjunctive Selection of the Collective Set of Items.] Let α be a normal DPSF. An instance of α-*U-ASSIGNMENT *problem (i.e., of the utilitarian assignment problem) consists of a set of agents $N = [n]$, a set of alternatives $A = \{a_1, \dots a_m\}$, a preference profile V of the agents, and a sequence $(\mathrm{cap}_{a_1}, \dots, \mathrm{cap}_{a_m})$ of alternatives' capacities. We ask for an assignment function Φ such that:*

1. *$|\Phi(N)| \leq K$;*
2. *$|\Phi^{-1}(a)| \leq \mathrm{cap}_a$ for all $a \in A$; and*
3. *$\ell_1^\alpha(\Phi)$ is maximized.*

Problem α-E-ASSIGNMENT (the egalitarian version of the base problem) is defined identically except that ℓ_1^α is replaced with ℓ_∞^α.

Our two problems can be viewed as generalizations of the winner determination problem for the Monroe [210] and Chamberlin–Courant [47] multiwinner voting systems. To model the Monroe system, it suffices to set the capacity of each alternative to be $\frac{|N|}{K}$ (for simplicity, throughout the chapter we assume that K divides $|N|$[3]). We will refer to thus restricted variants of our problems as the MONROE variants. To represent the Chamberlin–Courant system, we set alternatives' capacities to $|N|$. We will refer to the so-restricted variants of our problems as CC variants.

[3]In general, this assumption is not as innocent as it may seem. Often dealing with cases there K does not divide $|N|$ requires additional insights and care. However, for our algorithms and results, the assumption simplifies notation and does not lead to obscuring any unexpected difficulties.

5.3 Hardness of Approximation

We now present our inapproximability results for the Monroe and Chamberlin–Courant rules. Specifically, we show that there are no constant-factor approximation algorithms for the egalitarian variants (However, our result for the Monroe setting is more general than the result for the Chamberlin–Courant setting; the latter is for the Borda DPSF only.).

Naturally, these inapproximability results carry over to more general settings. For example, unless P = NP, there are no polynomial-time constant-factor approximation algorithms for the general E-Assignment Problem. On the other hand, our results do not preclude good approximation algorithms for the utilitarian case and, indeed, in Section 5.4 we provide such algorithms.

Theorem 5.1. *For each normal DPSF α (where each entry is polynomially bounded in the number of alternatives) and each constant factor r, with $0 < r \leq 1$, there is no r-approximation algorithm for α-*E-Monroe *unless* P = NP*.*

Proof. Let us fix a DPSF $\alpha = (\alpha^m)_{m \in \mathbb{N}}$, where each entry α^m is polynomially bounded in the number of alternatives m. For the sake of contradiction, let us assume that for some r, $0 < r \leq 1$, there is a polynomial-time r-approximation algorithm $\mathcal{A}$ for α-E-Monroe. We will show that the existence of this algorithm implies that X3C is solvable in polynomial time.

Let I be an X3C instance with ground set $U = \{1, 2, \dots, n\}$ and collection $\mathcal{F} = \{F_1, \dots, F_m\}$ of subsets of U. Each set in $\mathcal{F}$ has cardinality three. Further, without loss of generality, we can assume that n is divisible by three and that each $i \in U$ appears in at most three sets from $\mathcal{F}$. Given I, we form an instance I_M of α-E-Monroe as follows. Let $n' = 3 \cdot (\alpha^{m+1}(1) \cdot \lceil \frac{1-r}{r} \rceil + 3)$. The set N of agents is partitioned into two subsets, N_1 and N_2. N_1 contains n agents (intuitively, corresponding to the elements of the ground set U) and N_2 contains n' agents (used to enforce certain properties of the solution). The set of alternatives A is partitioned into two subsets, A_1 and A_2. We set $A_1 = \{a_1, \dots, a_m\}$ (members of A_1 correspond to the sets in $\mathcal{F}$), and we set $A_2 = \{b_1, \dots, b_{m'}\}$, where $m' = \frac{n'}{3}$.

For each j, $1 \leq j \leq n$, we set $M_f(j) = \{a_i \mid j \in F_i\}$. For each j, $1 \leq j \leq n$, we set the preference order of the j'th agent in N_1 to be of the form

$$M_f(j) \succ A_2 \succ A_1 - M_f(j).$$

Note that by our assumptions, $|M_f(j)| \leq 3$. For each j, $1 \leq j \leq n'$, we set the preference order of the j'th agent in N_2 to be of the form

$$b_{\lceil \frac{j}{3} \rceil} \succ A_2 - \{b_{\lceil \frac{j}{3} \rceil}\} \succ A_1.$$

Note that each agent in N_2 ranks the alternatives from A_1 in positions $m'+1, \dots, m'+m$. Finally, we set the number of candidates that can be selected to be $K = \frac{n+n'}{3}$.

Now, consider the solution Φ returned by $\mathcal{A}$ on I_M. We will show that $\ell_\infty^{\alpha^{m+m'}}(\Phi) \leq r\alpha^{m+m'}(3)$ if and only if I is a *yes*-instance of X3C.

($\Leftarrow$) If there exists an exact set cover of U with sets from $\mathcal{F}$, then it is easy to construct a solution for I_M where the satisfaction of each agent is greater or equal to $r \cdot \alpha^{m+m'}(3)$. Let $I \subseteq \{1, \ldots, m\}$ be a set such that $\bigcup_{i \in I} F_i = U$ and $|I| = \frac{n}{3}$. We assign each agent j from N_1 to the alternative a_i such that (a) $i \in I$ and (b) $j \in F_i$, and we assign each agent from N_2 to her most preferred alternative. Thus, Algorithm $\mathcal{A}$ has to return an assignment with the minimal satisfaction greater or equal to $r \cdot \alpha^{m+m'}(3)$.

($\Rightarrow$) For the other direction, we first show that $r \cdot \alpha^{m+m'}(3) \geq \alpha^{m+m'}(m')$. Since DPSFs are strictly decreasing, it holds that:

$$r \cdot \alpha^{m+m'}(3) \geq r \cdot (\alpha^{m+m'}(m') + m' - 3). \tag{5.1}$$

Then, by the definition of DPSFs, it holds that:

$$\alpha^{m+m'}(m') = \alpha^{m+1}(1). \tag{5.2}$$

Using the fact that $m' = (\alpha^{m+1}(1) \cdot \lceil \frac{1-r}{r} \rceil + 3)$ and using (5.2), we can transform inequality (5.1) to obtain the following:

$$\begin{aligned}
r \cdot \alpha^{m+m'}(3) &\geq r \cdot (\alpha^{m+m'}(m') + m' - 3) \\
&= r \cdot \left(\alpha^{m+m'}(m') + (\alpha^{m+1}(1) \cdot \left\lceil \frac{1-r}{r} \right\rceil + 3) - 3 \right) \\
&\geq r \cdot \alpha^{m+m'}(m') + (1-r) \cdot \alpha^{m+1}(1) \\
&= r \cdot \alpha^{m+m'}(m') + (1-r) \cdot \alpha^{m+m'}(m') = \alpha^{m+m'}(m').
\end{aligned}$$

This means that if the minimal satisfaction of an agent is at least $r \cdot \alpha^{m+m'}(3)$, then no agent was assigned to an alternative that he or she ranked beyond position m'. If some agent j from N_1 were assigned to an alternative from A_2, then, by the pigeonhole principle, some agent from N_2 would be assigned to an alternative from A_1. However, each agent in N_2 ranks the alternatives from A_1 beyond position m' and thus such an assignment is impossible. In consequence, it must be that each agent in j was assigned to an alternative that corresponds to a set F_i in $\mathcal{F}$ that contains j. Such an assignment directly leads to a solution for I. $\square$

Let us now move on to the case of E-CC family of problems. Unfortunately, in this case our inapproximability argument holds for the case of Borda DPSF only (though we believe that it can be adapted to other DPSFs as well). Further, in the previous theorem we have shown that existence of a constant-factor approximation algorithm implies that NP collapses to P. In the following theorem we will show a seemingly weaker collapse of W[2] to FPT.

To prove hardness of approximation for α_{B}-E-CC, we first prove the following simple lemma.

Lemma 5.2. *Let K, p, l be three positive integers and let X be a set of cardinality lpK. There exists a family $\mathcal{S} = \{S_1, \dots, S_{\binom{lK}{K}}\}$ of pK-element subsets of X such that for each K-element subset B of X, there is a set $S_i \in \mathcal{S}$ such that $B \subseteq S_i$.*

Proof. Set $X' = [lK]$ and let Y' be a family of all K-element subsets of X'. Replace each element i of X' with p new elements (at the same time replacing i with the same p elements within each set in Y' that contains i). As a result we obtain two new sets, X and Y, that satisfy the statement of the theorem (up to the renaming of the elements). $\square$

Theorem 5.3. *Let α_{B}^m be the Borda DPSF ($\alpha_{\mathrm{B}}^m(i) = m - i$). For each constant factor r, $0 < r \leq 1$, there is no r-approximation algorithm for α_{B}^m-*E-CC *unless* FPT = W[2].

Proof. For the sake of contradiction, let us assume that there is some constant r, $0 < r \leq 1$, and a polynomial-time r-approximation algorithm $\mathcal{A}$ for α_{B}^m-E-CC. We will show that the existence of this algorithm implies that SET-COVER is fixed-parameter tractable for the parameter K (since SET-COVER is known to be W[2]-complete for this parameter, this will imply FPT = W[2]).

Let I be an instance of SET-COVER with ground set $U = [n]$ and family $\mathcal{F} = \{F_1, F_2, \dots, F_m\}$ of subsets of U. Given I, we build an instance I_{CC} of α_{B}^m-E-CC as follows. The set of agents N consists of n subsets of agents, $N_1, \dots, N_n$, where each group N_i contains exactly $n' = \binom{\lceil \frac{2}{r} \rceil K}{K}$ agents. Intuitively, for each i, $1 \leq i \leq n$, the agents in the set N_i correspond to the element i in U. The set of alternatives A is partitioned into two subsets, A_1 and A_2, such that: (1) $A_1 = \{a_1, \dots, a_m\}$ is a set of alternatives corresponding to the sets from the family $\mathcal{F}$, and (2) A_2, $|A_2| = \lceil \frac{2}{r} \rceil \left\lceil \frac{m(1+r)}{K} \right\rceil K$, is a set of dummy alternatives needed for our construction. We set $m' = |A| = m + |A_2|$.

Before we describe the preference orders of the agents in N, we form a family $R = \{r_1, \dots, r_{n'}\}$ of preference orders over A_2 that satisfies the following condition: For each K-element subset B of A_2, there exists r_j in R such that all members of B are ranked among the bottom $\left\lceil \frac{m(1+r)}{K} \right\rceil K$ positions in r_j. By Lemma 5.2, such a construction is possible (it suffices to take $l = \lceil \frac{2}{r} \rceil$ and $p = \left\lceil \frac{m(1+r)}{K} \right\rceil$); further, the proof of the lemma provides an algorithmic way to construct R.

We form the preference orders of the agents as follows. For each i, $1 \leq i \leq n$, set $M_f(i) = \{a_t \mid i \in F_t\}$. For each i, $1 \leq i \leq n$, and each j, $1 \leq j \leq n'$, the j'th agent from N_i has preference order of the form:

$$M_f(i) \succ r_j \succ A_1 - M_f(i)$$

(we pick any arbitrary, polynomial-time computable order of candidates within $M_f(i)$ and $M_l(i)$).

Let Φ be an assignment computed by $\mathcal{A}$ on I_M. We will show that $\ell_{\infty}^{\alpha_{\mathrm{B}}^{m'}}(\Phi) \geq r \cdot (m' - m)$ if and only if I is a *yes*-instance of Set-Cover.

($\Leftarrow$) If there exists a solution for I (i.e., a cover of U with K sets from $\mathcal{F}$), then we can easily show an assignment where each agent is assigned to an alternative that he or she ranks among the top m positions (namely, for each j, $1 \leq j \leq n$, we assign all the agents from the set N_j to the alternative $a_i \in A_1$ such that $j \in F_i$ and F_i belongs to the alleged K-element cover of U). Under this assignment, the least satisfied agent's satisfaction is at least $m' - m$ and, thus, $\mathcal{A}$ has to return an assignment Φ where $\ell_{\infty}^{\alpha_{\mathrm{B}}^{m'}}(\Phi) \geq r \cdot (m' - m)$.

($\Rightarrow$) Let us now consider the opposite direction. We assume that $\mathcal{A}$ found an assignment Φ such that $\ell_{\infty}^{\alpha_{\mathrm{B}}^{m}}(\Phi) \geq r \cdot (m' - m)$ and we will show that I is a *yes*-instance of Set-Cover. We claim that for each i, $1 \leq i \leq n$, at least one agent j in N_i were assigned to an alternative from A_1. If all the agents in N_i were assigned to alternatives from A_2, then, by the construction of R, at least one of them would have been assigned to an alternative that he or she ranks at a position greater than $|A_2| - \left\lceil \frac{m(1+r)}{K} \right\rceil K = \left\lceil \frac{2}{r} \right\rceil \left\lceil \frac{m(1+r)}{K} \right\rceil K - \left\lceil \frac{m(1+r)}{K} \right\rceil K$. For $x = \left\lceil \frac{m(1+r)}{K} \right\rceil K$ we have:

$$\left\lceil \frac{2}{r} \right\rceil x - x \geq m' - m'r + mr$$

(we skip the straightforward calculation) and, thus, this agent would have been assigned to an alternative that he or she ranks at a position greater than $m' - m'r + mr$. As a consequence, this agent's satisfaction would be lower than $(m' - m)r$. Similarly, no agent from N_i can be assigned to an alternative from $M_l(i)$. Thus, for each i, $1 \leq i \leq n$, there exists at least one agent $j \in N_i$ that is assigned to an alternative from $M_f(i)$. In consequence, the covering subfamily of $\mathcal{F}$ consists simply of those sets F_k, for which some agent is assigned to alternative $a_k \in A_1$.

The presented construction gives the exact algorithm for Set-Cover problem running in time $f(K)(n+m)^{O(1)}$, where $f(K)$ is polynomial in $\binom{\lceil \frac{2}{r} \rceil}{K}$. The existence of such an algorithm means that Set-Cover is in FPT. On the other hand, we know that Set-Cover is W[2]-complete, and thus if $\mathcal{A}$ existed then FPT = W[2] would hold. $\square$

5.4 Algorithms for the Utilitarian Cases

We now turn to approximation algorithms for the Monroe and Chamberlin–Courant multiwinner voting rules in the utilitarian framework. Indeed, in this case it is possible to obtain high-quality approximation results. In particular, we show the first nontrivial (randomized) approximation algorithm for α_{B}-U-Monroe. We show that for each $\epsilon > 0$ we can provide a randomized polynomial-time algorithm that achieves $0.715 - \epsilon$ approximation ratio; the algorithm usually gives even better approximation guarantees. For the case of arbitrarily selected DPSF we show a $(1-e^{-1})$-approximation algorithm. Finally, by applying the results from Chapter 4 to the context of finding proportional representation, we show the first polynomial-time approximation scheme (PTAS) for α_{B}-U-CC. These results stand in sharp contrast to those from the previous section, where we have shown that approximation is hard for essentially all remaining variants of the problem.

The core difficulty in solving α-Monroe/CC-Assignment problems lays in selecting the alternatives that should be assigned to the agents. Given a preference profile and a set S of up to K alternatives, using a standard network-flow argument, it is easy to find a (possibly partial) optimal assignment Φ_{α}^{S} of the agents to the alternatives from S.

Proposition 5.4 (Implicit in the paper of Betzler et al. [27]). *Let α be a normal DPSF, N be a set of agents, A be a set of alternatives (together with their capacities; perhaps represented implicitly as for the case of the Monroe and Chamberlin–Courant rules), V be a preference profile of N over A, and S a K-element subset of A (where K divides $|N|$). Then there is a polynomial-time algorithm that computes a (possibly partial) optimal assignment Φ_{α}^{S} of the agents to the alternatives from S.*

Note that for the case of the Chamberlin–Courant rule the algorithm from the above proposition can be greatly simplified: To each voter we assign the candidate that he or she ranks highest among those from S. For the case of Monroe, unfortunately, we need the expensive network-flow-based approach. Nonetheless, Proposition 5.4 allows us to focus on the issue of selecting the winning alternatives and not on the issue of matching them to the agents.

Below we describe our algorithms for α_{B}-U-Monroe and for α_{B}-U-CC. Formally speaking, every approximation algorithm for α_{B}-U-Monroe also gives feasible results for α_{B}-U-CC. However, some of our algorithms are particularly well-suited for both problems and some are tailored to only one of them. Thus, for each algorithm we clearly indicate if it is meant only for the case of Monroe, only for the case of CC, or if it naturally works for both systems.

5.4.1 Algorithm A (Monroe)

Perhaps the most natural approach to solve α_{B}-U-Monroe is to build a solution iteratively: In each step we pick some not-yet-assigned alternative a_i (using some criterion) and assign it to those $\lceil \frac{N}{K} \rceil$ agents that (a) are not assigned to any other alternative yet, and (b) whose satisfaction of being matched with a_i is maximal. It turns out that this idea, implemented formally as Algorithm A (see pseudo code in Figure 5.1), works very well in many cases. We provide a lower bound on the total satisfaction it guarantees in the next lemma. Let us recall that $H_k = \sum_{i=1}^{k} \frac{1}{i}$ is the k'th *harmonic number* and that $H_k = \Theta(\log k)$.

```
   Notation: $\Phi \leftarrow$ a map defining a partial assignment, iteratively built by the
             algorithm.
        $\Phi^{\leftarrow} \leftarrow$ the set of agents for which the assignment is already defined.
        $\Phi^{\rightarrow} \leftarrow$ the set of alternatives already used in the assignment.
1  if $K \leq 2$ then
2      compute the optimal solution using an algorithm of Betzler et al. [27] and return.
3  $\Phi = \{\}$
4  for $i \leftarrow 1$ to $K$ do
5      $score \leftarrow \{\}$
6      $bests \leftarrow \{\}$
7      foreach $a_i \in A \setminus \Phi^{\rightarrow}$ do
8          $agents \leftarrow$ sort $N \setminus \Phi^{\leftarrow}$ so that if agent $j$ precedes agent $k$ then
           $pos_j(a_i) \leq pos_k(a_i)$
9          $bests[a_i] \leftarrow$ chose first $\lceil \frac{N}{K} \rceil$ elements from $agents$
10         $score[a_i] \leftarrow \sum_{j \in bests[a_i]} (m - pos_j(a_i))$
11     $a_{best} \leftarrow \mathrm{argmax}_{a \in A \setminus \Phi^{\rightarrow}} score[a]$
12     foreach $j \in bests[a_{best}]$ do
13         $\Phi[j] \leftarrow a_{best}$
```

Figure 5.1: The pseudocode for Algorithm A.

Lemma 5.5. *Algorithm A is a polynomial-time* $(1 - \frac{K-1}{2(m-1)} - \frac{H_K}{K})$*-approximation algorithm for* α_{B}-U-Monroe.

Proof. Our algorithm explicitly computes an optimal solution when $K \leq 2$ so we assume that $K \geq 3$. Let us consider the situation in the algorithm after the i'th iteration of the outer loop (we have $i = 0$ if no iteration has been executed yet). So far, the algorithm has picked i alternatives and assigned them to $i\frac{n}{K}$ agents (recall that for simplicity we assume that K divides n evenly). Hence, each agent has $\lceil \frac{m-i}{K-i} \rceil$ unassigned alternatives among her $i + \lceil \frac{m-i}{K-i} \rceil$ top-ranked alternatives. By pigeonhole principle, this means that there is an unassigned alternative a_ℓ who is ranked among top $i + \lceil \frac{m-i}{K-i} \rceil$ positions by at least $\frac{n}{K}$ agents. To see this, note that there are $(n - i\frac{n}{K})\lceil \frac{m-i}{K-i} \rceil$ slots for unassigned alternatives among the top $i + \lceil \frac{m-i}{K-i} \rceil$ positions in the preference orders of unassigned agents, and that there are $m - i$ unassigned alternatives. As a result, there must be an alternative a_ℓ for whom the number of agents that rank him or her among the top $i + \lceil \frac{m-i}{K-i} \rceil$ positions is at least:

$$\frac{1}{m-i}\left((n - i\frac{n}{K})\left\lceil \frac{m-i}{K-i} \right\rceil\right) \geq \frac{n}{m-i}\left(\frac{K-i}{K}\right)\left(\frac{m-i}{K-i}\right) = \frac{n}{K}.$$

In consequence, the $\lceil \frac{n}{K} \rceil$ agents assigned in the next step of the algorithm will have the total satisfaction at least $\lceil \frac{n}{K} \rceil \cdot (m - i - \lceil \frac{m-i}{K-i} \rceil)$. Thus, summing over the K iterations, the total satisfaction guaranteed by the assignment Φ computed by Algorithm Ais at least the following value: (to derive the fifth line from the fourth one we note that $K(H_K - 1) - H_K \geq 0$ when $K \geq 3$):

$$\begin{aligned}
\ell_1^{\alpha_b}(\Phi) &\geq \sum_{i=0}^{K-1} \frac{n}{K} \cdot \left(m - i - \lceil \frac{m-i}{K-i} \rceil\right) \\
&\geq \sum_{i=0}^{K-1} \frac{n}{K} \cdot \left(m - i - \frac{m-i}{K-i} - 1\right) \\
&= \sum_{i=1}^{K} \frac{n}{K} \cdot \left(m - i - \frac{m-1}{K-i+1} + \frac{i-2}{K-i+1}\right) \\
&= \frac{n}{K}\left(\frac{K(2m-K-1)}{2} - (m-1)H_K + K(H_K-1) - H_K\right) \\
&\geq \frac{n}{K}\left(\frac{K(2m-K-1)}{2} - (m-1)H_K\right) \\
&\geq (m-1)n\left(1 - \frac{K-1}{2(m-1)} - \frac{H_K}{K}\right)
\end{aligned}$$

If each agent were assigned to her top alternative, the total satisfaction would be equal to $(m-1)n$. Thus we get the following bound:

$$\frac{\ell_1^{\alpha_{\mathrm{B}}}(\Phi)}{\mathrm{OPT}} \leq 1 - \frac{K-1}{2(m-1)} - \frac{H_K}{K}.$$

This completes the proof. □

Note that in the above proof we measure the quality of our assignment against, a perhaps-impossible, perfect solution, where each agent is assigned to her top alternative. This means that for relatively large m and K, and small $\frac{K}{m}$ ratio, the algorithm can achieve a close-to-ideal solution irrespective of the voters' preference orders. We believe that this is an argument in favor of using Monroe's system in multiwinner elections. On the flip side, to obtain a better approximation ratio, we would have to use a more involved bound on the quality of the optimal solution. To see that this is the case, form an instance I of α_{B}-U-Monroe with n agents and m alternatives, where all the agents have the same preference order, and where we seek to elect K candidates (and where K divides n). It is easy to see that each solution that assigns the K universally top-ranked alternatives to the agents is optimal. Thus the total satisfaction of the agents in the optimal solution is:

$$\frac{n}{K}\left((m-1) + \cdots + (m-K)\right) = \frac{n}{K}\left(\frac{K(2m-K-1)}{2}\right) = n(m-1)\left(1 - \frac{K-1}{2(m-1)}\right).$$

By taking large enough m and K (even for a fixed value of $\frac{m}{K}$), the fraction $1 - \frac{K-1}{2(m-1)}$ can be arbitrarily close to the approximation ratio of our algorithm (the reasoning here is in the spirit of the idea of identifying maximally robust elections, as studied by Shiryaev, Yu, and Elkind [268]).

For small values of K, it is possible that the $\frac{H_K}{K}$ part of our approximation ratio would dominate the $\frac{K-1}{2(m-1)}$ part. In such cases we can use the result of Betzler et al. [27], who showed that for each fixed constant K, α_{B}-U-Monroe can be solved in polynomial time. Thus, for the finite number of cases where $\frac{H_K}{K}$ is too large, we can solve the problem optimally using their algorithm. In consequence, the quality of the solution produced by Algorithm A most strongly depends on the ratio $\frac{K-1}{2(m-1)}$. In most cases we can expect it to be small (for example, in Polish parliamentary elections $K = 460$ and $m \approx 6000$; in this case the greedy algorithm's approximation ratio is about 0.96).

Our algorithm has one more great advantage: Since it focuses on the top parts of voters' preference orders, it can achieve very good results even if the voters submit truncated ballots (that is, if they rank some of their top alternatives only). Below we present the formal analysis of the algorithm's approximation ratio for this case. Unfortunately, we did not obtain a closed form formula and, instead, we present the guaranteed approximation ratio as a sum. We also present the relation between the fraction of the top alternatives ranked by each of the voters and the approximation ratio for few values of m and K in Figure 5.2.

Proposition 5.6. *Let P be the number of top positions in the agents' preference orders that are known by the algorithm. In this case Algorithm A is a polynomial-time r-approximation algorithm for α_{B}-U-Monroe, where:*

$$r = \sum_{i=0}^{K-1} \frac{1}{n(m-1)} \max(r(i), 0)$$

and

$$r(i) = \begin{cases} \frac{n}{K}(m - i - \frac{m-i}{K-i}) & \text{if } \left(i + \frac{m-i}{K-i}\right) \leq P, \\ \frac{n}{K}\frac{(K-i)(m-i)}{4} & \text{if } \left(i + \frac{m-i}{K-i}\right) > P \text{ and } (2P - m) \geq i \geq (K-2), \\ \frac{n}{K}\frac{(m-P)(K-i)(P-i)}{m-i} & \text{otherwise.} \end{cases}$$

Proof. We use the same approach as in the proof of Lemma 5.5, except that we adjust our estimates of voters' satisfaction. Consider a situation after some i'th iteration of the algorithm's outer loop ($i = 0$ if we are before the first iteration). If $i + \frac{m-i}{K-i} \leq P$, then we can use the same lower bound for the satisfaction of the agents assigned in the $(i+1)$'th iteration as in the proof of Lemma 5.5. That is, the agents assigned in the $(i+1)$'th iteration will have satisfaction at least $r_1(i) = \frac{n}{K} \cdot (m - i - \frac{m-i}{K-i})$.

For the case where $i + \frac{m-i}{K-i} > P$, the bound from Lemma 5.5 does not hold, but we can use a similar approach to find a different one. Let $P_x \leq P$ be some positive integer. We are interested in the number x of not-yet assigned agents who rank some not-yet-selected alternative among their top P_x positions (after the i'th iteration). Similarly as in the proof of Lemma 5.5, using the pigeonhole principle we note that:

$$x \geq \frac{1}{m-i}\left(n - i\frac{n}{K}\right)(P_x - i) = \frac{n}{K} \cdot \frac{(K-i)(P_x-i)}{m-i}.$$

Thus, the satisfaction of the agents assigned in the $(i+1)$'th iteration is at least:

$$\min\left(x, \frac{n}{K}\right)(m - P_x) = \frac{n}{K} \cdot (m - P_x) \min\left(\frac{(K-i)(P_x-i)}{m-i}, 1\right). \tag{5.3}$$

The case $\frac{(K-i)(P_x-i)}{m-i} \geq 1$ (or, equivalently, $i + \frac{m-i}{K-i} \leq P_x$) implies that $i + \frac{m-i}{K-i} \leq P$ and for this case we lower-bound agents' satisfaction by $r_1(i)$. For the case where $\frac{(K-i)(P_x-i)}{m-i} \leq 1$, i.e. where $i + \frac{m-i}{K-i} \geq P_x$, equation (5.3) simplifies to:

$$\frac{n}{K} \cdot (m - P_x) \cdot \frac{(K-i)(P_x-i)}{m-i}. \tag{5.4}$$

We use this estimate for the satisfaction of the agents assigned in the $(i+1)$'th iteration for the cases where (a) $i + \frac{m-i}{K-i} \geq \frac{m+i}{2}$ and (b) $\frac{m+i}{2} \leq P$ (or, equivalently, $(2P - m) \geq i \geq (K-2)$). In this case we estimate (5.4) as follows:

$$\begin{aligned}\frac{n}{K} \cdot (m - P_x) \cdot \frac{(K-i)(P_x-i)}{m-i} &\geq \frac{n}{K} \cdot (m - \frac{m+i}{2}) \cdot \frac{(K-i)(\frac{m+i}{2} - i)}{m-i} \\ &= \frac{n}{K} \cdot \frac{(K-i)(m-i)^2}{4(m-i)} = \frac{n}{K} \cdot \frac{(K-i)(m-i)}{4}.\end{aligned}$$

For the remaining cases, we set $P_x = P$ and (5.4) becomes:

$$\frac{n}{K} \cdot \frac{(m-P)(K-i)(P-i)}{m-i}.$$

Naturally, we replace our estimates by 0 whenever they become negative.

To complete the proof, it suffices, as in the proof of Lemma 5.5, to note that $(m-1)n$ is an upper bound on the satisfaction achieved by the optimal solution. □

For example, for the case of Polish parliamentary elections ($K = 460$ and $m = 6000$), to achieve 90% of voters' optimal satisfaction, each voter would have to rank about 8.7% of the candidates.

Our results show that for most settings there is very little reason to ask the agents to rank all the alternatives. Using Proposition 5.6, election designers can estimate how many alternatives the agents should rank to obtain a particular level of satisfaction. Since computing preference orders can be expensive for the agents, this way they can save a large amount of effort.

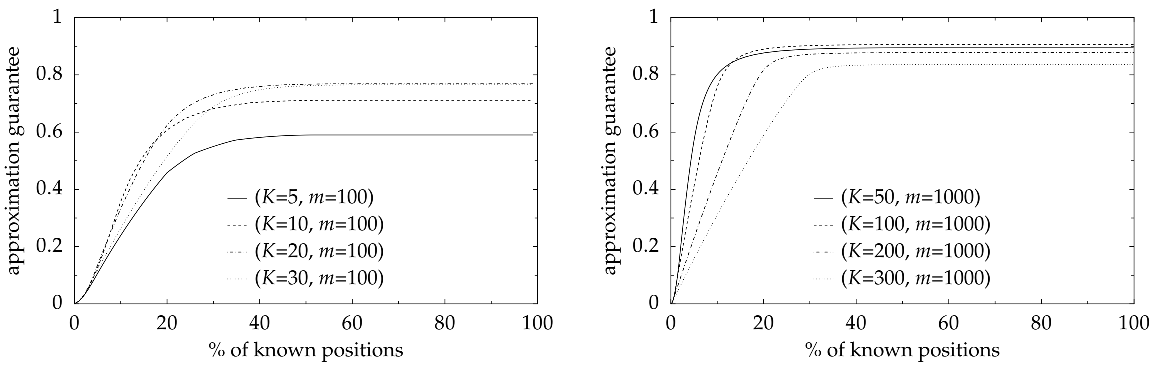

Figure 5.2: The relation between the percentage of the known positions and the approximation ratio of Algorithm A for α_{B}-U-MONROE.

5.4.2 Algorithm B (Monroe)

There are simple ways in which we can improve the quality of the assignments produced by Algorithm A. For example, our Algorithm B first runs Algorithm A and then, using Proposition 5.4, optimally reassigns the alternatives to the voters. As shown in Section 5.5, this very simple trick turns out to noticeably improve the results of the algorithm in practice (and, of course, the theoretical approximation guarantees of Algorithm A carry over to Algorithm B).

5.4.3 Algorithm C (Monroe, CC)

Algorithm C is a further heuristic improvement over Algorithm B. This time the idea is that instead of keeping only one partial function Φ that is iteratively extended up to the full assignment, we keep a list of up to d partial assignment functions, where d is a parameter of the algorithm. At each iteration, for each assignment function Φ among the d stored ones and for each alternative a that does not yet have agents assigned to by this Φ, we compute an optimal extension of this Φ that assigns agents to a. As a result we obtain possibly more than d (partial) assignment functions. For the next iteration we keep those d of them that give highest satisfaction.

We provide pseudocode for Algorithm C in Figure 5.3. If we take $d = 1$, we obtain Algorithm B. If we also disregard the last two lines prior to returning the solution, we obtain Algorithm A.

Algorithm C can also be adapted for the Chamberlin–Courant rule. The only difference concerns creating the assignment functions: we replace the contents of the first foreach loop with the following code:

```
Notation: We use the same notation as in Algorithm A;
        $Par \leftarrow$ a list of partial representation functions
$Par = []$
$Par$.push({})
for $i \leftarrow 1$ to $K$ do
    $newPar = []$
    for $\Phi \in Par$ do
        $bests \leftarrow \{\}$
        foreach $a_i \in A \setminus \Phi^{\rightarrow}$ do
            $agents \leftarrow$ sort $N \setminus \Phi^{\leftarrow}$ (agent $j$ precedes agent $k$ implies that
            $pos_j(a_i) \leq pos_k(a_i)$)
            $bests[a_i] \leftarrow$ chose first $\lceil \frac{N}{K} \rceil$ elements of $agents$
            $\Phi' \leftarrow \Phi$
            foreach $j \in bests[a_i]$ do
                $\Phi'[j] \leftarrow a_i$
            $newPar$.push($\Phi'$)
        sort $newPar$ according to descending order of the total satisfaction of the
        assigned agents
        $Par \leftarrow$ chose first $d$ elements of $newPar$
for $\Phi \in Par$ do
    $\Phi \leftarrow$ compute the optimal representative function using an algorithm of Betzler
    et al. [27] for the set of winners $\Phi^{\rightarrow}$
return the best representative function from $Par$
```

Figure 5.3: The pseudocode for Algorithm C.

```
foreach $a_i \in A \setminus \Phi^{\rightarrow}$ do
    $\Phi' \leftarrow \Phi$
    foreach $j \in N$ do
        if agent $j$ prefers $a_i$ to $\Phi'(j)$ then
            $\Phi'(j) \leftarrow a_i$
    $newPar$.push($\Phi'$)
```

Note that for the case of the Chamberlin–Courant rule Algorithm C can also be seen as a generalization of Algorithm GM that we will discuss later in Section 5.4.5.

5.4.4 Algorithm R (Monroe, CC)

Algorithms A, B and C achieve very high approximation ratios for the cases where K is small relative to m. For the remaining cases, where K and m are comparable, we can use a sampling-based randomized algorithm (denoted as Algorithm R) described below. We focus on the case of Monroe and we will briefly mention the case of CC at the end.

The idea of this algorithm is to randomly pick K alternatives and match them optimally to the agents, using Proposition 5.4. Naturally, such an algorithm might be very unlucky and pick K alternatives that all of the agents rank low. Yet, if K is comparable to m then it is likely that such a random sample would include a large chunk of some optimal solution. In the lemma below, we asses the expected satisfaction obtained with a single sampling step (relative to the satisfaction given by the optimal solution) and the probability that a single sampling step gives satisfaction close to the expected one. Naturally, in practice one should try several sampling steps and pick the one with the highest satisfaction.

Lemma 5.7. *A single sampling step of the randomized algorithm for α_{B}-U-Monroe achieves expected approximation ratio of $\frac{1}{2}(1+\frac{K}{m}-\frac{K^2}{m^2-m}+\frac{K^3}{m^3-m^2})$. Let p_ϵ denote the probability that the relative difference between the expected total satisfaction and the obtained total satisfaction is higher than ϵ. Then we have $p_\epsilon \leq \frac{1}{1+\epsilon}$.*

Proof. Let $N=[n]$ be the set of agents, $A=\{a_1,\ldots,a_m\}$ be the set of alternatives, and V be the preference profile of the agents. Let us fix some optimal solution Φ_{opt} and let A_{opt} be the set of alternatives assigned to the agents in this solution. For each $a_i \in A_{\mathrm{opt}}$, we write $\mathrm{sat}(a_i)$ to denote the total satisfaction of the agents assigned to a_i in Φ_{opt}. Naturally, we have $\sum_{a\in A_{\mathrm{opt}}}\mathrm{sat}(a)=\mathrm{OPT}$. In a single sampling step, we choose uniformly at random a K-element subset B of A. Then, we form a solution Φ_B by matching the alternatives in B optimally to the agents (via Proposition 5.4). We write K_{opt} to denote the random variable equal to $|A_{\mathrm{opt}}\cap B|$, the number of sampled alternatives that belong to A_{opt}. We define $p_i=\Pr(K_{\mathrm{opt}}=i)$. For each $j\in\{1,\ldots,K\}$, we write X_j to denote the random variable equal to the total satisfaction of the agents assigned to the j'th alternative from the sample. We claim that for each i, $0\leq i\leq K$, it holds that:

$$\mathbb{E}\left(\sum_{j=1}^{K}X_j\;\middle|\;K_{\mathrm{opt}}=i\right)\geq\frac{i}{K}\mathrm{OPT}+\frac{m-i-1}{2}\cdot\left(n-i\frac{n}{K}\right).$$

Why is this so? Given a sample B that contains i members of A_{opt}, our algorithm's solution is at least as good as a solution that matches the alternatives from $B\cap A_{\mathrm{opt}}$ in the same way as Φ_{opt}, and the alternatives from $B-A_{\mathrm{opt}}$ in a random manner. Since $K_{\mathrm{opt}}=i$ and each $a_j\in A_{\mathrm{opt}}$ has equal probability of being in the sample, it is easy to see that the expected value of $\sum_{a_j\in B\cap A_{\mathrm{opt}}}\mathrm{sat}(a_j)$ is $\frac{i}{K}\mathrm{OPT}$. After we allocate the agents from $B\cap A_{\mathrm{opt}}$, each of the remaining, unassigned agents has $m-i$ positions in her preference order where he ranks the agents from $A-A_{\mathrm{opt}}$. For each unassigned agents, the average score value associated with these positions is at least $\frac{m-i-1}{2}$ (this is so, because in the worst case the agent could rank the alternatives from $B\cap A_{\mathrm{opt}}$ in the top i positions). There are $(n-i\frac{n}{K})$ such not yet assigned agents and so the expected total satisfaction from assigning them randomly to the alternatives is $\frac{m-i-1}{2}\cdot(n-i\frac{n}{K})$. This proves our bound on the expected satisfaction of a solution yielded by optimally matching a random sample of K alternatives.

Since OPT is upper bounded by $(m-1)n$ (consider a possibly-nonexistent solution where every agent is assigned to his or her top preference), we get that:

$$\mathbb{E}\left(\sum_{j=1}^{K} X_j | K_{\text{opt}} = i\right) \geq \frac{i}{K}\text{OPT} + \frac{m-i-1}{2(m-1)} \cdot \left(1 - \frac{i}{K}\right)\text{OPT}.$$

We can compute the unconditional expected satisfaction of Φ_B as follows:

$$\begin{aligned}\mathbb{E}\left(\sum_{j=1}^{K} X_j\right) &= \sum_{i=0}^{K} p_i\,\mathbb{E}\left(\sum_{j=1}^{K} X_j | K_{\text{opt}} = i\right) \\ &\geq \sum_{i=0}^{K} p_i \left(\frac{i}{K}\text{OPT} + \frac{m-i-1}{2(m-1)} \cdot \left(1 - \frac{i}{K}\right)\text{OPT}\right).\end{aligned}$$

Since $\sum_{i=1}^{K} p_i \cdot i$ is the expected number of the alternatives in A_{opt}, we have that $\sum_{i=1}^{K} p_i \cdot i = \frac{K^2}{m}$ (one can think of summing the expected values of K indicator random variables; one for each element of A_{opt}, taking the value 1 if a given alternative is selected and taking the value 0 otherwise). Further, from the generalized mean inequality we obtain $\sum_{i=1}^{K} p_i \cdot i^2 \geq \left(\frac{K^2}{m}\right)^2$. In consequence, through routine calculation, we get that:

$$\begin{aligned}\mathbb{E}\left(\sum_{j=1}^{K} X_j\right) &\geq \left(\frac{K}{m}\text{OPT} + \frac{m^2-K^2-m}{2m(m-1)} \cdot \left(1 - \frac{K}{m}\right)\text{OPT}\right) \\ &= \frac{\text{OPT}}{2}\left(1 + \frac{K}{m} - \frac{K^2}{m^2-m} + \frac{K^3}{m^3-m^2}\right).\end{aligned}$$

It remains to assess the probability that the total satisfaction obtained through Φ_B is close to its expected value. We use the following relation:

$$p_\epsilon \cdot \left(\mathbb{E}\left(\sum_{j=1}^{K} X_j\right) - \epsilon\,\mathbb{E}\left(\sum_{j=1}^{K} X_j\right)\right) + (1-p_\epsilon) \cdot OPT \geq \mathbb{E}\left(\sum_{j=1}^{K} X_j\right).$$

Since $\mathbb{E}\left(\sum_{j=1}^{K} X_j\right) \geq \frac{\text{OPT}}{2}$, we get:

$$p_\epsilon \cdot \left(\mathbb{E}\left(\sum_{j=1}^{K} X_j\right) - \epsilon\,\mathbb{E}\left(\sum_{j=1}^{K} X_j\right)\right) + (1-p_\epsilon) \cdot 2\,\mathbb{E}\left(\sum_{j=1}^{K} X_j\right) \geq \mathbb{E}\left(\sum_{j=1}^{K} X_j\right).$$

From which we get that $p_\epsilon \leq \frac{1}{1+\epsilon}$. This completes the proof. □

In the next theorem we will see that to have a high chance of obtaining a high quality assignment, we need to repeat the sampling step many times. Thus, for practical purposes, by Algorithm R we mean an algorithm that repeats the sampling process a given number of times (this parameter is given as input) and returns the best solution found (the assignment is created using Proposition 5.4).

The threshold for $\frac{K}{m}$, where the sampling step is (in expectation) better than the Algorithm A is about 0.57. Thus, by combining the two algorithms, we can guarantee an expected approximation ratio of $0.715 - \epsilon$, for each fixed constant ϵ. The pseudo-code of the combination of the two algorithms (Algorithm AR) is presented in Figure 5.4.

Theorem 5.8. *For each fixed ϵ, Algorithm AR provides a $(0.715 - \epsilon)$-approximate solution for the problem α_{B}-U-Monroe with probability λ in time polynomial with respect to the input instance size and $-\log(1 - \lambda)$.*

Proof. Let ϵ be a fixed constant. We are given an instance I of α_{B}-U-Monroe. If $m \leq 1 + \frac{2}{\epsilon}$, we solve I using a brute-force algorithm (note that in this case the number of alternatives is at most a fixed constant). Similarly, if $\frac{H_K}{K} \geq \frac{\epsilon}{2}$ then we use the exact algorithm of Betzler et al. [27] for a fixed value of K (note that in this case K is no greater than a certain fixed constant).

On the other hand, if neither of the above conditions hold, we try both Algorithm A and a number of runs of the sampling-based algorithm. It is easy to check through routine calculation that if $\frac{H_K}{K} \leq \frac{\epsilon}{2}$ and $m > 1 + \frac{2}{\epsilon}$ then Algorithm A achieves approximation ratio no worse than $(1 - \frac{K}{2m} - \epsilon)$. We run the sampling-based algorithm $-\log(1 - \lambda)\frac{2+\epsilon}{\epsilon}$ times. The probability that a single run fails to find a solution with approximation ratio at least $\frac{1}{2}(1 + \frac{K}{m} - \frac{K^2}{m^2 - m} + \frac{K^3}{m^3 - m^2}) - \frac{\epsilon}{2}$ is $p_{\frac{\epsilon}{2}} \leq \frac{2}{2+\epsilon}$. Thus, the probability that at least one run will find a solution with at least this approximation ratio is at least:

$$1 - p_{\frac{\epsilon}{2}}^{-\log(1-\lambda)\frac{2+\epsilon}{\epsilon}} = 1 - \left(\frac{2}{2+\epsilon}\right)^{\frac{2+\epsilon}{2} \cdot -\log(1-\lambda)} \geq 1 - \exp\left(\log(1-\lambda)\right) = \lambda.$$

Since $m \leq 1 + \frac{2}{\epsilon}$, by routine calculation we see that the sampling-based algorithm with probability λ finds a solution with approximation ratio at least $\frac{1}{2}(1 + \frac{K}{m} - \frac{K^2}{m^2} + \frac{K^3}{m^3}) - \epsilon$. By solving the equality:

$$\frac{1}{2}\left(1 + \frac{K}{m} - \frac{K^2}{m^2} + \frac{K^3}{m^3}\right) = 1 - \frac{K}{2m}$$

we can find the value of $\frac{K}{m}$ for which the two algorithms give the same approximation ratio. By substituting $x = \frac{K}{m}$ we get equality $1 + x - x^2 + x^3 = 2 - x$. One can calculate that this equality has a single solution within $\langle 0, 1\rangle$ and that this solution is $x \approx 0.57$. For this x both algorithms guarantee approximation ratio of $0.715 - \epsilon$. For $x < 0.57$ the deterministic algorithm guarantees a better approximation ratio and for $x > 0.57$, the randomized algorithm does better. □

```
  Notation: We use the same notation as in Algorithm 5.1; W(·) denotes Lambert's
            W-Function.
  Parameters: $\lambda \leftarrow$ required probability of achieving the approximation ratio equal
              $0.715 - \epsilon$
1 if $\frac{H_K}{K} \geq \frac{\epsilon}{2}$ then
2     compute the optimal solution using an algorithm of Betzler et al. [27] and return.
3 if $m \leq 1 + \frac{2}{\epsilon}$ then
4     compute the optimal solution using a simple brute force algorithm and return.
5 $\Phi_1 \leftarrow$ solution returned by Algorithm A
6 $\Phi_2 \leftarrow$ run the sampling-based algorithm $-\log(1-\lambda) \cdot \frac{2+\epsilon}{\epsilon}$ times; select the
  assignment of the best quality
7 return the better assignment among $\Phi_1$ and $\Phi_2$
```

Figure 5.4: Algorithm AR—combination of Algorithms A and R.

Let us now consider the case of CC. It is just as natural to try a sampling-based approach for solving α_{B}-U-CC, as we did for the Monroe variant. Indeed, as recently (and independently) observed by Oren [229], this leads to a randomized algorithm with expected approximation ratio of $(1 - \frac{1}{K+1})(1 + \frac{1}{m})$. However, since we will later see an effective, deterministic, polynomial-time approximation scheme for α_{B}-U-CC, there is little reason to explore the sampling based approach.

5.4.5 Algorithm GM (Monroe, CC)

Algorithm GM (greedy marginal improvement) was introduced by Lu and Boutilier for the case of the Chamberlin–Courant rule. Here we generalize it to apply to Monroe's rule as well, and we show that it is a $1 - \frac{1}{e}$ approximation algorithm for α-U-Monroe. We point out that this approximation result for Monroe rule applies to all non-decreasing PSFs α. For the Monroe rule, the algorithm can be viewed as an extension of Algorithm B.

The algorithm proceeds as follows. We start with an empty set S. Then we execute K iterations. In each iteration we find an alternative a that is not assigned to agents yet, and that maximizes the value $\Phi_{\alpha}^{S \cup \{a\}}$. (A certain disadvantage of this algorithm for the case of Monroe is that it requires a large number of computations of Φ_{α}^{S}; since in Monroe's rule each alternative can be assigned to at most $\frac{n}{K}$ agents in the partial assignment Φ_{α}^{S}, computation of Φ_{α}^{S} is a slow process based on min-cost/max-flow algorithm.) We provide the pseudocode for Algorithm GM in Figure 5.5.

Theorem 5.9. *For any non-decreasing positional scoring function α Algorithm GM is an $(1 - \frac{1}{e})$-approximation algorithm for α-U-Monroe.*

Proof. The proof follows by applying the powerful result of Nemhauser et al. [220], which says that greedy algorithms achieve $1 - \frac{1}{e}$ approximation ratio when used

```
    Notation: $\Phi_\alpha^S$—the partial assignment that assigns a single alternative to at most
              $\lceil \frac{n}{K} \rceil$ agents, that assigns to the agents only the alternatives from $S$, and
              that maximizes the utilitarian satisfaction $\ell_1^\alpha(\Phi_\alpha^S)$.
1   $S \leftarrow \emptyset$
2   for $i \leftarrow 1$ to $K$ do
3       $a \leftarrow \mathrm{argmax}_{a \in A \setminus S} \ell_1^\alpha(\Phi_\alpha^{S \cup \{a\}})$
4       $S \leftarrow S \cup \{a\}$
5   return $\Phi_\alpha^S$
```

Figure 5.5: Pseudocode for Algorithm GM.

to optimize nondecreasing submodular functions (we explain these notions formally below). The main challenge in the proof is to define a function that, on one hand, satisfies the conditions of Nemhauser et al.'s result, and, on the other, models solutions for α-U-Monroe.

Let A be a set of alternatives, $N = [n]$ be a set of agents with preferences over A, α be an $|A|$-candidate DPSF, and $K \leq |A|$ be the number of representatives that we want to elect. We consider function $z : 2^A \to \mathbb{N}$ defined, for each set S, $S \subseteq A$ and $|S| \leq K$, as $z(S) = \ell_1^\alpha(\Phi_\alpha^S)$. Clearly, $z(S)$ is nondecreasing (that is, for each two sets A and B, if $A \subseteq B$ and $|B| \leq K$ then $z(A) \leq z(B)$. Since $\mathrm{argmax}_{S \subset A, |S|=K} z(S)$ is the set of winners under α-Monroe and since Algorithm GM builds the solution iteratively by greedily extending initially empty set S so that each iteration increases the value of $z(S)$ maximally, if z were submodular then by the results of Nemhauser et al. [220] we would get that Algorithm GM is a $(1 - \frac{1}{e})$-approximation algorithm. Thus, our goal is to show that z is submodular.

Formally, our goal is to show that for each two sets S and T, $S \subset T$, and each alternative $a \notin T$ it holds that $z(S \cup \{a\}) - z(S) \geq z(T \cup \{a\}) - z(T)$ (this is the formal definition of submodularity). First, we introduce a notion that generalizes the notion of a partial set of winners S. Let $s : A \to \mathbb{N}$ denote a function that assigns a capacity to each alternative (i.e., s gives a bound on the number of agents that a given alternative can represent). Intuitively, each set S, $S \subseteq A$, corresponds to the capacity function that assigns $\lceil \frac{n}{k} \rceil$ to each alternative $a \in S$ and 0 to each $a \notin S$. Given a capacity function s, we define a partial solution Φ_α^s to be one that maximizes the total satisfaction of the agents and that satisfies the new capacity constraints: $\forall_{a \in S} |(\Phi_\alpha^s)^{-1}(a)| \leq s(a)$. To simplify notation, we write $s \cup \{a\}$ to denote the function such that $(s \cup \{a\})(a) = s(a) + 1$ and $\forall_{a' \in S \setminus \{a\}} (s \cup \{a\})(a') = s(a')$. (Analogously, we interpret $s \setminus \{a\}$ as subtracting one from the capacity for a; provided it is nonzero.) Also, by $s \leq t$ we mean that $\forall_{a \in A} s(a) \leq t(a)$. We extend our function z to allow us to consider a subset of the agents only. For each subset N' of the agents and each capacity function s, we define $z_{N'}(s)$ to be the satisfaction of the agents in N' obtained under Φ_α^s. We will now prove a stronger variant of submodularity for our extended z. That is, we will show that for each two capacity functions s and t it holds that:

$$s \le t \Rightarrow z_N(s \cup \{a\}) - z_N(s) \ge z_N(t \cup \{a\}) - z_N(t). \tag{5.5}$$

Our proof is by induction on N. Clearly, Equation (5.5) holds for $N' = \emptyset$. Now, assuming that Equation (5.5) holds for every $N' \subset N$ we will prove its correctness for N. Let i denote an agent such that $\Phi_\alpha^{t \cup \{a\}}(i) = a$ (if there is no such agent then clearly the equation holds). Let $a_s = \Phi_\alpha^s(i)$ and $a_t = \Phi_\alpha^t(i)$. We have:

$$z_N(t \cup \{a\}) - z_N(t) = \alpha(\mathrm{pos}_i(a)) + z_{N\setminus\{i\}}(t) - \alpha(\mathrm{pos}_i(a_t)) - z_{N\setminus\{i\}}(t \setminus \{a_t\}).$$

We also have:

$$z_N(s \cup \{a\}) - z_N(s) \ge \alpha(\mathrm{pos}_i(a)) + z_{N\setminus\{i\}}(s) - \alpha(\mathrm{pos}_i(a_s)) - z_{N\setminus\{i\}}(s \setminus \{a_s\}).$$

Since Φ_α^t describes an optimal representation function under the capacity restrictions t, we have that:

$$\alpha(\mathrm{pos}_i(a_t)) + z_{N\setminus\{i\}}(t \setminus a_t) \ge \alpha(\mathrm{pos}_i(a_s)) + z_{N\setminus\{i\}}(t \setminus \{a_s\}).$$

Finally, from the inductive hypothesis for $N' = N \setminus \{i\}$ we have:

$$z_{N\setminus\{i\}}(s) - z_{N\setminus\{i\}}(s \setminus \{a_s\}) \ge z_{N\setminus\{i\}}(t) - z_{N\setminus\{i\}}(t \setminus \{a_s\}).$$

By combining these inequalities we get:

$$\begin{aligned} z_N(s \cup \{a\}) - z_N(s) &\ge \alpha(\mathrm{pos}_i(a)) + z_{N\setminus\{i\}}(s) - (\alpha(\mathrm{pos}_i(a_s)) + z_{N\setminus\{i\}}(s \setminus \{a_s\})) \\ &\ge \alpha(\mathrm{pos}_i(a)) - \alpha(\mathrm{pos}_i(a_s)) + z_{N\setminus\{i\}}(t) - z_{N\setminus\{i\}}(t \setminus \{a_s\}) \\ &\ge \alpha(\mathrm{pos}_i(a)) + z_{N\setminus\{i\}}(t) - \alpha(\mathrm{pos}_i(a_t)) - z_{N\setminus\{i\}}(t \setminus \{a_t\}) \\ &= z_N(t \cup \{a\}) - z_N(t). \end{aligned}$$

This completes the proof. □

Formally speaking, Algorithm GM is never worse than Algorithm A. For Borda satisfaction function, it inherits the approximation guarantees from Algorithm A, and for other cases Theorem 5.9 guarantees approximation ratio $1 - \frac{1}{e}$ (we do not know of any guarantees for Algorithm A for these cases). The comparison with Algorithms B and C is not nearly as easy. Algorithm GM is still likely better than them for satisfaction functions significantly different from Borda's, but for the Borda case our experiments show that Algorithm GM is much slower than Algorithms B and C and obtains almost the same or slightly worse results (see Section 5.5).

```
Notation: We use the same notation as in Algorithm C;
          $\text{num\_pos}_x(a) \leftarrow |\{i \in [n] \setminus \Phi^{\leftarrow} : pos_i(a) \le x\}|$ (the number of not-yet
          assigned agents that rank alternative $a$ in one of their first $x$ positions)
1  $\Phi = \{\}$
2  $x = \lceil \frac{m\mathrm{W}(K)}{K} \rceil$
3  for $i \leftarrow 1$ to $K$ do
4      $a_i \leftarrow \mathrm{argmax}_{a \in A \setminus \Phi^{\rightarrow}} \text{num\_pos}_x(a)$
5      foreach $j \in [n] \setminus \Phi^{\leftarrow}$ do
6          if $pos_j(a_i) < x$ then
7              $\Phi[j] \leftarrow a_i$
8  foreach $j \in A \setminus \Phi^{\leftarrow}$ do
9      $a \leftarrow$ such server from $\Phi^{\rightarrow}$ that $\forall_{a' \in \Phi^{\rightarrow}} pos_j(a) \le pos_j(a')$
10     $\Phi[j] \leftarrow a$
```

Figure 5.6: The algorithm for α_{B}-U-CC (Algorithm P).

5.4.6 Algorithm P (CC)

Since winner determination in the Chamberlin and Courant's system is a special case of the problem of selecting a collective set of items, described in Chapter 3, it is natural to apply some of the generic techniques from Chapter 4 to this case. In this subsection we recall Algorithm PosBoundAndGreedy from Section 4.3 in the context of the Chamberlin and Courant's system—we will refer to this algorithm in this specific context as Algorithm P. The pseudo-code of Algorithm P is given in Figure 5.6; for the pseudo-code of the generic version of this algorithm, Algorithm PosBoundAndGreedy, we refer the reader to Figure 4.2.

We recall that the idea of Algorithm P is to compute a certain value x and to greedily compute an assignment that (approximately) maximizes the number of agents assigned to one of their top-x alternatives. If after this process some agent has no alternative assigned, we assign him or her to her most preferred alternative from those already picked. (Recall that for nonnegative real numbers, Lambert's W-function, $\mathrm{W}(x)$, is defined to be the solution of the equation $x = \mathrm{W}(x)e^{\mathrm{W}(x)}$.)

Corollary 5.10. *Algorithm P is a polynomial-time* $(1 - \frac{2\mathrm{W}(K)}{K})$*-approximation algorithm for* α_{B}-U-CC.

Proof. Follows directly from Theorem 4.21. □

Corollary 5.11. *There is a PTAS for* α_{B}-U-CC.

Proof. Follows directly from Theorem 4.22. □

The idea used in Algorithm P can also be used to address a generalized E-CC problem. We can consider the following relaxation of E-CC: Instead of requiring that each agent's satisfaction is lower-bounded by some value, we ask that the satisfactions of a significant majority of the agents are lower-bounded by a given value. More formally, for a given constant δ, we introduce an additional quality metric:

$$\ell_{\min}^{\delta,\alpha}(\Phi) = \max_{N' \subseteq N : \frac{||N|| - ||N'||}{||N||} \leq \delta} \min_{i \in N'} \alpha(pos_i(\Phi(i))).$$

For a given $0 < \delta < 1$, by putting $x = \frac{-m\ln(\delta)}{K}$, we get $(1 + \frac{\ln(\delta)}{K})$-approximation algorithm for the $\ell_{\min}^{\delta,\alpha}(\Phi)$ metric.

Finally, we show that Algorithm P performs very well even if the voters cast truncated ballots. Proposition 5.12 gives the relation between the number of positions used by the algorithm and the approximation ratio. In Figure 5.7 we show this relation for some values of the parameters m and K.

Proposition 5.12. *Let Q be the number of top positions in the agents' preference orders that are known by the algorithm ($Q \leq \frac{m\mathrm{W}(K)}{K}$). Algorithm P that uses $x = Q$ instead of $x = \lceil \frac{m\mathrm{W}(K)}{K} \rceil$ is a polynomial-time $\left(\frac{m-Q}{m-1}(1 - e^{-\frac{QK}{m}})\right)$-approximation algorithm for α_{B}-U-CC.*

Proof. Let n_i denote the number of the agents not-yet-assigned until the $(i+1)$-th iteration of the algorithm. Using the same reasoning as in Theorem 4.21 we show that $n_i \leq n(1 - \frac{Q}{m})^i$. As before, our proof proceeds by induction on i. It is evident that the hypothesis is correct for $i = 0$. Now, assuming that $n_i \leq n(1 - \frac{Q}{m})^i$, we assess n_{i+1} as follows:

$$n_{i+1} \leq n_i - \frac{n_i Q}{m - i} \leq n_i \left(1 - \frac{Q}{m}\right) \leq n \left(1 - \frac{Q}{m}\right)^{i+1}.$$

This proves the hypothesis. Thus, we can bound n_K:

$$n_K \leq n \left(1 - \frac{Q}{m}\right)^K \leq n \left(\frac{1}{e}\right)^{\frac{QK}{m}}.$$

This means that the satisfaction of the assignment Φ returned by our algorithm is at least:

$$\ell_1^{\alpha_{\mathrm{B}}}(\Phi) \geq (n - n_K)(m - Q) \geq n(m - Q)(1 - e^{-\frac{QK}{m}}).$$

In effect, it holds that:

$$\frac{\ell_1^{\alpha_{\mathrm{B}}}(\Phi)}{\mathrm{OPT}} \geq \frac{n(m-Q)(1 - e^{-\frac{QK}{m}})}{n(m-1)} \geq \frac{m-Q}{m-1}\left(1 - e^{-\frac{QK}{m}}\right).$$

This completes the proof. □

For example, for Polish parliamentary elections ($K = 460$, $m = 6000$), it suffices that each voter ranks only 0.5% of her top alternatives (that is, about 30 alternatives) for the algorithm to find a solution with guaranteed satisfaction at least 90% of the one (possibly infeasible) where every voter is assigned to her top alternative.

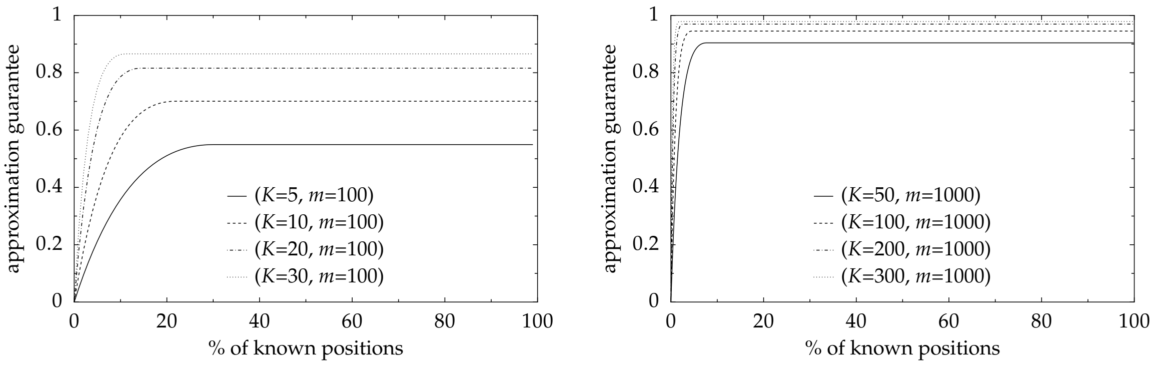

Figure 5.7: The relation between the percentage of the known positions and the approximation ratio of Algorithm P for α_{B}-U-CC.

5.4.7 ILP Formulation (Monroe, CC)

To experimentally measure the quality of our approximation algorithms, we compare the results against optimal solutions that we obtain using integer linear programs (ILPs) that solve the Monroe and Chamberlin–Courant winner determination problem. An ILP for the Monroe rule was provided by Potthoff and Brams [244], Lu and Boutilier [188] adapted it also for the Chamberlin–Courant rule with arbitrary PSF α. For the sake of completeness, below we recall the ILP whose optimal solutions correspond to α-U-Monroe winner sets for the given election (we also indicate which constraints to drop to obtain an ILP for finding α-U-CC winner sets):

1. For each i, $1 \leq i \leq n$, and each j, $1 \leq j \leq m$ we have a 0/1 variable a_{ij} indicating whether alternative a_j represents agent i. For each j, $1 \leq j \leq m$, we have a 0/1 variable x_j indicating whether alternative a_j is included in the set of winners.

2. Our goal is to maximize the value $\sum_{i=1}^{n} \alpha(pos_i(a_j))a_{ij}$ subject to the following constraints:

   (a) For each i and j, $1 \leq i \leq n, 1 \leq j \leq m$, $0 \leq a_{ij} \leq x_j$ (alternative a_j can represent agent i only if a_j belongs to the set of winners)

   (b) For each i, $1 \leq i \leq n$, $\sum_{1 \leq j \leq m} a_{ij} = 1$ (every agent is represented by exactly one alternative).

   (c) For each j, $1 \leq j \leq m$, $x_j \lfloor \frac{n}{K} \rfloor \leq \sum_{1 \leq i \leq n} a_{ij} \leq x_j \lceil \frac{n}{K} \rceil$ (each alternative either does not represent anyone or represents between $\lfloor \frac{n}{K} \rfloor$ and $\lceil \frac{n}{K} \rceil$ agents; if we remove these constraints then we obtain an ILP for the Chamberlin-Courant rule).

   (d) $\sum_{j=1}^{n} x_j \leq K$ (there are exactly K winners[4]).

[4]For the Monroe framework inequality here is equivalent to equality. We use the inequality so that deleting constraints from item (2c) leads to an ILP for the Chamberlin-Courant rule.

We used the GLPK 4.47 package (GNU Linear Programming Kit, version 4.47) to solve these ILPs, whenever it was possible to do so in reasonable time.

5.5 Empirical Evaluation of the Algorithms

In this section we present the results of empirical evaluation of algorithms from Section 5.4. In the experiments we evaluated versions of the randomized algorithms that use exactly 100 sampling steps. In all cases but one, we have used Borda PSF to measure voter satisfaction. In one case, with six candidates, we have used DPSF defined through vector $(3, 3, 3, 2, 1, 0)$ (we made this choice due to the nature of the data set used; see discussion later).

We have conducted four sets of experiments. First, we have tested all our algorithms on relatively small elections (up to 10 candidates, up to 100 agents). In this case we were able to compare the solutions provided by our algorithms with the optimal ones. (To obtain the optimal solutions, we were using the ILP formulations and the GLPK's ILP solver.) Thus we report the quality of our algorithms as the average of fractions C/C_{opt}, where C is the satisfaction obtained by a respective algorithm and C_{opt} is the satisfaction in the optimal solution. For each algorithm and data set, we also report the average fraction C/C_{ideal}, where C_{ideal} is the satisfaction that the voters would have obtained if each of them were matched to her most preferred alternative. In our further experiments, where we considered larger elections, we were not able to compute optimal solutions, but fraction C/C_{ideal} gives a lower bound for C/C_{opt}. We report this value for small elections so that we can see an example of the relation between C/C_{opt} and C/C_{ideal} and so that we can compare the results for small elections with the results for the larger ones. Further, for the case of Borda PSF the C/C_{ideal} fraction has a very natural interpretation: If its value for a given solution is v, then, on the average, in this solution each voter is matched to an alternative that he or she prefers to $(m - 1)v$ alternatives.

In our second set of experiments, we have run our algorithms on large elections (thousands of agents, hundreds of alternatives), coming either from the NetFlix data set (see below) or generated by us using one of our models. Here we reported the average fraction C/C_{ideal} only. We have analyzed the quality of the solutions as a function of the number of agents, the number of candidates, and the relative number of winners (fraction K/m). (This last set of results is particularly interesting because in addition to measuring the quality of our algorithms, it allows one to asses the size of a committee one should seek if a given average satisfaction of agents is to be obtained).

In the third set of experiments, we have investigated the effect of submitting truncated ballots (i.e., preference orders where only some of the top alternatives are ranked). Specifically, we focused on the relation between the fraction of ranked alternatives and the approximation ratio of the algorithms. We run our experiments on relatively large instances describing agents' preferences; thus, here as in the previous set of experiments, we used the NetFlix data set and the synthetic data sets. We report the quality of the algorithms as the ratio C/C_{ideal}.

In the fourth set of experiments we have measured running times of our algorithms and of the ILP solver. Even though all our algorithms (except for the ILP based ones) are polynomial-time, in practice some of them are too slow to be useful.

5.5.1 Experimental Data

For the evaluation of the algorithms we have considered both real-life preference-aggregation data and synthetic data, generated according to a number of election models. The experiments reported in this chapter predate the work of Mattei and Walsh [201] on gathering a large collection of data sets with preference data, but we mention that we have contributed several data sets to their collection.

Real-Life Data

We have used real-life data regarding people's preference on sushi types, movies, college courses, and competitors' performance in figure-skating competitions. One of the major problems regarding real-life preference data is that either people express preferences over a very limited set of alternatives, or their preference orders are partial. To address the latter issue, for each such data set we complemented the partial orders to be total orders using the technique of Kamishima [154]. (The idea is to complete each preference order based on those reported preference orders that appear to be similar.)

Some of our data sets contain a single profile, whereas the others contain multiple profiles. When preparing data for a given number m of candidates and a given number n of voters from a given data set, we used the following method: We first uniformly at random chose a profile within the data set, and then we randomly selected n voters and m candidates. We used preference orders of these n voters restricted to these m candidates.

Sushi Preferences. We used the set of preferences regarding sushi types collected by Kamishima [154].[5] Kamishima has collected two sets of preferences, which we call S1 and S2. Data set S1 contains complete rankings of 10 alternatives collected from 5000 voters. S2 contains partial rankings provided by 5000 voters over a set of 100 alternatives (each vote ranks 10 alternatives). We used Kamishima [154] technique to obtain total rankings.

Movie Preferences. Following Mattei et al. [199], we have used the NetFlix data set[6] of movie preferences (we call it Mv). NetFlix data set contains ratings collected from about 480 thousand distinct users regarding 18 thousand movies. The users

[5]The sushi data set is available under the following url: `http://www.kamishima.net/sushi/`
[6]http://www.netflixprize.com/

rated movies by giving them a score between 1 (bad) and 5 (good). The set contains about 100 million ratings. We have generated 50 profiles using the following method: For each profile we have randomly selected 300 movies, picked 10000 users that ranked the highest number of the selected movies, and for each user we have extended her ratings to a complete preference order using the method of Kamishima [154].

Course Preferences. Each year the students at the AGH University choose courses that they would like to attend. Depending on a particular year and the students' level of advancement, the students are offered a number of courses of which they have to select some that they will attend. In our case, the students were offered a choice of six courses of which they had to choose three. Thus the students were asked to give an unordered set of their three top-preferred courses and a ranking of the remaining ones (in case too many students selected a course, those with the highest GPA were enrolled and the remaining ones were moved to their less-preferred courses). In this data set, which we call Cr, we have 120 voters (students) and 6 alternatives (courses). However, due to the nature of the data, instead of using Borda count PSF as the satisfaction measure, we have used the vector $(3, 3, 3, 2, 1, 0)$. Currently this data set is available as part of PrefLib [201].

Figure Skating. This data set, which we call Sk, contains preferences of the judges over the performances in figure-skating competitions. The data set contains 48 profiles, each describing a single competition. Each profile contains preference orders of 9 judges over about 20 participants. The competitions include European skating championships, Olympic Games, World Junior, and World Championships, all from 1998.[7] (Note that while in figure skating judges provide numerical scores, this data set is preprocessed to contain preference orders.)

Synthetic Data

For our tests, we have also used profiles generated using three well-known distributions of preference orders.

Impartial Culture. Under the impartial culture model of preferences (which we denote IC), for a given set A of alternatives, each voter's preference order is drawn uniformly at random from the set of all possible total orders over A. While not very realistic, profiles generated using impartial culture model are a standard testbed of election-related algorithms.

Polya-Eggenberger Urn Model. Following McCabe-Dansted and Slinko [204] and Walsh [298], we have used the Polya-Eggenberger urn model [21] (which we denote Ur). In this model we generate votes as follows. We have a set A of m alternatives and an urn that initially contains all $m!$ preference orders over A. To generate a vote, we simply randomly pick one from the urn (this is our generated vote), and

[7]This data set is available under the following url: `http://rangevoting.org/SkateData1998.txt`.

then—to simulate correlation between voters—we return a copies of this vote to the urn. When generating an election with m candidates using the urn model, we have set the parameter a so that $\frac{a}{m!} = 0.05$ (Both McCabe-Dansted and Slinko [204] and Walsh [298] call this parameter b; we mention that those authors use much higher values of b but we felt that too high a value of b leads to a much too strong correlation between votes).

Generalized Mallow's Model. We refer to this data set as Ml. Let $\succ$ and $\succ'$ be two preference orders over some alternative set A. Kendal-Tau distance between $\succ$ and $\succ'$, denoted $d_K(\succ, \succ')$, is defined as the number of pairs of candidates $x, y \in A$ such that either $x \succ y \land y \succ' x$ or $y \succ x \land x \succ' y$.

Under Mallow's distribution of preferences [195] we are given two parameters: A *center* preference order $\succ$ and a number ϕ between 0 and 1. The model says that the probability of generating preference order $\succ'$ is proportional to the value $\phi^{d_K(\succ,\succ')}$. To generate preference orders following Mallow's distribution, we use the algorithm given by Lu and Boutilier [189].

In our experiments, we have used a mixture of Mallow's models. Let A be a set of alternatives and let ℓ be a positive integer. This mixture model is parameterized by three vectors, $\Lambda = (\lambda_1, \dots, \lambda_\ell)$ (where each λ_i is between 0 and 1, and $\sum_{i=1}^{\ell} \lambda_1 = 1$), $\Phi = (\phi_1, \dots, \phi_\ell)$ (where each ϕ_i is a number between 0 and 1), and $\Pi = (\succ_1, \dots, \succ_\ell)$ (where each $\succ_i$ is a preference order over A). To generate a vote, we pick a random integer i, $1 \leq i \leq \ell$ (each i is chosen with probability λ_i), and then generate the vote using Mallow's model with parameters $(\succ_i, \phi_i)$.

For our experiments, we have used $a = 5$, and we have generated vectors Λ, Φ, and Π uniformly at random.

5.5.2 Evaluation on Small Instances

We now present the results of our experiments on small elections. For each data set, we generated elections with the number of agents $n = 100$ ($n = 9$ for data set Sk because there are only 9 voters there) and with the number of alternatives $m = 10$ ($m = 6$ for data set Cr because there are only 6 alternatives there) using the method described in Section 5.5.1 for the real-life data sets, and in the natural obvious way for synthetic data. For each algorithm and for each data set we ran 500 experiments on different instances for $K = 3$ (for the Cr data set we used $K = 2$) and 500 experiments for $K = 6$ (for Cr we set $K = 4$). For Algorithm C (both for Monroe and for CC) we set the parameter d, describing the number of assignment functions computed in parallel, to 15. The results (average fractions C/C_{opt} and C/C_{ideal}) for $K = 3$ are given in Tables 5.1 and 5.3; the results for $K = 6$ are given in Tables 5.2 and 5.4 (they are almost identical as for $K = 3$). For each experiment in this section we also computed the standard deviation; it was always on the order of 0.01. The results lead to the following conclusions:

	Monroe					CC			
	A	B	C	GM	R	C	GM	P	R
S1	0.94	0.99	≈ 1.0	0.99	0.99	1.0	≈ 1.0	0.99	0.99
S2	0.95	0.99	1.0	≈ 1.0	0.99	1.0	≈ 1.0	0.98	0.99
Mv	0.96	≈ 1.0	1.0	≈ 1.0	0.98	1.0	≈ 1.0	0.96	≈ 1.0
Cr	0.98	0.99	1.0	≈ 1.0	0.99	1.0	≈ 1.0	1.0	≈ 1.0
Sk	0.99	≈ 1.0	1.0	≈ 1.0	0.94	1.0	≈ 1.0	0.85	0.99
IC	0.94	0.99	≈ 1.0	0.99	0.99	1.0	≈ 1.0	0.99	0.99
Ml	0.94	0.99	1.0	0.99	0.99	1.0	≈ 1.0	0.99	0.99
Ur	0.95	0.99	≈ 1.0	0.99	0.99	1.0	0.99	0.97	0.99

Table 5.1: The average quality of the algorithms compared with the optimal solution (C/C_{opt}) for the small instances of data and for $K = 3$ ($K = 2$ for Cr); $m = 10$ ($m = 6$ for Cr); $n = 100$ ($n = 9$ for Sk).

	Monroe					CC			
	A	B	C	GM	R	C	GM	P	R
S1	0.95	≈ 1.0	1.0	0.99	0.99	1.0	≈ 1.0	0.97	0.99
S2	0.94	0.99	≈ 1.0	0.99	0.99	1.0	≈ 1.0	0.98	≈ 1.0
Mv	0.95	0.99	1.0	≈ 1.0	0.98	1.0	≈ 1.0	0.97	≈ 1.0
Cr	0.96	≈ 1.0	1.0	≈ 1.0	0.99	1.0	1.0	1.0	1.00
Sk	0.99	≈ 1.0	1.0	≈ 1.0	0.88	1.0	1.0	0.91	≈ 1.0
IC	0.95	0.99	≈ 1.0	0.99	0.99	1.0	≈ 1.0	0.99	0.99
Ml	0.95	0.99	≈ 1.0	0.99	0.99	1.0	≈ 1.0	0.98	0.99
Ur	0.96	0.99	≈ 1.0	0.99	≈ 1.0	1.0	≈ 1.0	0.96	0.99

Table 5.2: The average quality of the algorithms compared with the optimal solution (C/C_{opt}) for the small instances of data and for $K = 6$ ($K = 4$ for Cr); $m = 10$ ($m = 6$ for Cr); $n = 100$ ($n = 9$ for Sk).

1. For the case of Monroe, already Algorithm A obtains very good results, but nonetheless Algorithms B and C improve noticeably upon Algorithm A. In particular, Algorithm C (for $d = 15$) obtains the highest satisfaction on all data sets and in almost all cases is able to find an optimal solution.

2. Both for Monroe and for CC, Algorithm R gives slightly worse solutions than Algorithm C.

3. The results do not seem to depend on the data sets used in the experiments (the only exception is Algorithm R for the Monroe system on data set Sk; however Sk has only 9 voters so it can be viewed as a border case).

	Monroe					CC			
	A	B	C	GM	R	C	GM	P	R
S1	0.85	0.89	0.9	0.89	0.89	0.92	0.89	0.91	0.92
S2	0.85	0.89	0.89	0.89	0.89	0.93	0.9	0.91	0.92
Mv	0.88	0.92	0.92	0.92	0.91	0.97	0.92	0.93	0.97
Cr	0.94	0.97	0.96	0.96	0.96	0.97	0.97	0.97	0.97
Sk	0.96	0.96	0.97	0.97	0.91	1.0	0.97	0.82	0.99
IC	0.8	0.84	0.85	0.84	0.84	0.85	0.83	0.84	0.85
Ml	0.83	0.88	0.88	0.9	0.88	0.92	0.90	0.89	0.94
Ur	0.8	0.85	0.86	0.87	0.85	0.9	0.87	0.87	0.89

Table 5.3: The average quality of the algorithms compared with the simple lower bound (C/C_{ideal}) for the small instances of data and for $K = 3$ ($K = 2$ for Cr); $m = 10$ ($m = 6$ for Cr); $n = 100$ ($n = 9$ for Sk).

	Monroe					CC			
	A	B	C	GM	R	C	GM	P	R
S1	0.91	0.96	0.96	0.95	0.95	0.98	0.98	0.96	0.98
S2	0.88	0.93	0.93	0.93	0.93	0.98	0.98	0.96	0.98
Mv	0.85	0.89	0.89	0.89	0.88	0.99	0.99	0.97	0.99
Cr	0.95	0.98	0.99	0.99	0.98	1.0	1.0	1.0	1.0
Sk	0.91	0.92	0.92	0.92	0.81	1.0	1.0	0.91	≈ 1.0
IC	0.91	0.95	0.95	0.94	0.95	0.96	0.96	0.95	0.95
Ml	0.89	0.94	0.94	0.94	0.93	0.97	0.98	0.95	0.98
Ur	0.91	0.95	0.95	0.94	0.95	0.98	0.98	0.94	0.97

Table 5.4: The average quality of the algorithms compared with the simple lower bound (C/C_{ideal}) for the small instances of data and for $K = 6$ ($K = 4$ for Cr); $m = 10$ ($m = 6$ for Cr); $n = 100$ ($n = 9$ for Sk).

5.5.3 Evaluation on Larger Instances

For experiments on larger instances we needed data sets with at least $n = 10000$ agents. Thus we used the NetFlix data set and synthetic data. (Additionally, we run the subset of experiments (for $n \leq 5000$) also for the S2 data set.) For the Monroe rule we present results for Algorithm A, Algorithm C, and Algorithm R, and for the Chamberlin–Courant rule we present results for Algorithm C and Algorithm R. We limit the set of algorithms for the sake of the clarity of the presentation. For Monroe we chose Algorithm A because it is the simplest and the fastest one, Algorithm C because it is the best generalization of Algorithm A that we were able to run in reasonable time, and Algorithm R to compare a randomized algorithm to deterministic ones. For the Chamberlin–Courant rule we chose Algorithm C because

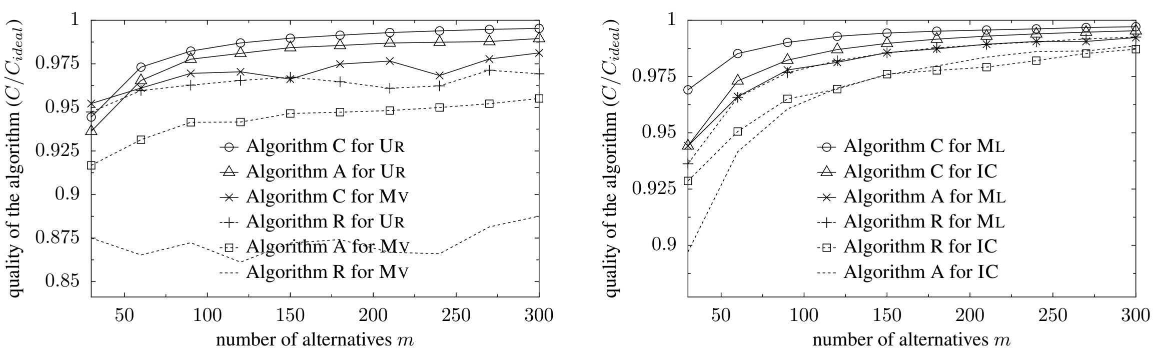

Figure 5.8: The relation between the number of alternatives m and the quality of the algorithms C/C_{ideal} for the Monroe system; $K/m = 0.3$; $n = 1000$.

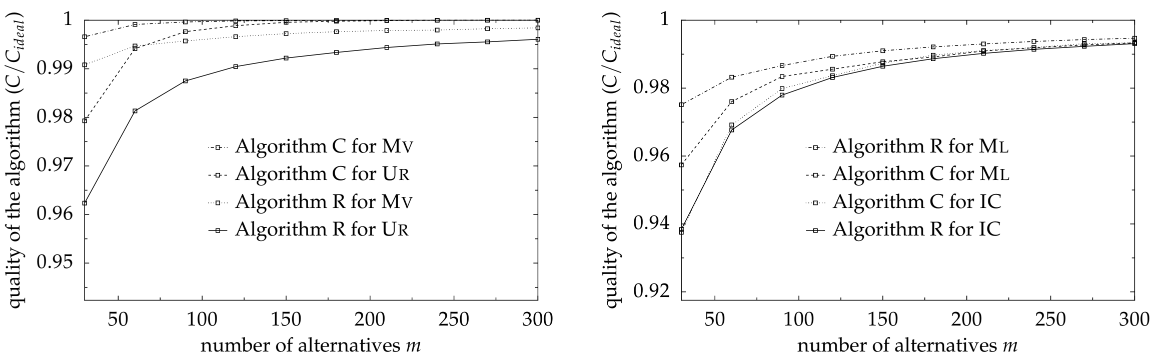

Figure 5.9: The relation between the number of alternatives m and the quality of the algorithms C/C_{ideal} for the Chamberlin–Courant system; $K/m = 0.3$; $n = 1000$.

it is, intuitively, the best one, and we chose Algorithm R for the same reason as in the case of Monroe.

First, for each data set and for each algorithm we fixed the value of m and K and for each n ranging from 1000 to 10000 with the step of 1000 we run 50 experiments. We repeated this procedure for 4 different combinations of m and K: ($m = 10$, $K = 3$), ($m = 10$, $K = 6$), ($m = 100$, $K = 30$) and ($m = 100$, $K = 60$). We measured the statistical correlation between the number of voters and the quality of the algorithms C/C_{ideal}. The ANOVA test in most cases showed that there is no such correlation. The only exception was S2 data set, for which we obtained an almost negligible correlation. For example, for $(m = 10, K = 3)$ Algorithm C under data set S2 for Monroe's system for $n = 5000$ gave $C/C_{\text{ideal}} = 0.88$, while for $n = 100$ (in the previous section) we got $C/C_{\text{ideal}} = 0.89$. Thus we conclude that in practice the number of agents has almost no influence on the quality of the results provided by our algorithms.

Next, we fixed the number of voters $n = 1000$ and the ratio $K/m = 0.3$, and for each m ranging from 30 to 300 with the step of 30 (naturally, as m changed, so did K to maintain the ratio K/m), we run 50 experiments. We repeated this procedure

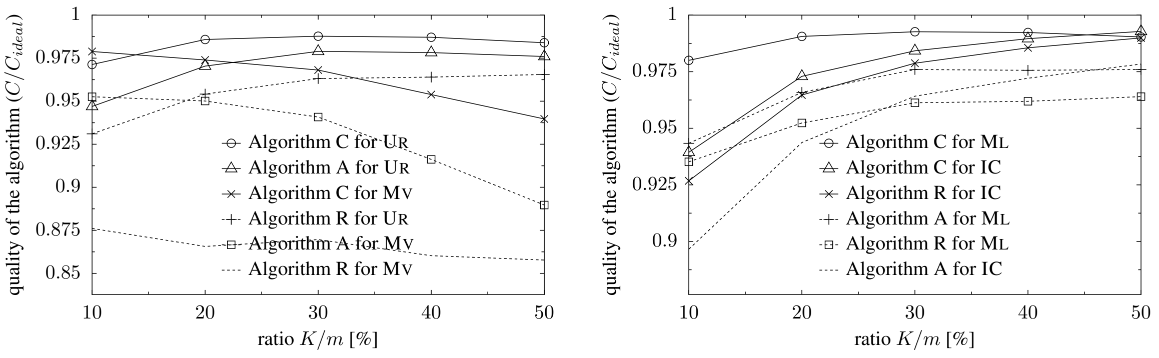

Figure 5.10: The relation between the ratio K/m and the quality of the algorithms C/C_{ideal} for the Monroe system; $m = 100$; $n = 1000$.

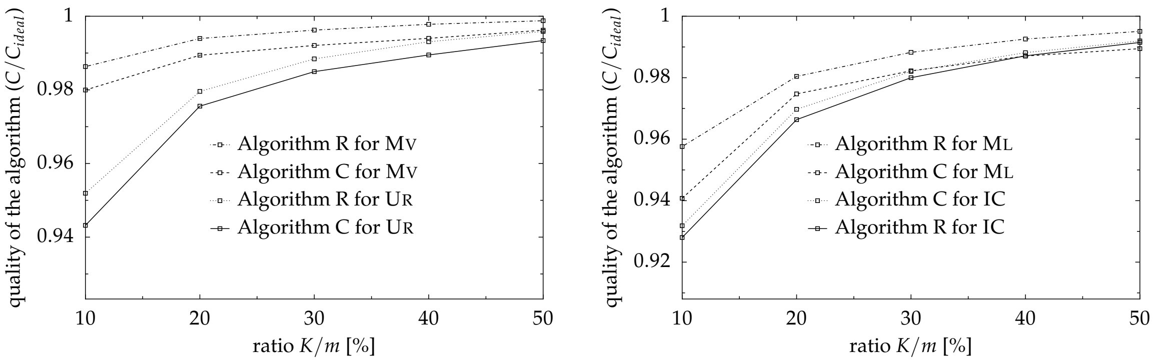

Figure 5.11: The relation between the ratio K/m and the quality of the algorithms C/C_{ideal} for the Chamberlin–Courant system; $m = 100$; $n = 1000$.

for $K/m = 0.6$. The relation between m and C/C_{ideal} for MV and UR, under both the Monroe rule and the Chamberlin–Courant rule, is given in Figures 5.8 and 5.9 (the results for $K/m = 0.6$ look similar).

Finally, we fixed $n = 1000$ and $m = 100$, and for each K/m ranging from 0.1 and 0.5 with the step of 0.1 we run 50 experiments. The relation between the ratio K/m and the quality C/C_{ideal} is presented in Figures 5.10 and 5.11.

For the case of Chamberlin–Courant system, increasing the size of the committee to be elected improves overall agents' satisfaction. Indeed, since there are no constraints on the number of agents matched to a given alternative, a larger committee means more opportunities to satisfy the agents. For the Monroe rule, a larger committee may lead to a lower total satisfaction. This happens if many agents like a particular alternative a lot, but only some of them can be matched to this alternative and others have to be matched to their less preferred ones. Nonetheless, we see that Algorithm C achieves $C/C_{\text{ideal}} = 0.925$ even for $K/m = 0.5$ for the NetFlix data set.

Our conclusions from these experiments are the following. For the Monroe rule, even Algorithm A achieves very good results. However, Algorithm C consistently achieves better (indeed, almost perfect) ones. For the Chamberlin–Courant rule the randomized algorithm on some datasets performs better than the deterministic ones. However, even in such cases, the improvement over the Algorithm C is small.

5.5.4 Truncated ballots

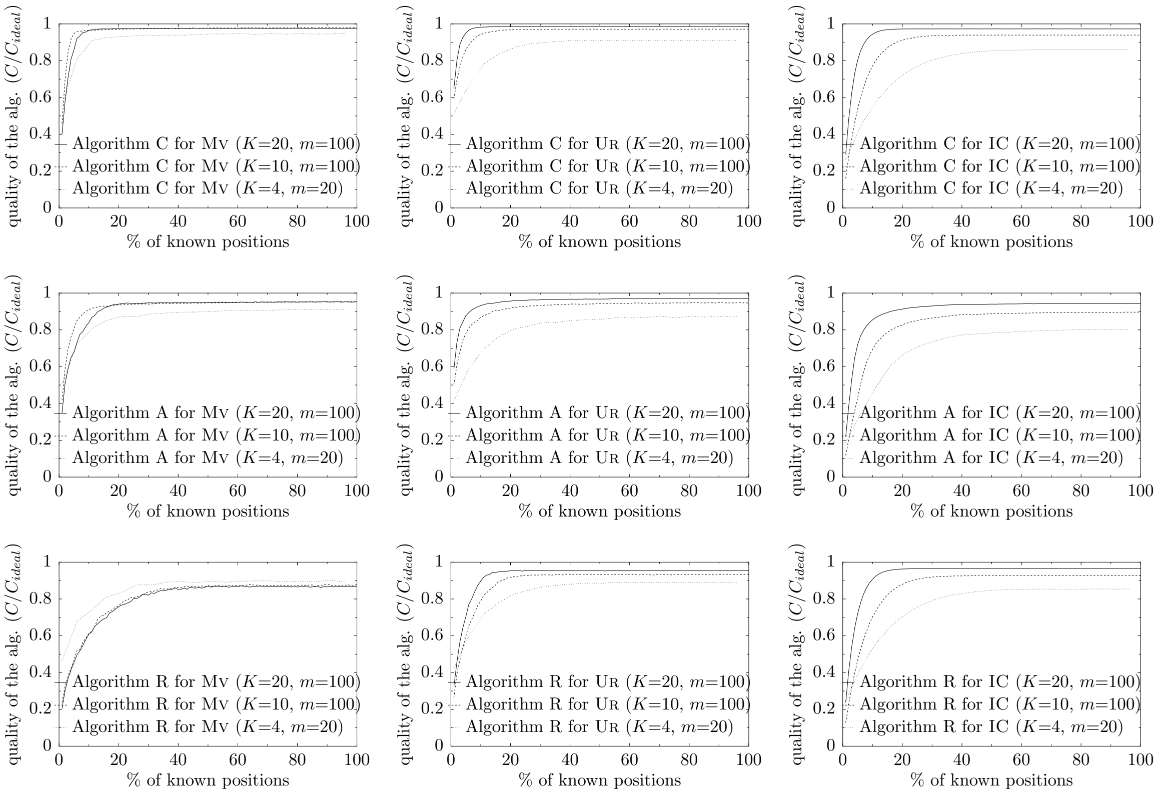

Figure 5.12: The relation between the percentage of known positions P/m [%] and the quality of the algorithm C/C_{ideal} for Algorithms C, A, and R for Monroe's system. Each row of the plots describes one algorithm; each column describes one data set; $n = 1000$. (Results for the Mallows model are similar to those for the urn model and are omitted for clarity.)

The purpose of our third set of experiments was to see how our algorithms behave in practical settings with truncated ballots. We conducted this part of evaluation on relatively large instances, including $n = 1000$ agents and up to $m = 100$ alternatives. Thus, in this set of experiments, we used the same sets of data as in the previous subsection: the Netflix data set and the synthetic distributions. Similarly, we evaluated the same algorithms: Algorithm A, C, and R for the case of Monroe's system, and Algorithm C, and R for the case of the Chamberlin–Courant system.

For each data set and for each algorithm we run experiments for 3 independent settings with different values of the parameters describing the elections: (1) $m = 100$,

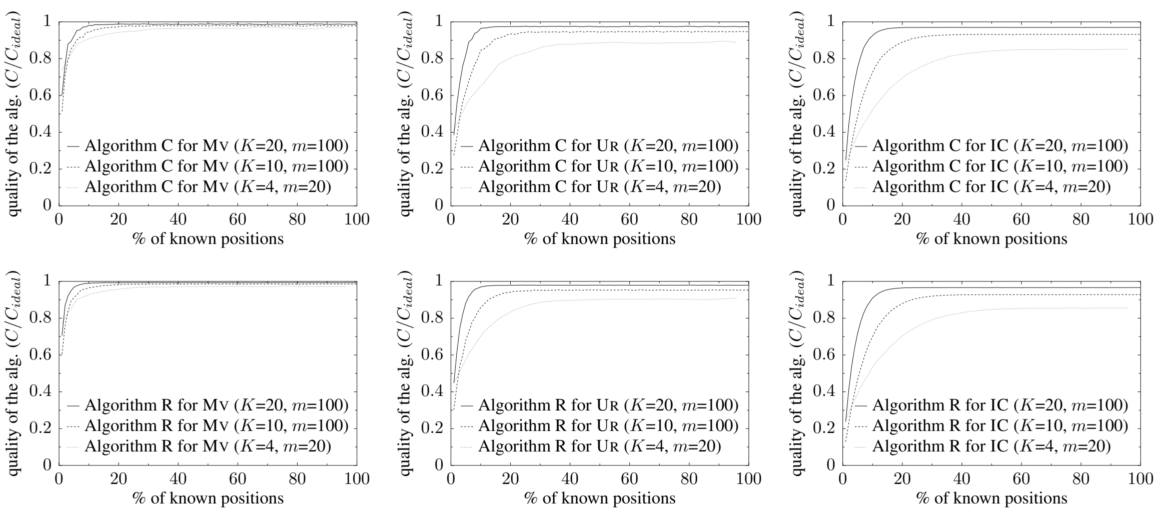

Figure 5.13: The relation between the percentage of known positions P/m [%] and the quality of the algorithm C/C_{ideal} for Algorithms C and R for the Chamberlin–Courant system. Each row of the plots describes one algorithm; each column describes one data set; $n = 1000$. (Results for the Mallows model are similar to those for the urn model and are omitted for clarity.)

$K = 20$, (2) $m = 100$, $K = 10$, and (3) $m = 20$, $K = 4$. For each setting we run the experiments for the values of P (the number of known positions) varying between 1 and m.

For each algorithm, data set, setting, and each value of P we run 50 independent experiments in the following way. From a data set we sampled a sub-profile of the appropriate size $n \times m$. We truncated this profile to the P first positions. We run the algorithm for the truncated profile and calculated the quality ratio C/C_{ideal}. When calculating C/C_{ideal} we assumed the worst case scenario, i.e., that the satisfaction of the agent from an alternative outside of his/her first P positions is equal to 0. In other words, we used the positional scoring function described by the following vector: $\langle m-1, m-2, \ldots, m-P, 0, \ldots 0\rangle$. Next, we averaged the values of C/C_{ideal} over all 50 experiments.

The relation between the percentage of the known positions in the preference profile and the average quality of the algorithm for the Monroe and Chamberlin–Courant systems are plotted in Figures 5.12 and 5.13, respectively. We omit the plots for Mallow's model, as in this case we obtained almost identical results as for the Urn model. We have the following conclusions.

1. All the algorithms require only small number of the top positions to achieve their best quality. Here, the deterministic algorithms are superior.

2. The small elections with synthetic distributions appear to be the worst case scenario—in such case we require the knowledge of about 40% of the top

positions to obtain the highest approximation ratios of the algorithms. In the case of the NetFlix data set, even on small instances the deterministic algorithms require only about 8% of the top positions to get their best quality (however the quality is already high for 3-5% of the top positions). For the larger number of the alternatives, the algorithms do not require more than 3% of the top positions to reach their top results.

3. Algorithm C does not only give the best quality but it is also most immune to the lack of knowledge. These results are more evident for the case of the Monroe system.

5.5.5 Running time

In our final set of experiments, we have measured running times of our algorithms on the data set Mv. We have used a machine with Intel Pentium Dual T2310 1.46GHz processor and 1.5GB of RAM. In Figure 5.14 we show the running times of the GLPK ILP solver for the Monroe and for Chamberlin–Courant rules. These running times are already large for small instances and they are increasing exponentially with the number of voters. For the Monroe rule, even for $K = 9, m = 30, n = 100$ some of the experiments timed out after 1 hour, and for $K = 9, m = 30, n = 200$ none of the experiments finished within one day. Thus we conclude that the real application of the ILP-based algorithm is very limited.

Example running times of the other algorithms for some combinations of n, m, and K are presented in Table 5.5. For the case of CC, essentially all the algorithms are very fast and the quality of computed solutions is the main criterion in choosing among them. For the case of Monroe, the situation is more complicated. While for

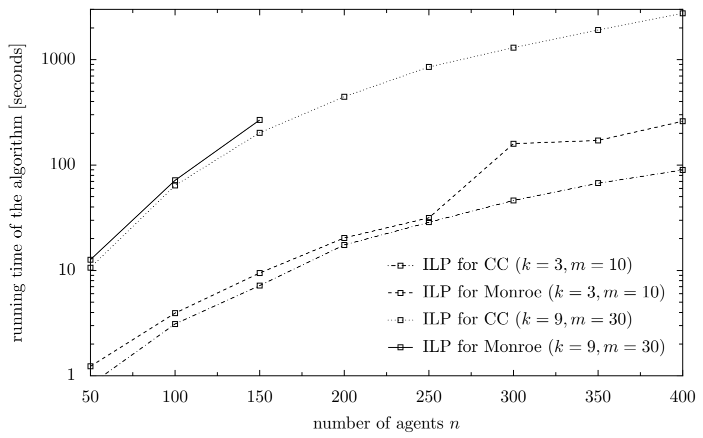

Figure 5.14: The running time of the standard ILP solver for the Monroe and for the Chamberlin–Courant systems. For Monroe's system, for $K = 9, m = 30$, and for $n \geq 200$ none of the single algorithm execution finished within 1 day.

		$m = 10$, $K = 3$			$m = 10$, $K = 6$		
	$n =$	2000	6000	10000	2000	6000	10000
Monroe	A	0.01	0.03	0.05	0.01	0.04	0.07
	B	0.08	0.9	2.3	0.2	1.4	3.6
	C	1.1	8	22	2.1	16	37
	GM	0.8	7.3	20	1.9	13	52
	R	7.6	50	180	6.5	52	140
CC	C	0.02	0.07	0.12	0.05	0.14	0.26
	GM	0.003	0.009	0.015	0.003	0.01	0.018
	P	0.009	0.032	0.05	0.008	0.02	0.05
	R	0.014	0.04	0.065	0.02	0.06	0.11

		$m = 100$, $K = 30$			$m = 100$, $K = 60$		
	$n =$	2000	6000	10000	2000	6000	10000
Monroe	A	0.5	1.6	2.8	0.9	2.8	4.9
	B	0.8	4	9.5	1.7	8	18
	C	38	140	299	64	221	419
	GM	343	2172	5313	929	5107	13420
	R	41	329	830	88	608	1661
CC	C	4.3	11	19	7.5	19	31
	GM	0.06	0.2	0.4	0.09	0.3	0.7
	P	0.03	0.1	0.26	0.03	0.1	0.2
	R	0.06	0.24	0.45	0.1	0.4	0.8

Table 5.5: Example running times of the algorithms [in seconds].

small elections all the algorithms are practical, for elections with thousands of voters, using Algorithm GM becomes problematic. Indeed, even Algorithm C can be seen as a bit too slow if one expects immediate results. On the other hand, Algorithms A and B seem perfectly practical and, as we have seen in the previous experiments, give high-quality results.

5.6 Related Work

A large number of papers are related to our research in terms of methodology (the study of computational complexity and approximation algorithms for winner determination under various NP-hard election rules), in terms of perspective and motivation (e.g., due to the resource allocation view of Monroe and Chamberlin–Courant rules that we take), and in terms of formal similarity (e.g., winner determination under the Chamberlin–Courant rule can be seen as a form of the facility location problem). Below we review this related literature.

NP-hardness of winner determination under the Monroe and Chamberlin–Courant rules was shown by Procaccia et al. [245] and by Lu and Boutilier [188]. Worse yet, the hardness holds even if various natural parameters of the election are small [27]. Rare easy cases include those, where the committee to be elected is small, or we consider the Chamberlin–Courant rule and the voters have single-peaked [27] or single-crossing preferences [283].

There are several single-winner voting rules for which winner determination is known to be NP-hard. These rules include, for example, Dodgson's rule [18,26,134], Young's rule [26,259], and Kemeny's rule [18,25,132]. For the single-transferable vote rule (STV), the winner determination problem becomes NP-hard if we use the parallel-universes tie-breaking [66]. Many of these hardness results hold even in the sense of parameterized complexity theory (however, there also is a number of fixed-parameter tractability results; see the references above for details).

These hardness results motivated the search for approximation algorithms. There are now very good approximation algorithms for Kemeny's rule [6,68,161] and for Dodgson's rule [41,42,97,138,203]. In both cases the results are, in essence, optimal. For Kemeny's rule there is a polynomial-time approximation scheme [161] and for Dodgson's rule the achieved approximation ratio is optimal under standard complexity-theoretic assumptions [41] (unfortunately, the approximation ratio is not constant but depends logarithmically on the number of candidates). On the other hand, for Young's rule it is known that no good approximation algorithms exist [41].

The work of Caragiannis et al. [42] and of Faliszewski et al. [97] on approximate winner determination for Dodgson's rule is particularly interesting from our perspective. In the former, the authors advocate treating approximation algorithms for Dodgson's rule as voting rules in their own right and design them to have desirable properties. In the latter, the authors show that a well-established voting rule (Maximin rule) is a reasonable (though not optimal) approximation of Dodgson's rule. This perspective is important for anyone interested in using approximation algorithms for winner determination in elections (as might be the case for our algorithms for the Monroe and Chamberlin–Courant rules).

In this chapter we take the view that the Monroe and Chamberlin–Courant rules are special cases of the following resource allocation problem. The alternatives are sharable resources, each with a certain capacity defined as the maximal number of agents that may share this resource. Each agent has preferences over the resources and is interested in getting exactly one. The goal is to select a predetermined number K of resources and to find an optimal allocation of these resources (see Section 5.2 for details). This provides a unified framework for the two rules and reveals the connection of proportional representation problem to other resource allocation problems. In particular, it closely resembles multi-unit resource allocation with single-unit demand [271] (see also the work of Chevaleyre et al. [54] for a survey of the most fundamental issues in the multiagent resource allocation theory) and resource allocation with sharable indivisible goods [7,54].

	Monroe	Chamberlin-Courant	General Assignment
Utilitarian case	Good approximation	Good approximation	Open problem
Egalitarian case	Inapproximability Theorem 5.1	Inapproximability Theorem 5.3	Inapproximability Theorem 5.1 Theorem 5.3

Table 5.6: Summary of approximability results for the Monroe and Chamberlin-Courant multiwinner voting systems.

We recall that other connections of the Monroe and Chamberlin–Courant rules to several other problems, including the facility location problem, are pointed out in Chapter 3.

5.7 Summary

In this chapter we have considered the winner determination problem under two election rules: Chamberlin and Courant and Monroe systems. We have shown that these two winner determination problems are special cases of the capacitated disjunctive variant of the problem of selecting a collective set of items, introduced and defined in Chapter 3. Since it is known that the winners for Chamberlin and Courant and Monroe voting rules are hard to compute [27,188,245,283], we focused on finding approximate solutions. We have shown that if we try to optimize the satisfaction of the least satisfied agent, then our problems are hard to approximate up to any constant factor. However, for the utilitarian case we suggest good approximation algorithms. In particular, for the Monroe system we suggest a randomized algorithm that for the Borda score achieves an approximation ratio arbitrarily close to 0.715 (and much better in many real-life settings), and $(1-\frac{1}{e})$-approximation algorithm for arbitrary positional scoring function. For the Chamberlin-Courant system, we have shown a polynomial-time approximation scheme (PTAS).

In Table 5.6 we present the summary of our (in)approximability results. In Table 5.7 we present specific results regarding our approximation algorithms for the utilitarian framework. In particular, the table clearly shows that for the case of Monroe, Algorithms B and C are not much slower than Algorithm A but offer a chance of improved performance. Algorithm GM is intuitively even more appealing, but achieves this at the cost of high time complexity. For the case of Chamberlin-Courant rule, theoretical results suggest using Algorithm P (however, see below).

We have provided experimental evaluation of the algorithms for computing the winner sets both for the Monroe and Chamberlin–Courant rules. While finding solutions for these rules is computationally hard in the worst case, it turned out that in practice we can obtain very high quality solutions using very simple algorithms. Indeed, both for the Monroe and Chamberlin-Courant rules we recommend using Algorithm C (or Algorithm A on very large Monroe elections). Our experimental evaluation confirms that the algorithms work very well in case of truncated ballots.

	Algorithm	Approximation	Runtime	Reference
Monroe	A	$1 - \frac{K-1}{2(m-1)} - \frac{H_K}{K}$	Kmn	Lemma 5.5
	B	as in Algorithm A	$Kmn+O(\Phi^S)$	Lemma 5.5
	C	as in Algorithm A	$dKmn+dO(\Phi^S)$	Lemma 5.5
	GM	as in Alg. A for Borda PSF; $1 - \frac{1}{e}$ for others	$KmO(\Phi^S)$	Theorem 5.9
	R	$\frac{1}{2}(1 + \frac{K}{m} - \frac{K^2m-K^3}{m^3-m^2})$	$\frac{\lvert\log(1-\lambda)\rvert}{K\epsilon^2}O(\Phi^S)$	Lemma 5.7
	AR	0.715	$\max(\mathrm{A},\mathrm{R})$	Theorem 5.8
CC	PTAS			Corollary 5.11
	P	$1 - \frac{2\mathrm{W}(K)}{K}$	$nm\mathrm{W}(K)$	Corollary 5.10
	GM	$1 - \frac{1}{e}$	Kmn	Lu and Boutilier [188]
	C	as in Algorithm GM	$dKm(n+\log dm)$	Lu and Boutilier [188]
	R	$(1 - \frac{1}{K+1})(1 + \frac{1}{m})$	$\frac{\lvert\log(1-\lambda)\rvert}{\epsilon^2}n$	Oren [229]

Table 5.7: A summary of the algorithms studied in this chapter. The top of the table regards algorithms for Monroe's rule and the bottom for the Chamberlin–Courant rule. In column "Approximation" we give currently known approximation ratio for the algorithm under Borda PSF, on profiles with m candidates and where the goal is to select a committee of size K. Here, $O(\Phi^S) = O(n^2(K + \log n))$ is the complexity of finding a partial representation function with the algorithm of Betzler et al. [27]. $\mathrm{W}(\cdot)$ denotes Lambert's W-Function.

We believe that our results mean that (approximations of) the Monroe and Chamberlin–Courant rules can be used in practice.

Our work leads to a number of further research directions. First, it would be very interesting to find a better upper bound on the quality of solutions for the Monroe and Chamberlin–Courant systems (with Borda PSF) than the simple $n(m-1)$ bound that we use (where n is the number of voters and m is the number of candidates). We use a different approach in our randomized algorithm, but it would be much more interesting to find a deterministic algorithm that beats the approximation ratios of our algorithms. One of the ways of seeking such a bound would be to consider Monroe's rule with "exponential" Borda PSF, that is, with PSF of the form, e.g., $(2^{m-1}, 2^{m-2}, \dots, 1)$. For such PSF our approach in the proof of Lemma 5.5 would not give satisfactory results and so one would be forced to seek different attacks. In a similar vein, it would be interesting to find out if there is a PTAS for Monroe's system.

In our work, we have focused on PSFs that are strictly decreasing. It would also be interesting to study PSFs which decrease but not strictly, that is to allow some equalities. We present results for t-approval PSF's α_t, which are defined as follows: $\alpha_t(i) = 1$ if $i \leq t$ and otherwise $\alpha_t(i) = 0$, in Chapter 6.

On a more practical side, it would be interesting to develop our study of truncated ballots. Our results show that we can obtain very high approximation ratios even when voters rank only relatively few of their top candidates. For example, to achieve 90% approximation ratio for the utilitarian Monroe system in Polish parliamentary election ($K = 460, m = 6000$), each voter should rank about 8.7% of her most-preferred candidates. However, this is still over 500 candidates. It is unrealistic to expect that the voters would be willing to rank this many candidates. Thus, how should one organize Monroe-based elections in practice, to balance the amount of effort required from the voters and the quality of the results?

Finally, going back to our general capacitated disjunctive variant of the problem of selecting a collective set of items, we note that we do not have any positive results for it (the negative results, of course, carry over from the more restrictive settings). Is it possible to obtain some good approximation algorithm for the problem with no restriction on the capacities of the alternatives (in the utilitarian setting)?

Chapter 6

Approximating the MaxCover Problem with Bounded Frequencies in FPT Time

In this chapter we further study the disjunctive variant of the problem of selecting a collective set of items. We observe that this variant with the approval utilities (the utilities that come from the set $\{0, 1\}$) is equivalent to MaxCover, the problem of finding a given-size family of subsets that covers as many elements from the ground set as possible. The MaxCover problem is well-known to be NP-hard and, under standard complexity-theoretic assumptions, the best possible polynomial-time approximation algorithm for it has approximation ratio $(1 - \frac{1}{e})$. Thus, in this chapter we study exponential-time approximation algorithms for several variants of the MaxCover problem, with the focus on the variants of the problem in which frequencies of the elements are bounded, and with the focus on algorithms that run in FPT time.

6.1 Introduction

Similarly to Chapter 5, in this chapter we focus on the disjunctive variant of the problem of selecting a collective set of items; here, however, in contrast to Chapter 5 we consider the approval utilities of the agents instead of the Borda-based ones. We recall that in this variant of the problem we are given a set of agents, a set of items, and the utility profile of the agents, specifying which agents approve which items.[1] Our goal is to select K items, so that to maximize the number of agents that approve at least one of the selected items. This variant corresponds to many appealing real-life problems such as finding winners in Chamberlin–Courant parliamentary elections [47], finding recommendations for agents [188], selecting activities [75], or

[1]An agent has utility one for the approved items and utility zero for the disapproved ones.

allocating students to university courses. For a detailed discussion on the applications of the model we refer the reader to Chapters 3 and 5.

This variant of the problem of selecting a collective set of items is equivalent to the MaxCover problem. We recall that in the MaxCover problem we are given a set N of n elements, a family $\mathcal{S} = \{S_1, \dots, S_m\}$ of m subsets of N, and an integer K. The goal is to find a size-at-most-K subcollection of $\mathcal{S}$ that covers as many elements from N as possible. Indeed, we can identify the agents with the elements and the subsets with the items. Further, we can say that the subset S contains an element i if and only if agent i approves of S. This way agent i gets utility one from the selected set of items if and only if element i is covered by the set of subsets corresponding to the selected items. Consequently, selecting a set of items to maximize the number of satisfied agents (the agents that approve at least one item in the selected set) is equivalent to selecting a set of subsets to maximize the number of covered elements.

We study approximation algorithms for (and parameterized complexity of) the MaxCover problem. Apart from considering MaxCover in its full generality, we also study its two specific variants. In the first variant we assume that the frequencies of the elements are bounded, i.e., that there is some constant p such that each element appears in at most p sets. A particularly well-known special case of MaxCover with frequencies upper-bounded by 2 is the MaxVertexCover problem: We recall that in the MaxVertexCover problem we are given a graph $G = (V, E)$ and the goal is to find K vertices that, jointly, are incident to as many edges as possible (i.e., the edges are the elements to be covered and the vertices are the sets; clearly, each edge "belongs to" exactly two vertices). Nonetheless, even for the frequency upper bound 2, MaxCover is considerably more general than MaxVertexCover (e.g., the former allows two sets to have more than one element in common, which is impossible in the latter[2]). In the second variant, we assume that the frequencies of the elements are lower-bounded by some constant p.

These variants correspond to the problem of selecting a collective set of items, when the agents approve of at most (or at least) a certain number of items. We believe that these requirements are rational because:

1. We do not expect people to approve of too many candidates/resources/items. E.g., in Polish parliamentary elections we approve of three candidates.

2. Our protocols can force the agents to approve of a certain number of items.

This chapter differs from the typical approach to the design of approximation algorithms in that we do not focus on polynomial-time algorithms, but also consider exponential-time ones. For example, we are interested in FPT approximation schemes, that is, in approximation algorithms that for each desired approximation

[2]This difference may not sound particularly significant, but due to it some algorithms for MaxVertexCover (e.g., an FPT approximation scheme of Marx [196]) do not generalize to the MaxCover problem.

ratio β output a β-approximate solution in exponential time, but where the exponential growth is only with respect to the number K of sets that we allow in the solution (and where β is considered to be a constant when computing the running time). In that respect, our work is very close in spirit to the recent study of Croce and Paschos [70], who—among other results—give moderately exponential time (but not FPT-time) approximation schemes for the MaxVertexCover problem. (However, there is also an FPT-time approximation scheme for MaxVertexCover due to Marx [196].) Such exponential-time approximation algorithms are desirable because they can achieve much better approximation ratios than the polynomial-time ones, while still being significantly faster than the currently-known exact algorithms. For a more detailed review of related work we refer the reader to Section 2.3. Below we briefly describe our findings and the motivation behind our research.

We obtain the following results (unless we mention otherwise, we always consider our problems to be parameterized by K, the number of the sets allowed in the solution). First, building on the approach of Guo et al. [126], in Section 6.3 we show that the MaxCover problem with bounded frequencies is W[1]-complete. On the other hand, without the frequency upper-bound assumption, MaxCover is W[2]-hard and we show that it belongs to W[P]. We also consider several other parameters and, in particular, we show that MaxCover is W[2]-complete for the parameter that combines the number of sets we can use in the solution and the number of elements that we are allowed to leave uncovered. The core of this chapter is, however, in Section 6.4. There, we show that for each β, $0 < \beta < 1$, there is an FPT β-approximation algorithm for the MaxCover problem with bounded frequencies. On the other hand, for the case where each element appears in *at least* p out of m sets, we show that the standard MaxCover greedy approximation algorithm (i.e., one that picks one-by-one those sets that include most not-yet-covered elements) achieves approximation ratio $1 - e^{-\frac{pK}{m}}$ (for the general case, this algorithm's approximation ratio is $1 - \frac{1}{e}$). Finally, we consider a variant of the MaxCover problem where instead of maximizing the number of covered elements, we minimize the number of those that remain uncovered. We refer to this problem as the MinNonCovered problem. Under the assumption of upper-bounded frequencies, we show a randomized approximation algorithm that for each given β, $\beta > 1$, and each given probability $1 - \epsilon$, outputs in FPT time a β-approximate solution with probability at least $1 - \epsilon$ (the FPT time is with respect to K, β, and ϵ). Finally, in Section 6.5 we consider two exponential-time approximation algorithms for the unrestricted MaxCover problem. Both of these algorithms solve a part of the problem in a greedy way and a part using some exact algorithm, but they differ in the order in which they apply each of these strategies. We show a smooth transition between the running times of these algorithms and their approximation ratios.

6.2 Definitions

We assume that the reader is familiar with standard notions regarding (approximation) algorithms, computational complexity theory and parameterized complexity theory. For a brief review of these concepts we refer the reader to Chapter 2.

In this section we recall the definitions of the problems that we consider in this chapter.

Definition 6.1. *An instance $I = (N, \mathcal{S}, K)$ of the* MaxCover *problem consists of a set N of n elements, a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m subsets of N, and nonnegative integer K. The goal is to find a subcollection $\mathcal{C}$ of $\mathcal{S}$ of size at most K that maximizes $\| \bigcup_{S \in \mathcal{C}} S\|$.*

Definition 6.2. *The* MinNonCovered *problem is defined in the same way as the* MaxCover *problem, except the goal is to find a subcollection $\mathcal{C}$ such that $\|N\| - \| \bigcup_{S \in \mathcal{C}} S\|$ is minimal.*

In the decision variant of MaxCover (of MinNonCovered) we are additionally given an integer T (an integer T') and we ask if there is a collection of up to K sets from $\mathcal{S}$ that cover at least T elements (that leave at most T' elements uncovered).

In terms of the optimal solutions, MaxCover and MinNonCovered are equivalent. Nonetheless, they do differ when considered from the point of view of approximation. For example, if there were a solution that covered all the n elements, then a β-approximation algorithm for MaxCover, $0 < \beta < 1$, would be free to return a solution that covered only βn of them, but a γ-approximation algorithm for the MinNonCovered problem, $\gamma > 1$, would have to provide an optimal solution that covered all the elements.

Given an instance I of MaxCover (MinNonCovered), we say that an element e has frequency t if it appears in exactly t sets. We mostly focus on the variants of MaxCover and MinNonCovered where there is a given constant p such that each element's frequency is at most p. We refer to these problems as variants with bounded frequencies.

MaxVertexCover is a variant of MaxCover with frequencies of the elements bounded by 2, where we are given a graph $G = (V, E)$, the edges are the elements to be covered, and vertices define the sets that cover them (a vertex covers all the incident edges). SetCover and VertexCover are variants of MaxCover and MaxVertexCover, respectively, where we ask if it is possible to cover all the elements (all the edges).

For an overview of the related literature on MaxCover and MaxVertexCover we refer the reader to Section 2.3.

6.3 Worst-Case Complexity Results

We start our parameterized study of the MaxCover problem by considering its worst-case complexity. We first consider MaxCover with bounded frequencies. It follows directly from the literature that the problem is W[1]-hard, and here we show that it is, in fact, W[1]-complete (unless the frequency bound p is exactly 1; then it is optimal to simply pick the sets with highest cardinalities).

One of the standard ways of showing W[1]-membership is to give a reduction to the Short-Nondeterministic-Turing-Machine-Computation problem (shown to be W[1]-complete for parameter k by Cesati [46]).

Definition 6.3. *In the Short-Nondeterministic-Turing-Machine-Computation problem we are given a single-tape nondeterministic Turing machine M (described as a tuple including the input alphabet, the work alphabet, the set of states, the transition function, the initial state and the accepting/rejecting states), a string x over M's input alphabet, and an integer k. The question is whether there is an accepting computation of M that accepts x within k steps.*

Theorem 6.1. *For each constant p greater than 2, the* MaxCover *problem with frequencies upper-bounded by p is* W[1]*-complete (when parameterized by the number of sets in the solution).*

Proof. The hardness follows directly from the W[1]-hardness of the MaxVertexCover problem [126]. We prove membership in W[1] by reducing MaxCover with bounded frequencies to the Short-Nondeterministic-Turing-Machine-Computation problem.

Let p be some fixed constant and let $I = (N, \mathcal{S}, K, L)$ be our input instance, where N is a set of elements, $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a family of subsets of N (each element from N appears in at most p sets from $\mathcal{S}$), and K and L are two integers. This is the decision variant of the problem, thus we have L in the input; we ask if there is a collection of up to K sets from $\mathcal{S}$ that jointly cover at least L elements. W.l.o.g., we assume that $K \geq m$. We form single-tape nondeterministic Turing machine M to execute the following algorithm (on empty input string); the idea of the algorithm is to employ the standard inclusion-exclusion principle:

1. Guess the indices $i_1, \ldots, i_K$ of K sets from $\mathcal{S}$.

2. Set $T = 0$.

3. For each subset A of $\{i_1, \ldots, i_K\}$ of size up to p, do the following: If $\|A\|$ is odd, add $\|\bigcap_{i \in A} S_i\|$ to T, and otherwise subtract $\|\bigcap_{i \in A} S_i\|$ from T.

4. If $T \geq L$ then we accept and otherwise we reject.

It is easy to see that this algorithm can indeed be implemented on a single-tape nondeterministic Turing machine with a sufficiently large (but polynomially bounded) work alphabet and state space. The only issue that might require a comment is the computation of $\| \bigcap_{i \in A} S_i \|$. Since sets A contain at most p elements, we can precompute these values and store them in M's transition function.

The correctness of the algorithm follows directly from the inclusion-exclusion principle and the fact that each element appears in at most p sets:

$$\|S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_K}\| = \sum_{\ell \in [K]} \|S_{i_\ell}\| - \sum_{\substack{\ell' \in [K] \\ \ell'' \in [K] \\ \ell' \neq \ell''}} \|S_{i_{\ell'}} \cap S_{i_{\ell''}}\| + \sum_{\substack{\ell' \in [K] \\ \ell'' \in [K] \\ \ell''' \in [K] \\ \ell' \neq \ell'' \\ \ell' \neq \ell''' \\ \ell'' \neq \ell'''}} \|S_{i_{\ell'}} \cap S_{i_{\ell''}} \cap S_{i_{\ell'''}}\| - \cdots$$

In general, the above formula should include intersections of up to K sets. However, since in our case each element appears in at most p sets, the intersection of more than p sets are always empty. This shows that the algorithm is correct and concludes the proof. □

For the sake of completeness, we mention that both the unrestricted variant of the problem and the one where we put a lower bound on each element's frequency are W[2]-hard.

Theorem 6.2. *For each constant p, $p \geq 1$,* MaxCover *where each element belongs to at least p sets if* W[2]*-hard.*

Proof. To show W[2]-hardness, we give a reduction from SetCover. In the SetCover problem we ask whether there exist K subsets that cover all the elements (we give a reduction for the parameter K). Let $I = (N, \mathcal{S})$ be an input instance of SetCover. W.l.o.g., we can assume that each element from N belongs to at least one set in $\mathcal{S}$. We form an instance I' of MaxCover which is identical to I, except (a) for each $e \in N$, we modify $\mathcal{S}$ to additionally include $p-1$ copies of set $\{e\}$, and (b) we run the MaxCover algorithm asking whether the maximal number of the elements covered by K subsets is at least equal to $\|N\|$. Clearly, in I' each element belongs to at least p sets and I' is a yes-instance of MaxCover if and only if I is a yes-instance of SetCover. □

So far, we were not able to show that MaxCover (even with lower-bounded frequencies) is in W[2]. Nonetheless, it is quite easy to show that the problem belongs to W[P].

Below we present reduction from the Bounded-Nondeterministic-Turing-Machine-Computation problem, the problem that is defined similarly to the Short-Nondeterministic-Turing-Machine-Computation problem, but in addition we are also given an integer m, and we ask if machine M accepts its input within m

steps, of which at most k are nondeterministic. Cesati has shown that this problem is W[P]-complete [46].

Theorem 6.3. *For each constant p, $p \geq 1$,* MaxCover *where each element belongs to at least p sets is in* W[P] *(when parameterized by the number of sets in the solution).*

Proof. We give a reduction from MaxCover to the Bounded-Nondeterministic-Turing-Machine-Computation problem. On input $I = (N, \mathcal{S}, K, T)$, where N, $\mathcal{S}$, and K are as usual and T is the lower bound on the number of elements that we should cover, we produce a machine that on empty input executes the following algorithm:

1. It nondeterministically guesses up to K names of sets from $\mathcal{S}$ and writes these names on the tape (each name of a set from $\mathcal{S}$ is a single symbol).

2. Deterministically, for each name of the set produced in the previous step, the machine writes on the tape the names of those elements from this set that have not been written on the tape yet.

3. The machine counts the number of names of elements written on the tape. If there were at least T of them, it accepts. Otherwise it rejects.

It is easy to see that we can produce a description of such a machine in polynomial time with respect to $|I|$. Further, it is clear that its nondeterministic running time is bounded by some polynomial of $|I|$ and that it makes at most k nondeterministic steps. □

It is quite interesting to also consider MaxCover with other parameters. First, recall that for parameter T, the number of elements that we should cover, Bläser has shown that MaxCover is in FPT [32]. What can we say about parameter $T' = n - T$, i.e., the number of elements we can leave uncovered (this, in essence, means considering the MinNonCovered problem, but for the worst-case setting it is more convenient to speak of the parameter T')? In this case, the problem is immediately seen to be para-NP-complete (that is, the problem is NP-complete even for a constant value of the parameter).

Corollary 6.4. *The* MaxCover *problem is para-*NP*-complete when parameterized by the number T' of elements that can be left uncovered. This holds even if each element's frequency is upper-bounded by some constant p, $p \geq 2$.*

Proof. The following trivial reduction from SetCover suffices: Given an input instance $I = (N, \mathcal{S}, K)$, output an instance $(N, \mathcal{S}, K, 0)$, i.e., an identical one, where we require that the number of elements left uncovered is 0. Since the reduction is clearly correct and works for the constant value of the parameter, we get pare-NP-completeness. To obtain the result for upper-bounded frequencies, simply use VertexCover instead of SetCover in the reduction. □

parameter	worst-case complexity of MaxCover
K	W[2]-hard, in W[P] W[1]-complete for upper-bounded frequencies
T	FPT [32]
(K, T)	FPT [32]
T'	para-NP-complete
(K, T')	W[2]-complete

Table 6.1: Parameterized worst-case complexity results for unrestricted MaxCover and MinNonCovered. The parameters are as follows: K is the number of sets we can use in the solution, T is the number of elements we are required to cover, and $T' = n - T$ is the number of elements we can leave uncovered.

However, if we consider the joint parameter (K, T'), then the MaxCover problem becomes W[2]-complete.

Theorem 6.5. MaxCover *is* W[2]*-complete when parameterized by both the number K of sets that can be used in the solution and the number T' of elements that can be left uncovered.*

Proof. We obtain W[2]-hardness by simply observing that the reduction given in Corollary 6.4 suffices. To prove W[2]-membership, we give a reduction from MaxCover (with parameter (K, T')) to SetCover (with parameter K).

Let $I = (N, \mathcal{S}, K, T')$ be an input instance of MaxCover. We form an instance $I' = (N', \mathcal{S}', K + T')$ of SetCover as follows. Let $N' = N \cup D' \cup D''$, where $D' = \{d'_1, \ldots, d'_K\}$ and $D'' = \{d''_1, \ldots, d''_{T'}\}$. For each set $S \in \mathcal{S}$ and each $d'_i \in D'$, we set $S(d'_i) = S \cup \{d'_i\}$. We set $\mathcal{S}' = \mathcal{S}'_1 \cup \mathcal{S}'_2$, where (a) $\mathcal{S}'_1 = \{S(d'_i) : (S \in \mathcal{S}) \wedge (d'_i \in D')\}$, and (b) $\mathcal{S}'_2 = \{\{e, d''_i\} : e \in N, d''_i \in D''\}$.

It is easy to see that if I is a yes-instance of MaxCover then I' is a yes-instance of SetCover: If for I it is possible to cover $n - T'$ elements of N using K sets, then for I it is possible to (a) use K sets from $\mathcal{S}'_1$ to cover $n - T'$ elements from N and all the elements from D', and (b) use T' sets from $\mathcal{S}'_2$ to cover all the elements from D'' and the remaining T' elements from N. For the other direction, assume that I' is a yes-instance of SetCover. However, covering the elements from D' requires one to use at least K sets from $\mathcal{S}'_1$ (which correspond to the sets from $\mathcal{S}$) and covering the elements in D'' requires at least T' sets from $\mathcal{S}'_2$. Since each set from $\mathcal{S}'_2$ covers exactly one element from N, it is easy to see that if I' is a yes-instance, then it must be possible to cover at least $\|N\| - T'$ elements from N using K sets from $\mathcal{S}$. □

We summarize our worst-case complexity results in Table 6.1. Not surprisingly, using the parameter T' (i.e., in essence, considering the MinNonCovered problem) leads to higher computational complexity than using parameter T (i.e., in essence, considering the MaxCover problem). For the parameter K, the exact complexity of unrestricted MaxCover remains open.

6.4 Algorithms for the Case of Bounded Frequencies

In this section we present our approximation algorithms for the MaxCover and MinNonCovered problems, for the case where we either upper-bound or lower-bound the frequencies of the elements. We first consider the MaxCover problem, both with upper-bounded frequencies and with lower-bounded frequencies, and then move on to the MinNonCovered problem with upper-bounded frequencies.

6.4.1 The MaxCover Problem with Upper Bounded Frequencies

We will now present an FPT approximation scheme for MaxCover with upper-bounded frequencies. While Marx [196] has already shown an FPT approximation scheme for MaxVertexCover, his approach cannot be directly generalized to the MaxCover problem with bounded frequencies (although there are some similarities between the algorithms). Interestingly, our algorithm for MaxCover, when applied to the MaxVertexCover problem, is considerably faster than the algorithm of Marx [196]. We will give a brief comparison of the two algorithms after presenting our approach.

Intuitively, our algorithm works in a very simple way. Given an instance $I = (N, \mathcal{S}, K)$ of MaxCover (with frequencies bounded by some constant p) and a required approximation ratio β, the algorithm simply picks some of the sets from $\mathcal{S}$ with highest cardinalities (the exact number of these sets depends only on K, p, and β), tries all K-element subcollections of sets from this group, and returns the best one. This approach is formalized as algorithm BoundAndExplore in Figure 6.1. The following theorem explains that indeed the algorithm achieves a required approximation ratio.

Theorem 6.6. *For each instance $I = (N, \mathcal{S}, K)$ of* MaxCover *where each element from N appears in at most p sets in $\mathcal{S}$, the algorithm* BoundAndExplore *from Figure 6.1 outputs a β-approximate solution in time* $\mathrm{poly}(n, m) \cdot \binom{\frac{2pK}{(1-\beta)}+K}{K}$.

Proof. It is immediate to establish the running time of the algorithm. We show that its approximation ratio is, indeed, β.

Consider some input instance I. Let $\mathcal{C}$ be the solution returned by the algorithm BoundAndExplore and let $\mathcal{C}^*$ be some optimal solution. Let c be an arbitrary function such that for each element e such that $\exists_{S \in \mathcal{C}^*} : e \in S$, $c(e)$ is some $S \in \mathcal{C}^*$ such that $e \in S$. We refer to c as the *coverage function*. Intuitively, the coverage

```
**Parameters**:
    $(N, \mathcal{S}, K)$ — input MaxCover instance
    $p$ — bound on the number of sets each element can belong to
    $\beta$ — the required approximation ratio of the algorithm
1 $\mathcal{A} \leftarrow \lceil \frac{2pK}{(1-\beta)} + K \rceil$ sets from $\mathcal{S}$ with the highest cardinalities ;
2 **return** *$K$-element subset of $\mathcal{A}$ that covers most elements* ;
```

Figure 6.1: The algorithm BoundAndExplore for the MaxCover problem with frequency upper bounded by p.

function assigns to each element covered under $\mathcal{C}^*$ (by, possibly, many different sets) the particular set "responsible" for covering it. We say that S covers e if and only if $c(e) = S$. Let OPT denote the number of elements covered by $\mathcal{C}^*$.

We will show that $\mathcal{C}$ covers at least βOPT elements. Naturally, the reason why $\mathcal{C}$ might cover fewer elements than $\mathcal{C}^*$ is that some sets from $\mathcal{C}^*$ may not be present in $\mathcal{A}$, the set of the subsets considered by the algorithm. We will show an iterative procedure that starts with $\mathcal{C}^*$ and, step by step, replaces those members of $\mathcal{C}^*$ that are not present in $\mathcal{A}$ with the sets from $\mathcal{A}$. The idea of the proof is to show that each such replacement decreases the number of covered element by at most a small amount.

Let $\ell = \|\mathcal{C}^* \setminus \mathcal{A}\|$. Our procedure will replace the ℓ sets from $\mathcal{C}^*$ that do not appear in $\mathcal{A}$ with ℓ sets from $\mathcal{A}$. We renumber the sets so that $\mathcal{C}^* \setminus \mathcal{A} = \{S_1, \dots, S_\ell\}$. We will replace the sets $\{S_1, \dots, S_\ell\}$ with sets $\{S'_1, \dots, S'_\ell\}$ defined through the following algorithm. Assume that we have already computed sets $S'_1, \dots, S'_{i-1}$ (thus for $i = 1$ we have not yet computed anything). We take S'_i to be a set from $\mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \dots, S'_{i-1}\})$ such that the set $(\mathcal{C}^* \setminus \{S_1, \dots, S_i\}) \cup \{S'_1, \dots, S'_i\}$ covers as many elements as possible. During the i'th step of this algorithm, after we replace S_i with S'_i in the set $(\mathcal{C}^* \setminus \{S_1, \dots, S_{i-1}\}) \cup \{S'_1, \dots, S'_{i-1}\}$, we modify the coverage function as follows:

1. for each element e such that $c(e) = S_i$, we set $c(e)$ to be undefined;

2. for each element $e \in S'_i$, if $c(e)$ is undefined then we set $c(e) = S'_i$.

After replacing S_i with S'_i, it may be the case that fewer elements are covered by the resulting collection of sets. Let x_i denote the difference between the number of elements covered by $(\mathcal{C}^* \setminus \{S_1, \dots, S_i\}) \cup \{S'_1, \dots, S'_i\}$ and by $(\mathcal{C}^* \setminus \{S_1, \dots, S_{i-1}\}) \cup \{S'_1, \dots, S'_{i-1}\}$ (or 0, if by a fortunate coincidence there are more elements covered after replacing S_i with S'_i). By the construction of the set $\mathcal{A}$ and the fact that $S_i \notin \mathcal{A}$, each set from $\mathcal{A}$ contains more elements than S_i. Thus we infer that every set from $\mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \dots, S'_{i-1}\})$ must contain at least x_i elements covered by $(\mathcal{C}^* \setminus \{S_1, \dots, S_{i-1}\}) \cup \{S'_1, \dots, S'_{i-1}\}$. Indeed, if some set $S' \in \mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \dots, S'_{i-1}\})$ contained fewer than x_i elements covered by $(\mathcal{C}^* \setminus \{S_1, \dots, S_{i-1}\}) \cup \{S'_1, \dots, S'_{i-1}\}$, S'

would have to cover at least

$$\|S'\| - (x_i - 1) \geq \|S_i\| - (x_i - 1)$$

elements uncovered by $(\mathcal{C}^* \setminus \{S_1, \ldots, S_{i-1}\}) \cup \{S'_1, \ldots, S'_{i-1}\}$. But this would mean that after replacing S_i with S', the difference between the number of covered elements would be at most $(x_i - 1)$.

Let $\mathcal{C}_2^*$ denote the set obtained after the above-described ℓ iterations. Since, for each i, the set $(\mathcal{C}^* \setminus \{S_1, \ldots, S_{i-1}\}) \cup \{S'_1, \ldots, S'_{i-1}\}$ is a subset of $\mathcal{C}^* \cup \mathcal{C}_2^*$, we know that, for each i, each set from $\mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \ldots, S'_\ell\})$ (there is $\|\mathcal{A}\| - K$ such sets) must contain at least x_i elements covered by $\mathcal{C}^* \cup \mathcal{C}_2^*$ (there is at most 2OPT such elements). Since each element is contained in at most p sets, we infer that for each i, $x_i(\|\mathcal{A}\| - K) \leq 2\mathrm{OPT}p$ and, as a consequence, $x_i \leq \frac{2\mathrm{OPT}p}{\|\mathcal{A}\|-K} = \frac{2\mathrm{OPT}p(1-\beta)}{2pK}$. Thus we conclude that (recall that $\ell \leq K$):

$$\sum_{i=1}^{\ell} x_i \leq 2\mathrm{OPT}pK\frac{(1-\beta)}{2pK} = (1-\beta)\mathrm{OPT}$$

That is, after our process of replacing the sets from $\mathcal{C}^*$ that do not appear in $\mathcal{A}$ with sets from $\mathcal{A}$, at most $(1-\beta)$OPT elements fewer are covered. This means that there are K sets in $\mathcal{A}$ that together cover at least βOPT elements. Since the algorithm tries all size-K subsets of $\mathcal{A}$, it finds a solution that covers at least βOPT elements. $\square$

Our analysis is tight up to the constant factor of $\frac{3}{4}$. Below we present a family of parameters β and instances of MaxCover with upper-bounded frequencies on which our algorithm achieves approximation ratio $(\frac{3}{4} + \frac{3}{4}\beta)$

Proposition 6.7. *There is a family $\mathcal{I}$ of pairs (I, β) where I is an instance of* MaxCover *with bounded frequencies and β is a real number, $0 < \beta < 1$, such that for each $(I, \beta) \in \mathcal{I}$, if we use the algorithm* BoundAndExplore *from Figure 6.1 to find a β-approximate solution for I, it outputs an at-most $((\frac{3}{4} + \frac{3}{4}\beta)\mathrm{OPT}(I))$-approximate one.*

Proof. We describe how to construct pairs (I, β) from the set $\mathcal{I}$. We let p be the bound of the frequencies of elements in I and we let K be the number of sets that we can use in the solution. We choose p and K to be sufficiently large, and β to be sufficiently close to 1 (the exact meaning of "sufficiently large" and "sufficiently close to 1" will become clear at the end of the proof; elements of $\mathcal{I}$ differ in the particular choices of p, K, and β). We require that $\frac{1}{1-\beta}$ is an integer and that p divides K.

We now proceed with the construction of instance $I = (N, \mathcal{S}, K)$ for our choice of p, K, and β. We set $x = \frac{2pK}{(1-\beta)} + K$; x is the number of highest-cardinality sets from $\mathcal{S}$ that the algorithm BoundAndExplore will consider on instance I. By our choice of β and K, x is an integer and is divisible by p. We form N, the set of elements to be covered, to consist of two disjoint subsets, N_1 and N_2, such that $\|N_1\| = \binom{x}{p}$

and $\|N_2\| = \binom{x}{p}\frac{Kp}{x}$. We form the family $\mathcal{S}$ to consist of two subfamilies, $\mathcal{S}_1$ and $\mathcal{S}_2$, defined as follows:

1. There are x subsets in $\mathcal{S}_1$, $\mathcal{S}_1 = \{S_1, \ldots, S_x\}$. We form the sets in S_1 so that: (a) sets from $\mathcal{S}_1$ are subsets of N_1, (b) each element from N_1 belongs to exactly p different sets from $\mathcal{S}_1$, and (c) no two elements from N_1 belong to the same p sets from $\mathcal{S}_1$. Specifically, we build sets $(S_1, \ldots, S_m)$ as follows. Let f be some one-to-one mapping between elements in N_1 and p-element subsets of $[x]$. For each $e \in N_1$, e belongs exactly to the sets $S_{i_1}, \ldots, S_{i_p}$ such that $f(e) = \{i_1, \ldots, i_p\}$. Note that each set $S_i \in \mathcal{S}_1$ contains exactly $\binom{x-1}{p-1} = \binom{x}{p}\frac{p}{x}$ elements.

2. $\mathcal{S}_2$ contains K sets, each covering exactly $\binom{x}{p}\frac{p}{x}$ different elements from N_2 (and no other elements) so that no two sets from $\mathcal{S}_2$ overlap.

This completes our description of I. It is easy to see that each optimal solution for I covers exactly $K\binom{x}{p}\frac{p}{x}$ elements; each set contains exactly $\binom{x}{p}\frac{p}{x}$ elements and, there are K that are pairwise disjoint (for example the K sets in $\mathcal{S}_2$).

Nonetheless, the algorithm BoundAndExplore is free to choose any x sets from $\mathcal{S}$ to include within $\mathcal{A}$, the collection of sets from which it forms the solution, and, in particular, it is free to pick the x sets from $\mathcal{S}_1$.[3]

Let us fix some arbitrary collection $\mathcal{S}'$ of K sets from $\mathcal{S}_1$. For each j, $0 \le j \le K$, let $h(j)$ be the number of elements from N_1 that belong to exactly j sets in $\mathcal{S}'$. The number of elements covered by $\mathcal{S}'$ is exactly $K\binom{x}{p}\frac{p}{x} - \sum_{j=2}^{K}(j-1)h(j)$. How to compute $h(j)$? Using mapping f, it suffices to count the number of p-element subsets of $[x]$ that contain the indices of exactly j sets from $\mathcal{S}'$. In effect, we have $h(j) = \binom{K}{j}\binom{x-K}{p-j}$. We upper bound the number of sets covered by $\mathcal{S}'$ with:

$$K\binom{x}{p}\frac{p}{x} - h(2) = K\binom{x}{p}\frac{p}{x} - \binom{K}{2}\binom{x-K}{p-2}.$$

Consequently, on instance I the algorithm achieves the following approximation ratio $\frac{K\binom{x}{p}\frac{p}{x} - \binom{K}{2}\binom{x-K}{p-2}}{K\binom{x}{p}\frac{p}{x}}$, which is equal to:

$$1 - \frac{\binom{K}{2}\binom{x-K}{p-2}}{K\binom{x}{p}\frac{p}{x}} = 1 - \frac{\binom{K}{2}\binom{x-K}{p}\frac{p(p-1)}{(x-K-p+2)(x-K-p+1)}}{K\binom{x}{p}\frac{p}{x}}.$$

Now, if x is large in comparison with p and K (which happens for sufficiently large β), then $\frac{\binom{x-K}{p}}{\binom{x}{p}} \approx 1$. Also, for sufficiently large x and p (and for $x \gg p, K$) we have

[3] We could also ensure that each set in $\mathcal{S}_1$ contained one of $\frac{x}{p}$ additional elements, forcing the algorithm to pick exactly the sets from $\mathcal{S}_1$, but that would obscure the presentation of our argument.

$\frac{p}{x-K-p+2} \approx \frac{p}{x}$ and $\frac{p-1}{x-K-p+1} \approx \frac{p}{x}$. Finally, for sufficiently large K we have $\binom{K}{2} \approx \frac{K^2}{2}$. Thus, for large values of β, K, and p, we can approximate the above ratio with the following expression:

$$1 - \frac{\frac{K^2}{2} \cdot \frac{p^2}{x^2}}{K\frac{p}{x}} = 1 - \frac{1}{2} \cdot \frac{Kp}{\frac{2pK}{(1-\beta)} + K} \approx 1 - \frac{1}{2} \cdot \frac{Kp}{\frac{2pK}{(1-\beta)}} = 1 - \frac{1}{4} \cdot (1-\beta) = \frac{3}{4} + \frac{3}{4}\beta.$$

This completes our argument. □

Let us now compare our algorithm to that of Marx [196] for the case of MaxVertexCover. Briefly put, the idea behind Marx's algorithm is as follows: Consider vertices in the order of nonincreasing degrees. If the degree of the vertex with the highest degree is large enough, then K vertices with the highest degrees already cover sufficiently many edges to give a desired approximate solution. If the highest degree is not large enough, then there is an exact, color-coding based, FPT algorithm that solves the problem optimally. Our algorithm is similar in the sense that we also focus on a group of sets with highest cardinalities (sets' cardinalities in MaxCover correspond to vertex degrees in MaxVertexCover). However, instead of simply picking K largest ones, we make a careful decision as to which exactly to take.[4] Further, our algorithm has a better running time than that of Marx. To achieve approximation ratio β, the algorithm presented by Marx has running time at least $\Omega((\frac{K^3}{1-\beta})^{(\frac{K^3}{1-\beta})})$. For us, the exponential factor in the running time is $\binom{\frac{2pK}{(1-\beta)}+K}{K}$. On the other hand, we should point out that Marx's algorithm's running time stems mostly from the exact part and the algorithm given there is interesting in its own right.

6.4.2 The MaxCover Problem with Lower-Bounded Frequencies

Let us now move on to the case of MaxCover with lower-bounded frequencies. It turns out that in this case the standard greedy algorithm, given here as the algorithm Greedy in Figure 6.2, can—for appropriate inputs—achieve a better approximation ratio than in the unrestricted case.

Theorem 6.8. *The algorithm* Greedy *from Figure 6.2 is a polynomial-time* $(1 - e^{-\frac{pK}{m}})$*-approximation algorithm for the* MaxCover *problem with frequency lower bounded by* p*, on instances with* m *elements where we can pick up to* K *sets.*

[4]Indeed, it is possible to build an example where picking sets with highest cardinalities would not work. This trick works in Marx's algorithm because he considers graphs and, thus, can bound the negative effect of covering the same element by different sets; in the MaxCover problem this seems difficult to do.

```
Parameters:
  $(N, \mathcal{S}, K)$ — input MaxCover instance
  $p$ — lower bound on the number of the sets each element belongs to
$C = \{\}$;
for $i \leftarrow 1$ to $K$ do
    $Cov \leftarrow \{e \in N : \exists_{S \in C} e \in S\}$ ;
    $S_{\mathrm{best}^{(K)}} \leftarrow \mathrm{argmax}_{S \in \{S_1, \ldots, S_m\} \setminus C} \ \{e \in N \setminus Cov : e \in S\}\|$;
    $C \leftarrow C \cup \{S_{\mathrm{best}^{(K)}}\}$
return $C$
```

Figure 6.2: The algorithm Greedy for the MaxCover problem with frequency lower bounded by p.

Proof. The algorithm clearly runs in polynomial time and so we show it's approximation ratio. Let $I = (N, \mathcal{S}, K)$ be an input instance of MaxCover and let p be an integer such that each element from N belongs to at least p sets from $\mathcal{S}$.

We prove by induction that for each i, $0 \leq i \leq K$, after the i'th iteration of the algorithm's main loop, the number of uncovered elements is at most $n(1 - \frac{p}{m})^i$. Naturally, for $i = 0$ the number of uncovered elements is exactly n, the total number of elements. Suppose that the inductive assumption holds for some $(i-1)$, $1 \leq i < K$ and let x be the number of elements still uncovered after the $(i-1)$-th iteration (by the inductive assumption, we have $x \leq n(1 - \frac{p}{m})^{i-1}$). Since each element belongs to at least p sets and neither of the sets containing the uncovered elements is yet selected, by the pigeonhole principle there is a not-yet-selected set that contains at least $\lceil x\frac{p}{m} \rceil$ of the uncovered elements. In consequence, the number of elements still uncovered after the i-th iteration is at most:

$$x - x\frac{p}{m} = x\left(1 - \frac{p}{m}\right) \leq n\left(1 - \frac{p}{m}\right)^i.$$

Thus after K iterations the number of uncovered elements is at most:

$$n\left(1 - \frac{p}{m}\right)^K = n\left(1 - \frac{p}{m}\right)^{\frac{m}{p} \cdot \frac{pK}{m}} \leq ne^{-\frac{pK}{m}}.$$

Since the number of covered elements in the optimal solution is at most n, the algorithm's approximation ratio is $(1 - e^{-\frac{pK}{m}})$. $\square$

Naturally, the standard approximation ratio of $(1 - e^{-1})$ of the greedy algorithm still applies and we get the following corollary.

Corollary 6.9. *The algorithm* Greedy *from Figure 6.2 gives approximation guarantee of* $(1 - e^{-\max(\frac{pK}{m}, 1)})$.

The analysis given in Theorem 6.8 is tight. Below we present a family of instances on which the algorithm reaches exactly the promised approximation ratio.

Proposition 6.10. *For each α, $\alpha \geq 1$, there is an instance $I(\alpha)$ of* MaxCover *(with m sets. element frequency lower-bounded by p, K sets to use, and $\frac{pK}{m} = \alpha$) such that on input $I(\alpha)$, the algorithm* Greedy *from Figure 6.2 achieves approximation ratio no better than $(1 - e^{-\frac{pK}{m}})$.*

Proof. Let us fix some α, $\alpha > 1$. We choose integers p, K, and m so that: (a) $p = \frac{\alpha m}{K}$, (b) $m \gg K$ (and, thus, $p \gg K$), and (c) p, m, and K are sufficiently large (the exact meaning of "sufficiently large" will become clear at the end of the proof).

We form instance $I(\alpha) = (N, \mathcal{S}, K)$ as follows. We let $N = N_1 \cup \cdots \cup N_K$, where $N_1, \ldots, N_K$ are pairwise-disjoint sets, each of cardinality $\binom{m-K}{p-1}$ (thus $\|N\| = K\binom{m-K}{p-1}$). The family $\mathcal{S}$ consists of two subfamilies, $\mathcal{S}_1$ and $\mathcal{S}_2$:

1. $\mathcal{S}_1$ consists of $m - K$ sets, $S_1, \ldots, S_{m-K}$, constructed as follows. For each i, $1 \leq i \leq K$, let f_i be some one-to-one mapping from N_i to $(p-1)$-element subsets of $[m-K]$. For each i, $1 \leq i \leq K$, if $e \in N_i$ and $f_i(e) = \{j_1, \ldots, j_{p-1}\}$ then we include e in sets $S_{j_1}, S_{j_2}, \ldots, S_{j_{p-1}}$. Note that for each S_ℓ in $\mathcal{S}_2$, $\|S_\ell\| = K\binom{m-K}{p-2}$; for each i, $1 \leq i \leq K$, S_ℓ contains $\binom{m-K}{p-2}$ elements from N_i; to see this, it suffices to count how many $(p-1)$-elements subsets of $[m-K]$ there are that contain j.

2. $\mathcal{S}_2 = \{N_1, \ldots, N_K\}$.

Note that, by our construction, each element from N belongs to exactly p sets from $\mathcal{S}$ ($p-1$ from $\mathcal{S}_1$ and one from $\mathcal{S}_2$).

Naturally, the K disjoint sets from $\mathcal{S}_2$ form the optimal solution and cover all the elements. We will now analyze the operation of the algorithm Greedy on input $I(\alpha)$.

We claim that the algorithm Greedy will select sets from $\mathcal{S}_1$ only. We show this by induction. Fix some ℓ, $1 \leq \ell \leq K$, and suppose that until the beginning of the ℓ'th iteration the algorithm chose sets from $\mathcal{S}_1$ only. This means that, for each i, $1 \leq i \leq K$, each set N_i contains exactly $\binom{m-K-\ell}{p-1}$ uncovered elements. Why is this the case? Assume that the algorithm selected sets $S_{j_1}, \ldots, S_{j_\ell}$. An element $e \in N_i$ is uncovered if and only if $f_i(e) \cap \{j_1, \ldots, j_\ell\} = \emptyset$; $\binom{m-K-\ell}{p-1}$ is the number of $(p-1)$-element subsets of $[m-K]$ that do not contain any members of $\{j_1, \ldots, j_\ell\}$. So, if in the ℓ'th iteration the algorithm chooses some set from $\mathcal{S}_2$, it would cover these additional $\binom{m-K-\ell}{p-1}$ elements. On the other hand, if it chose a set from $\mathcal{S}_1$, it would additionally cover Kx elements, where $x = \binom{m-K-\ell}{p-1} - \binom{m-K-\ell-1}{p-1}$. By our choice, we have $pK > m$ and, thus, $K > \frac{m-K}{p-1}$. We can now see that the following holds:

$$
\begin{aligned}
Kx &= K\left(\binom{m-K-\ell}{p-1} - \binom{m-K-\ell-1}{p-1}\right) = K\binom{m-K-\ell-1}{p-2} \\
&= \frac{K(p-1)}{m-K-\ell}\binom{m-K-\ell}{p-1} \geq K\frac{p-1}{m-K}\binom{m-K-\ell}{p-1} > \binom{m-K-\ell}{p-1}.
\end{aligned}
$$

That is, in the ℓ'th iteration the algorithm Greedy picks a set from $\mathcal{S}_1$. This proves our claim.

Let us now assess the approximation ratio the algorithm Greedy achieves on $I(\alpha)$. By the above reasoning, we know that it leaves $\binom{m-2K}{p-1}$ uncovered elements in each N_i, $1 \leq i \leq K$. Thus the fraction of the uncovered elements is bounded by the following expression (see some explanation below):

$$\begin{aligned}
\frac{K\binom{m-2K}{p-1}}{K\binom{m-K}{p-1}} &= \frac{(m-2K)!(m-p-K+1)!}{(m-K)!(m-p-2K+1)!} \\
&= \frac{(m-p-K+1)(m-p-K)\dots(m-p-2K)}{(m-K)(m-K-1)\dots(m-2K+1)} \\
&\geq \left(\frac{m-2K-p}{m-2K+1}\right)^K = \left(1 - \frac{p+1}{m-2K+1}\right)^K \approx e^{-\frac{pK}{m}}.
\end{aligned}$$

The first inequality holds by iterative application of the simple observation that if $1 \leq x \leq y$ then $\frac{x-1}{y-1} \leq \frac{x}{y}$. To obtain the final estimate, we observe that for sufficiently large p and m (where $m \gg K$), we have $\frac{p+1}{m-2K+1} \approx \frac{p}{m} = \frac{\alpha}{K}$. For sufficiently large K, $(1-\frac{\alpha}{K})^K \approx e^{-\alpha} = e^{-\frac{pK}{m}}$ (by the fact that $p = \frac{\alpha m}{K}$). Since the optimal solution covers all the elements, we have that the algorithm Greedy on input $I(\alpha)$ achieves approximation ratio no better than $1 - e^{-\frac{pK}{m}}$. □

Theorem 6.8 has some interesting implications. Let us consider a version of the MaxCover problem in which the ratio $\frac{p}{m}$ between the frequency lower bound p and the number of sets m is constant. This problems arises, e.g., if we use approval-based variant of the Chamberlin-Courant's election system with a requirement that each voter must approve at least some constant fraction (e.g., 10%) of the candidates. There exists a polynomial-time approximation scheme (PTAS) for this version of the problem.

Definition 6.4. *For each α, $0 < \alpha \leq 1$, let α-MaxCover be a variant of MaxCover for instances that satisfy the following conditions: If p is a lower-bound on the frequencies of the elements and there are m sets, then $\frac{p}{m} \geq \alpha$.*

Theorem 6.11. *For each α, $0 < \alpha \leq 1$, there is a PTAS for α-MaxCover.*

Proof. Fix some α, $0 < \alpha \leq 1$. Let $I = (N, \mathcal{S}, K)$ be input instance of α-MaxCover and let β be our desired approximation ratio. We let m be the number of set in $\mathcal{S}$ and p be the lower bound on element frequencies. By definition, we have $\frac{p}{m} \geq \alpha$. If $K > -\frac{m}{p}\ln(1-\beta)$ then we can run the algorithm Greedy from Figure 6.2 and, by Theorem 6.8, we obtain approximation ratio β. Otherwise, K is bounded by a constant and enumerating all K-element subsets of $\mathcal{S}$ gives a polynomial exact algorithm for the problem. □

The exact complexity of α-MaxCover is quite interesting. Using the algorithm Greedy, we show that it belongs to the second level of Kintala and Fisher's β-hierarchy of limited nondeterminism [165]. In effect, it is unlikely that the problem is NP-complete.

Definition 6.5 (Kintala and Fisher [165]). *For each positive integer k, β^k is the class of decision problems that can be solved in polynomial time, using additionally at most $O(\log^k n)$ nondeterministic bits (where n is the size of the input instance).*

It is easy to see that β^1 is simply the class of problems solvable in polynomial time; we can simulate $O(\log n)$ bits of nondeterminism by trying all possible combinations. However, class β^2 appears to be greater than P but smaller than NP (of course, since we do not know if P $\neq$ NP, this is only a conjecture).

Theorem 6.12. *For each α, $0 < \alpha < 1$, the decision variant of α-MaxCover is in β^2.*

Proof. Fix some α, $0 < \alpha < 1$. We will give a β^2-algorithm for α-MaxCover. Let $I = (N, \mathcal{S}, K, T)$ be an instance of α-MaxCover (recall that T is the number of elements we are required to cover). We let p be the lower bound on elements' frequencies in I, we let $m = \|\mathcal{S}\|$, and we let $n = \|N\|$. By definition, we have $\frac{p}{m} \geq \alpha$. W.l.o.g., we assume that $\|I\| \geq n + m$.

Our algorithm works as follows. If $K > \frac{1}{\alpha}\ln(n)$ then we run the algorithm Greedy and output its solution. Otherwise, we guess K names of the sets from $\mathcal{S}$ and check if these sets cover at least T elements. If so, we accept and otherwise we reject on this computation path.

First, it is clear that the algorithm uses at most $O(\log^2 |I|)$ nondeterministic bits. We execute the nondeterministic part of the algorithm only if $K < \frac{1}{\alpha}\ln(n) \leq \frac{1}{\alpha}\ln|I|$ and each set's name requires at most $\log m \leq \log|I|$ bits. Altogether, we use at most $O(\log^2 |I|)$ bits of nondeterminism.

Second, we need to show the correctness of the algorithm. Clearly, if the algorithm uses the nondeterministic part then certainly it finds an optimal solution. Consider then that the algorithm uses the deterministic part, based on the algorithm Greedy. In this case we know that $K > \frac{1}{\alpha}\ln(n)$. Thus, the approximation ratio of the algorithm Greedy is greater than: $1 - e^{-\alpha K} > (1 - e^{-\ln n}) = 1 - \frac{1}{n}$. That is, the algorithm returns a solution that covers more than $\mathrm{OPT}(1 - \frac{1}{n})$ elements and, since $\mathrm{OPT} \leq n$ and the number of covered elements is integer, the algorithm must find an optimal solution. □

6.4.3 The MinNonCovered Problem

In this section we consider the MinNonCovered problem, that is, a version of MaxCover where the goal is to minimize the number of elements left uncovered.

```
Parameters:
  $(N, \mathcal{S}, K)$ — input MinNonCovered instance
  $p$ — bound on the number of sets each element can belong to
  $\beta$ — the required approximation ratio of the algorithm
  $\epsilon$ — the allowed probability of achieving worse than $\beta$ approximation ratio

RecursiveSearch($s$, $partial$):
    if $s = 0$ then
        return $partial$;
    else
        $e \leftarrow$ randomly select element not-yet covered by
            $partial$;
        $\mathrm{best}^{(K)} \leftarrow \emptyset$;
        foreach $S \in \mathcal{S}$ such that $e \in S$ do
            $sol \leftarrow$ RecursiveSearch($(s-1)$, $partial \cup \{S\}$);
            if $sol$ is better than $\mathrm{best}^{(K)}$ then
                $\mathrm{best}^{(K)} \leftarrow sol$;
        return $\mathrm{best}^{(K)}$;

Main():
    $\mathrm{best}^{(K)} = \emptyset$;
    for $i \leftarrow 1$ to $\left\lceil -\ln \epsilon / \left(\frac{\beta-1}{\beta}\right)^K \right\rceil$ do
        $sol$ = RecursiveSearch($K$, $\emptyset$);
        if $sol$ is better than $\mathrm{best}^{(K)}$ then
            $\mathrm{best}^{(K)} \leftarrow sol$;
    return $\mathrm{best}^{(K)}$;
```

Figure 6.3: The algorithm RandRecursiveSearch for the MinNonCovered problem with frequency upper bounded by p.

In this case we give a randomized FPT approximation scheme (presented as the algorithm RandRecursiveSearch in Figure 6.3).

Intuitively, the idea behind our approach is to extend a simple bounded-search-tree algorithm for SetCover with upper-bounded frequencies to the case of MaxCover. An FPT algorithm for SetCover with frequencies upper-bounded by some constant p could work recursively as follows: If there still is some uncovered element e, then nondeterministically guess one of the at-most-p sets that contain e and recursively solve the smaller problem. The recursion tree would have at most K levels and p^K leaves. The same approach does not work directly for MaxCover because we do not know which element e to pick (in SetCover the choice is irrelevant because we have to cover all the elements). However, it turns out that if we choose e randomly then, in expectation, we achieve a good result.

Theorem 6.13. *The algorithm* RandRecursiveSearch *from Figure 6.3 outputs a β-approximate solution for the* MinNonCovered *problem with probability* $(1-\epsilon)$.

The time complexity of the algorithm is

$$\mathrm{poly}(n,m) \cdot \left\lceil -\ln\epsilon / \left(\frac{\beta-1}{\beta}\right)^K \right\rceil \cdot p^K$$

.

Proof. Let $I = (N, \mathcal{S}, K)$ be our input instance of the MinNonCovered problem and fix some β, $\beta > 1$, and ϵ, $0 < \epsilon < 1$. Each element from N appears in at most p sets from $\mathcal{S}$.

By p_s we denote the probability that a single invocation of the function `RecursiveSearch` (from the `Main` function) returns a β-approximate solution. We will first show that p_s is at least $\left(\frac{\beta-1}{\beta}\right)^K$, and then we will invoke the standard argument that if we make $\left\lceil \frac{-\ln\epsilon}{p_s} \right\rceil$ calls to `RecursiveSearch`, then taking the best output gives a β-approximate solution with probability $(1-\epsilon)$.

Let $\mathcal{C}^*$ be some optimal solution for I, let $N^* \subseteq N$ be the set of elements covered by $\mathcal{C}^*$, and let $U^* = N \setminus N^*$ be the set of the remaining, uncovered elements. Consider a single call to `RecursiveSearch` from the "for" loop within the function `Main`. Let Ev denote the event that during such a call, at the beginning of each recursive call, at least a $\frac{\beta-1}{\beta}$ fraction of the elements not covered by the constructed solution (i.e., the solution denoted *partial* in the algorithm) belongs to N^*. Note that if the complementary event, denoted $\overline{Ev}$, occurs, then `RecursiveSearch` definitely returns a β-approximate solution. Why is this the case? Consider some tree of recursive invocations of `RecursiveSearch`, and some invocation of `RecursiveSearch` within this tree. Let X be the number of elements not covered by *partial* at the beginning of this invocation. If at most $\frac{\beta-1}{\beta}X$ of the not-covered elements belong to N^*, then—of course—the remaining at least $\frac{1}{\beta}X$ of them belong to U^*. In other words, then we have $\frac{1}{\beta}X \leq \|U^*\|$ and, equivalently, $X \leq \beta\|U^*\|$. This means that *partial* already is a β-approximate solution, and so the solution returned by the current invocation of `RecursiveSearch` will be β-approximate as well. (Naturally, the same applies to the solution returned at the root of the recursion tree.)

Now, consider the following random process $\mathcal{P}$. (Intuitively, $\mathcal{P}$ models a particular branch of the `RecursiveSearch` recursion tree.) We start from the set N' of all the elements, $N' = N$, and in each of the next K steps we execute the following procedure: We randomly select an element e from N' and if e belongs to N^*, we remove from N' all the elements covered by the first[5] set from $\mathcal{C}^*$ that covers e. Let p_{opt} be the probability that a call to `RecursiveSearch` (within `Main`) finds an optimal solution for I, and let $p_{\mathrm{opt}|Ev}$ be the same probability, but under the condition that Ev takes place. It is easy to see that p_{opt} is greater or equal than the probability that in each step $\mathcal{P}$ picks an element from N^*. Let p_{hit} be the probability that in each step $\mathcal{P}$

[5]We assume that the sets in $\mathcal{C}^*$ are ordered in some arbitrary way.

```
Parameters:
  $(N, \mathcal{S}, K)$ — input MaxCover instance
  $X$ — the parameter of the algorithm
  $\mathcal{A}(\cdot)$ — an exact algorithm for MaxCover (returns the set of sets to be used in the cover)
1 $C = \{\}$;
2 for $i \leftarrow 1$ to $X$ do
3     $Cov \leftarrow \{e \in N : \exists_{S \in C} e \in S\}$ ;
4     $S_{\mathrm{best}^{(K)}} \leftarrow \mathrm{argmax}_{S \in \{S_1, \dots, S_m\} \setminus C} \|\{e \in N \setminus Cov : e \in S\}\|$;
5     $C \leftarrow C \cup \{S_{\mathrm{best}^{(K)}}\}$
6 $uCov \leftarrow N \setminus \{e \in N : \exists_{S \in C} e \in S\}$ ;
7 $C' \leftarrow \mathcal{A}(uCov, (K - X), S \setminus C)$ ;
8 return $C \cup C'$
```

Figure 6.4: An approximation algorithm GreedyAndExpo for the unrestricted MaxCover problem.

picks an element from N^*, under the condition that at the beginning of every step more than $\frac{(\beta-1)}{\beta}$ fraction of the elements in N' belong to N^*. Again, it is easy to see that $p_{\mathrm{opt}|Ev} \geq p_{hit}$. Further, it is immediate to see that $p_{hit} \geq \left(\frac{\beta-1}{\beta}\right)^K$.

Altogether, combining all the above findings, we know that the probability that `RecursiveSearch` returns a β-approximate solution is at most:

$$p_s \geq \mathrm{P}(\overline{Ev}) + \mathrm{P}(Ev)p_{\mathrm{opt}|Ev} \geq p_{\mathrm{opt}|Ev} \geq \left(\frac{\beta - 1}{\beta}\right)^K.$$

(That is, either the event Ev does not take place and `RecursiveSearch` definitely returns a β-approximate solution, or Ev does occur, and then we lower-bound the probability of finding a β-approximate solution by the probability of finding the optimal one.)

To conclude, the probability of finding a β-approximate solution in one of the $x = \left\lceil -\ln \epsilon / \left(\frac{\beta-1}{\beta}\right)^K \right\rceil$ independent invocations of `RecursiveSearch` from `Main` is at least:

$$1 - \left(1 - \left(\frac{\beta - 1}{\beta}\right)^K\right)^x \geq 1 - e^{\ln \epsilon} = 1 - \epsilon.$$

Establishing the running time of the algorithm is immediate, and so the proof is complete. □

The algorithm RandRecursiveSearch is very useful, especially in conjunction with the algorithm BoundAndExplore. The former one has to provide a very good solution if it is possible to cover almost all the elements and the latter one has to provide a very good solution if in every solution many elements must be left uncovered.

6.5 Algorithms for the Unrestricted Variant

So far we have focused on the MAXCOVER problem where element frequencies were either upper- or lower-bounded. Now we consider the completely unrestricted variant of the problem. In this case we give exponential-time approximation schemes that, nonetheless, are not FPT.

The main idea, which is similar to that of Cygan et. al [73] and of Croce and Paschos [70], is to solve part of the problem using an exact algorithm and to solve the remaining part using the greedy algorithm (i.e., the algorithm GREEDY from Figure 6.2). There are two possible ways in which this idea can be implemented: Either we can first run the exact algorithm and then solve the remaining part of the instance using the greedy algorithm, or the other way round. We consider both approaches, though a variant of the "brute-force-first-then-greedy" approach appears to be superior (at least as long as we do not have exact algorithms that are significantly faster than a brute-force approach).

We start with the analysis of the algorithm GREEDYANDEXPO, which first runs the greedy part and then completes it using an exact algorithm.

Theorem 6.14. *Let $\mathcal{A}$ be an exact algorithm for the* MAXCOVER *problem with time complexity $f(K, n, m)$. For each instance $I = (N, \mathcal{S}, K)$ of* MAXCOVER *and for each X, $0 \leq X \leq K$, the algorithm* GREEDYANDEXPO *from Figure 6.4 returns a $\left(1 - \frac{X}{K}e^{-\frac{X}{K}}\right)$-approximate solution for I and runs in time $f(K-X, n, m)+$* $\mathrm{poly}(n, K, m)$*).*

Proof. Establishing the running time of the algorithm is immediate and, thus, below we focus on showing the approximation ratio.

Let $I = (N, \mathcal{S}, K)$ be an instance of MAXCOVER and let X be an integer, $1 \leq X \leq K$. We rename the elements in $\mathcal{S}$ so that $\mathcal{S} = \{S_1, \ldots, S_m\}$ and $S_1, \ldots S_X$ are the consecutive elements selected in the first, greedy, "for loop" in Figure 6.4. For each i, $1 \leq i \leq m$, let $c_i = \|S_i \setminus (S_1 \cup \cdots \cup S_{i-1})\|$. Let N_{OPT} denote the set of elements covered by some optimal solution and set $\mathrm{OPT} = \|N_{\mathrm{OPT}}\|$. Let Cov_i denote the set $S_1 \cup \cdots \cup S_{i-1}$. (That is, Cov_i is the set of elements in the variable Cov in the Figure 6.4 right before executing the i'th iteration of the "for loop". Of course, $Cov_1 = \emptyset$.) Naturally, for each i, $1 \leq i \leq m$, we have $\|Cov_i\| = \sum_{j=1}^{i-1} c_i$.

We claim that for each i, $1 \leq i \leq X$, there exist $(K-i)$ sets from $\mathcal{S} \setminus \{S_1, \ldots S_{i-1}\}$ that cover at least $\frac{K-i}{K}$ fraction of the elements from $N_{\mathrm{OPT}} \setminus Cov_{i-1}$. Why is this the case? First, note that there are some K sets from $\mathcal{S} \setminus \{S_1, \ldots S_{i-1}\}$ that cover $N_{\mathrm{OPT}} \setminus Cov_{i-1}$ (it suffices to take the K sets from some optimal solution, if need be, replace those that belong to $\{S_1, \ldots, S_{i-1}\}$ with some arbitrarily chosen ones from $\mathcal{S} \setminus \{S_1, \ldots, S_{i-1}\}$). Let $Q_1, \ldots, Q_K$ be these K sets. Consider some arbitrary assignment of the elements from $N_{\mathrm{OPT}} \setminus Cov_{i-1}$ to the sets $Q_1, \ldots, Q_K$, such that each element is assigned to exactly one set. Further, consider an ordering of these sets according to the increasing number of assigned elements. If the i'th set in the ordering is assigned at most fraction $\frac{1}{K}$ of the elements, than each of the sets preceding the i'th one in the ordering also is assigned at most fraction $\frac{1}{K}$ of the elements. In consequence, the last $(K-i)$ sets from the ordering cover at least fraction $\frac{K-i}{K}$ of the elements. On the other hand, if the i'th set in the order is assigned more than fraction $\frac{1}{K}$ of the elements then the following sets also are and, once again, the last $(K-i)$ elements cover at least fraction $\frac{K-i}{K}$ of the elements.

In consequence, we see that for each i, $1 \le i \le X$, $c_i \ge \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1} c_j)$. The reason is that since there are $K-i$ sets among $\mathcal{S} \setminus \{S_1, \ldots, S_{i-1}\}$ that cover fraction $\frac{K-i}{K}$ of elements from $N_{\mathrm{OPT}} \setminus Cov_i$, at least one of them must cover $\frac{1}{K}(\mathrm{OPT} - \| Cov_i \|)$. S_i is chosen as a set that covers most sets from $N - Cov_i$. It covers c_i elements from $N - Cov_i$, and, thus, $c_i \ge \frac{1}{K}(\mathrm{OPT} - \| Cov_i \|) = \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1})$.

We can now proceed with computing the algorithm's approximation ratio. By the above reasoning, we observe that the solution provided by the algorithm GreedyAndExpo covers at least $c = \sum_{i=1}^{X} c_i + \frac{K-X}{K}(\mathrm{OPT} - \sum_{i=1}^{X} c_i) = \frac{X}{K}\sum_{i=1}^{X} c_i + \frac{K-X}{K}\mathrm{OPT}$. Now, we assess the minimal value of $\sum_{i=1}^{X} c_i$. Minimization of $\sum_{i=1}^{X} c_i$ can be viewed as a linear programming task with the following constraints: for each i, $1 \le i \le X$, $c_i \ge \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1} c_j)$. Since we have X variables and X constraints, we know that the minimum is achieved when each constraint is satisfied with equality (see, e.g., [293]). Thus a solution to our linear program consists of values $c_{1,\min}, \ldots, c_{X,\min}$ that, for each i, $1 \le i \le X$, satisfy $c_{i,\min} = \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1} c_{j,\min})$. By induction, we show that for each i, $1 \le i \le X$, $c_{i,\min} = \frac{1}{K}\left(\frac{K-1}{K}\right)^{i-1}\mathrm{OPT}$. Indeed, the claim is true for $i = 1$:

$$c_{1,\min} = \frac{1}{K}\mathrm{OPT}$$

Now, assuming that $c_{i,\min} = \frac{1}{K}\left(\frac{K-1}{K}\right)^{i-1}\mathrm{OPT}$, we calculate $c_{(i+1),\min}$:

$$\begin{aligned}
c_{(i+1),\min} &= \frac{1}{K}\left(\mathrm{OPT} - \sum_{j=1}^{i} c_{j,\min}\right) \\
&= \frac{1}{K}\mathrm{OPT}\left(1 - \frac{1}{K}\sum_{j=1}^{i}\left(\frac{K-1}{K}\right)^{j-1}\right) \\
&= \frac{1}{K}\mathrm{OPT}\left(1 - \frac{1}{K} \cdot \frac{1 - \left(\frac{K-1}{K}\right)^{i}}{1 - \left(\frac{K-1}{K}\right)}\right) \\
&= \frac{1}{K}\mathrm{OPT}\left(1 - \left(1 - \left(\frac{K-1}{K}\right)^{i}\right)\right)
\end{aligned}$$

```
    Parameters:
     $(N, \mathcal{S}, K)$ — input MaxCover instance
     $X$ — the parameter of the algorithm
1 $C = \{\}$;
2 $C_{best} = \{\}$;
3 foreach $(K - X)$-element subset $C$ of $\mathcal{S}$ do
4     for $i \leftarrow (K - X + 1)$ to $K$ do
5         $Cov \leftarrow \{e \in N : \exists_{S \in C} e \in S\}$ ;
6         $S_{\mathrm{best}^{(K)}} \leftarrow \mathrm{argmax}_{S \in \{S_1, \dots, S_m\} \setminus C} \{e \in N \setminus Cov : e \in S\}\|$;
7         $C \leftarrow C \cup \{S_{\mathrm{best}^{(K)}}\}$
8     $C_{best} \leftarrow$ better solution among $C_{best}$ and $C$;
9 return $C_{best}$
```

Figure 6.5: The approximation algorithm ExpoAndGreedy for the MaxCover problem.

$$= \frac{1}{K}\mathrm{OPT}\left(\frac{K-1}{K}\right)^{i}.$$

Thus we can lower-bound the number of elements covered by the algorithm GreedyAndExpo as follows:

$$\begin{aligned}
c &= \frac{X}{K}\sum_{i=1}^{X} c_i + \frac{K-X}{K}\mathrm{OPT} = \mathrm{OPT}\left(\frac{X}{K^2}\sum_{i=1}^{X}\left(\frac{K-1}{K}\right)^{i-1} + \frac{K-X}{K}\right) \\
&= \mathrm{OPT}\left(\frac{X}{K^2}\cdot\frac{1-\left(\frac{K-1}{K}\right)^{X}}{1-\left(\frac{K-1}{K}\right)} + \frac{K-X}{K}\right) = \mathrm{OPT}\left(\frac{X}{K}\left(1-\left(\frac{K-1}{K}\right)^{X}\right) + \frac{K-X}{K}\right) \\
&\geq \mathrm{OPT}\left(1 - \frac{X}{K}e^{-\frac{X}{K}}\right).
\end{aligned}$$

This completes the proof. □

The idea of the proof of Theorem 6.14 is similar to the algorithm of Cygan et. al [73] for the problem of weighted set cover. Theorem 6.14 would give a good-quality result provided we knew an optimal algorithm with better complexity than the exhaustive search. Otherwise, we can obtain very good results using algorithm ExpoAndGreedy, shown in Figure 6.5, which first runs a brute-force approach and completes it using the greedy algorithm.

Theorem 6.15. *For each instance $I = (N, \mathcal{S}, K)$ of* MaxCover *and each integer X, $0 \leq X \leq K$, the algorithm* ExpoAndGreedy *from Figure 6.5 computes an $\left(1 - \frac{X}{K}e^{-1}\right)$-approximate solution for I in time $\binom{m}{K-X} + \mathrm{poly}(K, n, m)$.*

Proof. Let $I = (N, \mathcal{S}, K)$ be our input instance and let $\mathcal{C}^*$, $\mathcal{C}^* \subseteq \mathcal{S}$, denote some optimal solution. Let $\mathcal{C}^*_X$ denote a subset of $(K - X)$-elements from $\mathcal{C}^*$ that together cover the greatest number of the elements. Thus the sets from $\mathcal{C}_X^*$ cover at least a fraction $\frac{K-X}{K}$ of all the elements covered by the optimal solution. Consider the problem of covering the elements uncovered by $\mathcal{C}_X^*$ with X sets from $(\mathcal{S} \setminus \mathcal{C}_X^*)$. We know that $(\mathcal{C}^* \setminus \mathcal{C}_X^*$ is an optimal solution for this problem. On the other hand, we also know that the greedy algorithm achieves approximation ratio $(1 - \frac{1}{e})$ for the problem. Thus, the approximation ratio for the original problem is:

$$\left(\frac{K-X}{K} + \frac{X}{K}\left(1 - \frac{1}{e}\right)\right) = \left(1 - \frac{X}{K}e^{-1}\right).$$

It is immediate to establish the running time of the algorithm and so the proof is complete. □

If we wish to solve MaxVertexCover rather than MaxCover, then in the algorithm ExpoAndGreedy we should replace the greedy approximation algorithm with that of Ageev and Sviridenko [5].

Corollary 6.16. *There exists an* $\left(1 - \frac{X}{4K}\right)$*-approximation algorithm for* MaxVertexCover *problem running in time* $\binom{m}{K-X} + \mathrm{poly}(K, n, m)$

It is quite evident that as long as algorithm $\mathcal{A}$ used within algorithm GreedyAndExpo in Figure 6.4 is the simple brute-force algorithm that tries all possible solutions, then algorithm ExpoAndGreedy is superior; in the same time it achieves a better approximation ratio. It turns out that, for the case of MaxVertexCover, the algorithm ExpoAndGreedy (in the variant from Corollary 6.16) is also better than the algorithm of Croce and Paschos [70].[6]

The idea behind the algorithm of Croce and Paschos [70] for MaxVertexCover is similar to that behind our algorithm ExpoAndGreedy. Specifically, given two algorithms for MaxVertexCover, approximation algorithm $\mathcal{A}_a$ and exact algorithm $\mathcal{A}_e$, for a given value X it first uses $\mathcal{A}_e$ to find am optimal solution that uses $K - X$ vertices (out of the K vertices that we are allowed to use in the full solution), then it removes these $K - X$ vertices and solves the remaining part of the problem using $\mathcal{A}_e$. Assuming that β_a is the approximation ratio of the algorithm $\mathcal{A}_a$, this approach results in the approximation ratio equal to $\left(\frac{X}{K} + \beta_a\left(1 - \frac{X}{K}\right)^2\right)$.

Below we compare the algorithm ExpoAndGreedy (version from Corollary 6.16) with the algorithm of Croce and Paschos [70]. As the components $\mathcal{A}_a$ and $\mathcal{A}_e$ we use, respectively, the $\frac{3}{4}$-approximation algorithm of Ageev and Sviridenko [5] and the

[6]The algorithm GreedyAndExpo cannot be directly compared to the algorithm of Croce and Paschos [70] for the following reason. The algorithm GreedyAndExpo uses specifically a greedy algorithm which is the best known approximation algorithm for MaxCover, but which is suboptimal for MaxVertexCover. In contrast, the algorithm of Croce and Paschos [70] can use, e.g., the $\frac{3}{4}$-approximation algorithm of Ageev and Sviridenko [5]. One could, of course, try to use the algorithm of Ageev and Sviridenko in the algorithm GreedyAndExpo, but our analysis does not work for this case.

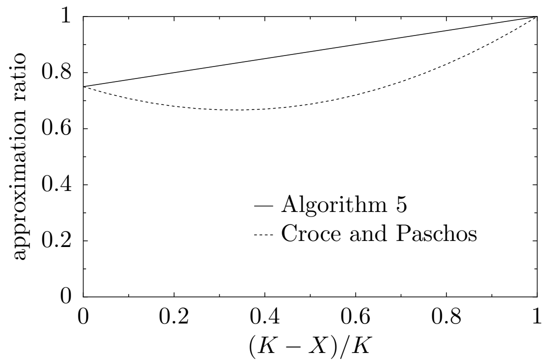

Figure 6.6: The comparison of the approximation ratios of the algorithm ExpoAndGreedy from Figure 6.5 and the algorithm of Croce and Paschos [70] for MaxVertexCover.

brute-force algorithm that tries all possible solutions. The best known exact algorithm for MaxVertexCover is due to Cai [39] and has the complexity $O(m^{0.792K})$, but this algorithm uses exponential amount of space. Since exponential space complexity might be much less practical than exponential time complexity, we decided to use the brute-force approach (to the best of our knowledge, there is no better exact algorithm running in polynomial space). We present our comparison in Figure 6.6. The x-axis represents the parameter $\frac{K-X}{K}$, measuring the fraction of the solution obtained using the exact algorithm (for 0 we use the approximation algorithm alone and for 1 we use the exact algorithm alone). On the y-axis we give approximation ratio of each algorithm. In other words, for each point on the x-axis we set the X parameters of the algorithms to be equal, so that their running times are the same, and we compare their approximation guarantees.

We conclude that, as long as we use the brute-force algorithm as the exact one, the algorithm ExpoAndGreedy gives considerably better approximation guarantees than that of Croce and Paschos. Figure 6.6 also exposes one potential weakness of the algorithm of Croce and Paschos. Apparently, for some cases increasing the complexity of the algorithm results in the decrease of its approximation guarantee.

It is quite interesting to understand the reasons behind the differing performance of the algorithm ExpoAndGreedy and that of Croce and Paschos. In some sense, the algorithms are very similar. If we use the brute-force algorithm as the exact one in the algorithm of Croce and Paschos, then the main difference is that our algorithm runs the approximation algorithm for each possible solution tried by the brute-force algorithm, and Croce and Paschos's algorithm only runs the approximation algorithm once, for the best partial solution. In effect, our algorithm can exploit situations where it is better when the exact algorithm does not find an optimal solution for the subproblem, but rather leaves ground for the approximation algorithm to do well. Naturally, such strategy is only possible if we have additional knowledge of the structure of the exact algorithm (here, the brute-force algorithm). The result of

Croce and Paschos pays the price for being more general and being able to use any combination of the approximation algorithm and the exact algorithm.

6.6 Conclusions

Motivated by the study of recommendation systems, item selection mechanisms, and winner-determination under Chamberlin–Courant's voting rule (with approval misrepresentation), we have considered the MaxCover problem with bounded frequencies and its minimization variant, the MinNonCovered problem, from the point of view of approximability by FPT algorithms. We have shown that for upper-bounded frequencies there is an FPT approximation scheme for MaxCover and a randomized FPT approximation scheme for MinNonCovered. For lower-bounded frequencies we have shown that the standard greedy algorithm for MaxCover may achieve a better approximation ratio than in the unrestricted case. Finally, we have shown that in the unrestricted case there are good exponential-time approximation algorithms (though, not FPT ones) that combine exact and greedy algorithms and smoothly exchange the quality of the approximation for the running time. Some of our results regarding MaxCover with bounded frequencies improve previously known results for MaxVertexCover. In particular, our Algorithm BoundAndExplore improves upon the approximation scheme given by Marx, and our algorithm ExpoAndGreedy improves upon the result of Croce and Paschos [70] (provided we use brute-force algorithm as the underlying exact algorithm in the scheme proposed by Croce and Paschos; this is reasonable if we are interested in algorithms that use only polynomial amount of space).

The results presented in this chapter complement the results from Chapter 5. Our results, inter alia, show that it is possible to achieve a reasonably good proportional representation even for the approval utilities of the agents. Such results are significant because, in practice, it is much easier for an agent to point out a group of approved alternatives instead of providing their precise ranking. On the other hand, some of the algorithms provided in this chapter require super-polynomial running time, and so they are much slower then those from Chapter 5 (yet, still much faster than pure brute-force algorithms).

There are several interesting directions for future research. For example, is it possible to obtain FPT approximation schemes for MaxCover with lower-bounded element frequencies? Further, what is the exact complexity of MaxCover (with or without lower-bounded frequencies)? We have quickly observed its W[2]-hardness, but does it belong to W[2]? (It is quite easy, however, to show that it belongs to W[P].) We are also interested in the exact complexity of MaxCover with lower-bounded frequencies for the case where we require the ratio of frequency lower-bound and the number of sets to be at least some given value α? We have given a PTAS for this variant of the problem (Theorem 6.11) and have shown its membership in β^2, but we did not attempt to prove its completeness for any particular complexity class.

Part II

Scheduling models

In the following three chapters (Chapters 7, 8, and 9) we consider resource allocation problems in more specific, scheduling-based models. In contrast to the previous part, here we consider resource allocation problems in settings in which we are able to construct more involved models including additional elements, such as jobs, servers, the network, and the functions characterizing these elements.

In the three following chapters we consider three models describing three different computer systems. Chapter 7 is an intermediate step in our analysis. On one hand we start from a model describing general agents—thus, such model is also applicable to distributed non-computer systems. On the other hand, we show that this general model can be specialized with the precise scheduling model. Chapters 8 and 9 concern specific scheduling models in computer systems.

The analysis of this part of the dissertation shows a diversified view on the agents behavior and interaction. Chapter 7 considers strategic agents and studies their competition. Chapter 8 examines the problem of finding a fair schedule from a cooperative game theory standpoint. Chapter 9 describes an organizationally centralized distributed system owned by a single entity.

In our models we define solution concepts describing the stability of the considered systems, and analyze the existence of these solution concepts. We also analyze the complexity of computing these stable states in the considered systems. Additionally, we analyze various optimization problems and study their complexity. The high level contribution presented in this part of the dissertation is the following: (i) We provide a game-theoretic analysis of the stability of the three example distributed systems. (ii) We establish the complexity of the computational problems in the considered systems. (iii) For the computationally hard problems we present alternative solutions. These solutions include: approximation algorithms, FPT algorithms, and heuristic algorithms. Below we give a brief outline of this part of the dissertation.

In Chapter 7 we propose a theoretical model describing the strategic interaction between teams of agents competing for an employment in a project. The agents are heterogeneous and might have different skills. In the most general part of this chapter we assume that there exists an oracle that can answer whether a given subset of agents is capable of completing a project before the given deadline. The agents have their requirements for minimal salaries, but they are also strategic and will always prefer to work for a higher wage. The project issuer runs an auction to find a team that will be awarded the project. The project issuer is strategic too—she wants to find a team capable of completing the project at the smallest possible cost. Our problem models various situations in which agents can form consortia to win participation in a large project, including crowd-sourcing (e.g., amazon mechanical turk), on-line service sites, but also public tenders for large civil engineering projects (in which agents represent contractor companies specialized in certain branches of construction or engineering).

We propose several solution concepts describing the winning coalitions and we study computational complexity of finding them. We show algorithms finding (weakly) winning coalitions in a polynomial number of calls to the oracle. As a consequence, we show that the complexity of finding (weakly) winning coalitions depends solely on the complexity of an oracle. In other words, the problem is solvable in polynomial time if and only if the oracle can check in polynomial time whether there exists a team of agents that is able to complete the project before a deadline.

Next, we propose a specific scheduling-based model for an oracle, in which the project consists of multiple jobs, and the individual agents have certain skills (expressed as the speeds of completing different jobs). We show that under this specific model the problem is NP-hard and hard to approximate, even if the values of the various natural parameters are low. On the other hand, we show that the problem can be solved using standard integer programming solvers.

In Chapter 8 we consider a distributed system consisting of multiple organizations sharing their infrastructures. Each organization participating in the system contributes to the global pool a certain number of processors. As the reward, every organization is eligible to run its jobs on all the processors from the common pool. We study a specific scheduling model: on-line, non-clairvoyant scheduling of sequential jobs. The started jobs cannot be stopped, canceled, preempted, or moved to other processors. In this model we look for scheduling algorithms that are fair for the participating organizations.

We show how to model the fair scheduling problem as a cooperative game and how to use the Shapley value to determine the ideal fair schedule. In contrast to the previous works in this area, our approach does not use the concept of money. This makes our considerations more practical, as (i) money discourages people from cooperating [296], and (ii) in many contexts, it is not clear how to valuate the completion of a job or the usage of a resource (especially when workload changes dynamically). We present exact, ideally-fair scheduling algorithm. Although algorithm has exponential running time, it is fixed parameter tractable, when parameterized by the number of organizations. Thus, if the number of participating organizations is relatively low, the problem of finding a ideally fair schedule is computationally tractable. In general, we show that the problem is NP-hard and hard to approximate. However, we show good approximation algorithms for some restricted cases, and a heuristic algorithm that gives good results when evaluated on real data traces.

In Chapter 9 we address resource allocation problems in the geographically distributed systems. We consider load balancing in the systems that serve large numbers of users from all over the world (the examples of such systems include content delivery networks, web search systems, and P2P storage systems). In contrast to the common approach to load balancing, in our work we assume that the communication latencies between the servers in the system are non-negligible. The perceived processing time of a user request is thus composed of the time needed to route the request to the server and the true processing time. Once a request reaches a target server, the processing time depends on the total load of that server; this dependency is described by a load function.

We present two algorithms for general load functions: the centralized one and the distributed one; and one algorithm for the specific load functions describing the system in which requests arrive in batches. We show load balancing algorithms that work for a broad class of load functions and we analyze their complexity. The complexity of our algorithms for general load functions depends on the number of machines, the total load in the system, and the derivatives of the load functions. These are any-time algorithms, i.e. we can stop them at any moment and extract a (possibly suboptimal) solution. Our algorithms also have other interesting properties, e.g., they are robust to failures, and the machines do not require any additional synchronization.

Chapter 7

People are Processors: Coalitional Auctions for Complex Projects

To successfully complete a complex project, be it a construction of an airport or of a backbone IT system or a crowd-sourced project, agents (companies or individuals) must form a team (a coalition) having required competences and resources. A team can be formed either by the project issuer based on individual agents' offers (centralized formation) or by the agents themselves (decentralized formation) bidding for a project as a consortium—in this case many feasible teams compete for the employment contract. In these models, we investigate rational strategies of the agents (what salary should they request? with whom should they team up?) under different organizations of the market. We propose various concepts allowing to characterize the stability of the winning teams. We show that there may be no (rigorously) strongly winning coalition, but the weakly winning and the auction-winning coalitions are guaranteed to exist. In a general setting, with an oracle that can point out a feasible coalition or claim there is no such, we show how to find winning coalitions with a polynomial number of calls to the oracle. We also determine the complexity of the problem in a special case in which a project is a set of independent tasks. Each task must be processed by a single agent, but processing speeds differ between agents and tasks.

7.1 Introduction

Modern projects are getting complex: involved, intricate, and consisting of many varied yet interrelated parts [16,304]. The successful completion of such complex projects requires coordinated operation of a number of highly-specialized people and companies, often organized as teams of subcontractors [12]. For instance, in the construction industry, to build an apartment building (a rather standard endeavor), typically, 30 to 40 individual sub-contractors are involved in 100 to 150 separate activities [297].

Indeed, assigning sub-tasks of a complex project to multiple subcontractors is common [50]. According to Edwards [92], in the United Kingdom, the proportions of construction employees employed by sub-contractors in years 1983–1998 has grown by 20%. In the United Kingdom, between 2008 and 2011, the number of people who work as freelancers has increased by 12% [213]; in Australia in 2012, 17.2% of the workforce were self-employed (8.5% as independent contractors) [140]. These are only a few examples of a growing tendency to develop various projects by employing many specialized sub-contractors instead of a single company. We envision that with the proliferation (and further development) of crowd-sourcing and collaboration platforms, this tendency will further strengthen.

Nevertheless, it is not clear how to organize the market both for the issuer of the project (in this chapter referred to as the client) and the subcontractors (in this chapter referred to as the agents). Interaction between the agents applying for the employment in a project and the client is described in the *hiring a team* problem [14,52,53,108,142,155,286]. In the hiring a team problem, the agents have private costs of participating in the project. The agents may have different sets of skills, thus only certain teams are able to complete the project on time. The client organizes an auction in which individual agents place their bids, i.e., their required salaries. After collecting the bids, the client selects the cheapest feasible team, i.e., the team of agents that is able to complete the project on time, with the lowest total bid.

We generalize the original approach from the hiring a team problem by exploring two organizations of the market. The original approach corresponds to the *centralized setting*, where the agents communicate only with the client by issuing their bids. Our main contribution lies in considering a fully decentralized setting where *the whole teams bid for a project*. To the best of our knowledge this formulation of the problem is novel and leads to a new class of games. This formulation has a natural real-world representation: a client does not want to coordinate a project and to deal with individual subcontractors, but instead expects that the subcontractors coordinate between themselves and propose a bid for completing the whole project as a consortium. Additionally, we generalize the original approach by considering two types of agents' compensation. In the *project salary model* (corresponding to the original approach) the agents are payed for their whole participation in the project (irrespectively of the contributed effort). We contribute a new payment model, the *hourly salary model*, in which the agents are payed for the time spent working on the project.

Although our main contribution lies in considering the decentralized setting, we also complement the literature on the centralized one (the hire a team problem) [14, 52,53,108,142,155,286]. To the best of our knowledge, the current literature focuses on designing truthful mechanisms that encourage agents to ask for the salaries corresponding to their actual costs of participation. The known truthful mechanisms, however, result in a significant overpayment by the client [14,52,53,108,155,286]. In contrast, we consider a market in which the agents are payed their asking salaries and nothing more. This mechanism, which corresponds to the first-price auctions, is manipulable, but once the agents set their asking salaries no overpayment is required. Since the first-price auctions are, in general, resistant to collusions [170], we believe this auction system is relevant in our case.

Finally, in contrast to, but also complementing, the previous works, we focus on the computational aspects of finding winning teams rather than on the designing truthful mechanism.

Throughout this chapter we assume that we are given an oracle that, for a given team of agents, can determine whether this team is feasible, i.e., whether it can successfully complete the project. Further, given a budget for the project and the costs of the agents, the oracle can find a feasible team and the cheapest feasible team. Our approach generalizes two models known in the literature: commodity auctions [202] and path auctions [223]. In a commodity auction, there is a set of items $I = \{i_1, i_2, \ldots, i_q\}$ and agents owning certain subsets of I. A coalition is feasible if the agents have together all the items from I. A commodity auction can be mapped to our problem by considering that I is a set of independent activities; an agent owning a subset corresponds to an agent that is able to complete these activities. In a path auction, there is a graph G with two distinguished vertices: a source s and a target t. The agents correspond to the vertices in the graph, some vertices are connected with the edges. The coalition is feasible if the participating agents form a path from s to t. For the general case, we also point out that the oracle can use algorithms for coalition formation [253] to solve the subproblem of finding (cheapest/best) feasible coalitions. Specifically, coalition formation protocols in the context of the complex projects (without game-theoretic consideration) were considered by Kraus et al. [168]. We also point out that some works consider that agents' skills, in our case known by the oracle, are their private information, and can be revealed only at additional cost [31].

Since we consider coalitions of agents with sufficient skills to complete the project, our model resembles cooperative skill games [17] and coalitional resource games [306]. These games, however, consider the stability of the *grand* coalition and interaction between its members. Similarly, the cooperative game theory view on coalition formation in the context of complex projects was considered by Kraus et al. [169]. Our approach, on the other hand, is to expose the competition between multiple teams/coalitions. Thus, we do not apply the typical cooperative game theory concepts [231] and, instead, model the cooperation and the competition of the agents as a non-cooperative game.

The contributions of this chapter are as follows: (i) First we identify and formalize a new class of coalition games, which are the extensions of the games from the hiring a team problem. In the centralized setting (Section 7.3), where the agents communicate only with the client, (ii) we prove that a Strong Nash Equilibrium (SNE) always exists unless there is no feasible coalition. We show how to find an SNE, and how the client

Table 7.1: The summary of the notation used in the further part of this chapter.

Symbol	Meaning
v	maximal price for the project
N	set of agents
$\phi_i^{\min}$	minimal salary of the i-th agent
$\mathcal{C} = \langle N_{\mathcal{C}}, \phi_{\mathcal{C}}, c_{\mathcal{C}} \rangle$	coalition participating in the auction
$N_{\mathcal{C}}$	set of agents forming the coalition $\mathcal{C}$
$\phi_{\mathcal{C}}$	function assigning salaries to the members of the coalition $\mathcal{C}$
$\phi_i = \phi_{\mathcal{C}}(i)$	salary of the i-th agent (if the coalitions known from the context)
$\phi_{\mathcal{C}}^{tot}(i)$	total amount of money agent i gets in coalition $\mathcal{C}$
$c_{\mathcal{C}} = \sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}^{tot}(i)$	total cost (the bid) of the coalition $\mathcal{C}$
$\sigma_{\mathcal{C}}$	schedule of the coalition $\mathcal{C}$ (assigning to each member of the coalition $\mathcal{C}$ the amount of time this agent needs to spend on the project)
t_i	time that the i-th agent spends working on the project

can select the cheapest coalition with only a polynomial number of calls to the oracle. In the decentralized setting (Section 7.4), (iii) we propose two concepts of winning coalitions. We prove that a *strongly winning* coalition may not exist, but a *weakly winning* coalition is guaranteed to exist (provided there exists a feasible coalition). We show how to find weakly/strongly winning coalitions. Still, for the decentralized setting, (iv) we propose two mechanisms that the client can use to find the winning coalition (Section 7.5). We introduce the concept of an auction-winning coalition and show how to find one. By specifying an oracle, our results can be applied to two different problems known in the literature: the commodity auctions and the path auctions. In Section 7.6 we propose another way in which the general oracle can be replaced with a concrete scheduling model (it is a generalization of the model from the commodity auctions setting) and (v) determine the exact complexity of this concrete problem.

7.2 The Auction Model for Complex Projets

In this section we introduce the notation and describe the model further used in this chapter. The notation is also summarized in Table 7.1.

We consider a model in which a client (an issuer) submits a single complex project to be executed. The client has a certain valuation v of the project, that is the maximal price that she is able to pay for completing the project.

There is a set $N = \{1, 2, \ldots n\}$ of n agents. For each agent i, we define $\phi_i^{\min} > 0$ to be the agent's *minimal salary* for which i is willing to work. This minimal salary may correspond to the agent's personal cost of participating in the project. The agent prefers to work for $\phi_i^{\min}$ than not to work (and then to work for higher salary). The value $\phi_i^{\min}$ is private to the agent—neither the issuer nor the other agents know $\phi_i^{\min}$. However, in order to analyze the behavior of the agents in our system, we assume that agents have some beliefs about the minimal salaries of other agents.

A subset of the agents' population N forms a coalition to be awarded the project[1]; the chapter's core contribution is on how this process should be organized.

A coalition $\mathcal{C}$ is a triple $\langle N_{\mathcal{C}}, \phi_{\mathcal{C}}, c_{\mathcal{C}} \rangle$ consisting of the set of participating agents $N_{\mathcal{C}} \subseteq N$, a salary function $\phi_{\mathcal{C}} : N_{\mathcal{C}} \to \mathbb{N}$ assigning salaries to member agents, and the total cost of the coalition $c_{\mathcal{C}} \in \mathbb{N}$—the total amount of money earned by the participants of $\mathcal{C}$. Salaries are discrete (not only money is discrete, but also it is common in real-world auctions to specify a minimal difference between two successive bids). However, to get some computational results, in some, clearly marked, places we assume that the salaries can be rational numbers.

The same coalition may organize the work of its members on the project in various ways. Each such a way may require different amount of effort from different participants. To capture this property, we introduce a notion of a *schedule*, $\sigma_{\mathcal{C}} : N_{\mathcal{C}} \to \mathbb{N}$, that assigns to each member of a coalition the amount of time this agent needs to spend on the project. Of course, there may exist many schedules for a single coalition. We will expand the discussion on the notion of schedule in Section 7.6.

We consider two models of agents' compensation. Let $\phi_{\mathcal{C}}^{tot}(i)$ denote the total amount of money agent i gets in coalition $\mathcal{C}$ (naturally, $c_{\mathcal{C}} = \sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}^{tot}(i)$). In the *project salary* model $\phi_{\mathcal{C}}^{tot}(i)$ is equal to the salary of the agent $\phi_{\mathcal{C}}(i)$ (and thus does not depend on the amount of work assigned to that agent). In the *hourly salary* model $\phi_{\mathcal{C}}^{tot}(i)$ is equal to the product of the salary $\phi_{\mathcal{C}}(i)$ and the time t_i that i spends on processing her part of the project (t_i is known from the schedule).

In the project salary model the agents are interested in earning as much money as possible. The hourly salary model represents a different environment in which agents perhaps work on many projects simultaneously; thus the agents are interested in having maximal salary per time unit (thus, e.g., an agent prefers to work $t_i = 1$ time unit with a salary $\phi_i = 3$ to working $t_i = 2$ time units with a salary $\phi_i = 2$).

Different schedules might result in different completion times of the project. If the schedule results in a completion time that is satisfactory for the project issuer, we say that the schedule is *feasible*. Of course for some coalitions there might not exist a feasible schedule (e.g., if the coalition members are lacking certain critical skills). We assume that there is an oracle that can answer whether a given schedule is feasible or not. This very general setting can be specified by providing concrete models for the oracle. For instance, in Section 7.6 we show that by appropriately specifying the

[1]For the sake of the clarity of the presentation we refer to the teams of agents as to the coalitions, even if it sometimes abuses the definition of the coalition from the cooperative game theory.

oracle, our results can be applied to two different settings known in the literature: to the commodity auctions setting and the path auctions setting. In Section 7.6 we also show how to replace the general oracle with a new concrete scheduling model.

A coalition $\mathcal{C}$ is *feasible* if there exist a feasible schedule such that: (i) the project budget is not exceeded ($c_{\mathcal{C}} \leq v$), and (ii) the cost $c_{\mathcal{C}}$ of the coalition $\mathcal{C}$ is consistent with the salaries $\phi_{\mathcal{C}}$. Specifically, in the project salary model $c_{\mathcal{C}} = \sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}(i)$. In the hourly salary model $c_{\mathcal{C}} = \sum_{i \in N_{\mathcal{C}}} t_i \phi_{\mathcal{C}}(i)$, where t_i is the number of time units the agent i needs, according to the feasible schedule, to spend working on the project. Moreover, the asking salaries are no-lower than the minimal salaries, $\phi_{\mathcal{C}}(i) \geq \phi_i^{\min}$.

A coalition $\mathcal{C}$ is *cheaper* than $\mathcal{C}'$ if it has a strictly lower cost $c_{\mathcal{C}} < c_{\mathcal{C}'}$ or if it has the same cost, but it is preferred by a deterministic tie-breaking rule $\prec$, $N_{\mathcal{C}} \prec N_{\mathcal{C}'}$ (for the sake of concreteness we assume that $\prec$ is the lexicographic order in which a coalition is represented by a concatenation of the sorted list of the names of its members).

Throughout this chapter we use the FIND FEASIBLE COALITION (FFC) and FIND CHEAPEST FEASIBLE COALITION (FCFC) problems. We reduce other problems to FFC and FCFC (we will also show that FCFC can be polynomially reduced to FFC).

Problem 7.1 (FFC: Find Feasible Coalition)**.** *An instance of FFC consists of a project (with a budget v) and the set N of the agents with (known) minimal required salaries $\phi_i^{\min}$. The question is to find some feasible coalition or to claim there is no such.*

Problem 7.2 (FCFC: Find Cheapest Feasible Coalition)**.** *An instance is the same as in the* FFC *problem. The question is to find some cheapest feasible coalition or to claim there is no such.*

We use the general model as defined above in Sections 7.3, 7.4 and 7.5. Since the complexity of the above problems clearly depends on the underlying model for the oracle, we assume that our oracle can solve the FFC problem. This allows us to study a very general setting of the problem, abstracting from concrete notions of a schedule or of a division of labor in a coalition.

To solve FFC, the oracle must know the underlying model, in particular the minimal salaries of agents and the maximal price of the project v; we may also assume that the oracle is local to an agent. i.e., the oracle knows the minimal salary of this agent and agent's beliefs about minimal salaries of others. Then, to get the exact computational results, we need to define in a compact form, e.g., which coalitions are able to complete the project. In Section 7.6 we consider a specific model of FFC in which the project is a set of independent, indivisible tasks that need to be completed before a certain deadline and the agents have certain skills, i.e., speeds with which they process the tasks. We also discuss other specific models for the oracle that lead to the concrete models considered in the literature.

We consider two models of forming coalitions. First, in Section 7.3, we consider the *centralized formation*. This model is similar to the model from the *hiring a team* problem [14,52,53,108,142,155,286]. Agents submit their bids i.e., (asking) salaries ϕ_i, directly to the client (project issuer). The client chooses the members of the coalition that is awarded the project. Naturally, the client chooses the members so that the cheapest feasible coalition is formed. The members of the winning coalition are paid according to their asking salaries ϕ_i. This is different from the literature on the set system auctions that considers different payment mechanisms that ensure the truthfulness of the agents [14,52,53,108,155,286].

Second, in Sections 7.4 and 7.5, we consider the *decentralized formation* of the coalition. Agents communicate and are able to form coalitions by binding agreements. A coalition sends a bid—the total cost $c_{\mathcal{C}}$—to the client; the bid represents the compensation the coalition expects to get for completing the whole project. The cheapest coalition $\mathcal{C}^*$ wins the project and is paid $c_{\mathcal{C}^*}$; then $c_{\mathcal{C}^*}$ is allotted to the members of the winning coalition according to the salary function $\phi_{\mathcal{C}^*}$.

Our problem models various situations in which agents can form consortia to win a large project, including: croudsourcing (e.g., amazon mechanical turk), online service sites, but also public tenders for large civil engineering projects (in which agents represent contractor companies specialized in a certain branch of construction or engineering).

7.3 Centralized Formation of Coalitions

In the centralized model we assume that the agents submit their asking salaries ϕ_i directly to the client (the issuer of the project). The client, having the asking salaries of the agents, wants to form the cheapest feasible coalition (that is able to complete the project before the deadline). In this section, we first show that this problem reduces to FFC, the problem of finding a feasible coalition. Then, we analyze the optimal bidding strategies of agents.

Proposition 7.1. *The problem* FCFC *can be solved in time $O((\log v + n) \mathit{ffc})$, where ffc is the complexity of the problem* FFC*.*

Proof. First, we solve FFC with binary search over v to find the lowest bid v^* for which there still exists a feasible coalition.

Next, we need to find the coalition bidding v^* that is preferred by the tie-breaking rule. We recall that $\mathcal{C} \prec \mathcal{C}'$ if $\mathcal{C}$ precedes $\mathcal{C}'$ in the lexicographic order. We consider the agents in the increasing order of their names. For each agent i we decrease her salary by 1 ($\phi_i^{\min} := \phi_i^{\min} - 1$) and solve FFC for $v = v^* - 1$. If there is one, this means that in the initial setting there exists a feasible coalition offering bid v^* and having agent i as a member. We store i as a member of the winning coalition. With the modified salary of i and an updated budget of $v^* = v^* - 1$ we consider the next agent. Otherwise we reset the agent salary $\phi_i^{\min}$ and the budget v^* to their previous values and consider the next agent. $\square$

Proposition 7.2. *Having the asking salaries of the agents, the problem of finding the winning coalition can be solved in time $O(fcfc)$, where fcfc is the complexity of the problem* FCFC.

Proof. Solving the problem requires solving FCFC with the minimal salaries of the agents set to their asking salaries ($\phi_i^{\min} = \phi_i$). $\square$

The agents may behave strategically and manipulate their asking salaries to maximize their payoffs. We model this problem as a strategic game. An action of agent i is her asking salary $\phi_i \geq \phi_i^{\min}$. The payoff of i is ϕ_i if and only if i is a member of the cheapest feasible coalition; otherwise the payoff of i is 0.

Interestingly, in such setting, in the project salary model, there exist sets of vectors of actions which are stable against collaborative strategies of the agents. We recall that a vector of the agents' actions is a Strong Nash Equilibrium (SNE, [15]) if no subset of the agents can change its actions so that all the deviating agents obtain strictly better payoffs.

For each subset of the agents $N' \subseteq N$, by $\mathcal{C}^*(N')$ we denote the cheapest feasible coalition using only the agents from N' (the coalition $\mathcal{C}^*(N')$ does not exist if there is no feasible coalition consisting of the agents from N').

Theorem 7.3. *In the project salary model, if there exists a feasible coalition then there exists a Strong Nash Equilibrium. In every SNE, the set of the agents that get positive payoffs is the set of agents forming the cheapest feasible coalition, $N_{\mathcal{C}^*(N)}$.*

Proof. Let $N^* = N_{\mathcal{C}^*(N)}$ be the set of the agents participating in the cheapest feasible coalition. We say that the action ϕ_i of the agent i is *minimal* if and only if $\phi_i = \phi_i^{\min}$. We show how to construct the asking salaries ϕ_i^* of the agents from N^* that, together with the minimal actions of the agents outside N^*, form the Strong Nash Equilibrium. A sketch of proof is as follows. We show the set of linear inequalities for the variables $\phi_i, i \in N^*$. Let us denote the maximal values of ϕ_i which satisfy the inequalities as ϕ_i^* (maximal in the sense that if we increase any value ϕ_i^*, then the new values will not satisfy all the inequalities any more). We show that the actions ϕ_i^* of the agents from N^*, together with the minimal actions of the agents outside of N^*, form an SNE and that the set of the solutions ϕ_i^* that satisfy all the inequalities is nonempty.

The first inequality states that the values ϕ_i must lead to a feasible solution:

$$\sum_{i \in N^*} \phi_i \leq v. \tag{7.1}$$

Next, as $\mathcal{C}^*$ is the cheapest feasible coalition, for each feasible coalition $\mathcal{C}'$ ($N^* \neq N_{\mathcal{C}'}$) such that $N^* \prec N_{\mathcal{C}'}$, $\mathcal{C}^*$ must have (weakly) lower cost:

$$\sum_{i \in N^* \setminus N_{\mathcal{C}'}} \phi_i \leq \sum_{i \in N_{\mathcal{C}'} \setminus N^*} \phi_i^{\min}. \tag{7.2}$$

For $\mathcal{C}'$ preferred over $\mathcal{C}^*$ ($N^* \neq N_{\mathcal{C}'}$ and $N_{\mathcal{C}'} \prec N^*$), $\mathcal{C}^*$ must have strongly lower cost:

$$\sum_{i \in N^* \setminus N_{\mathcal{C}'}} \phi_i < \sum_{i \in N_{\mathcal{C}'} \setminus N^*} \phi_i^{\min}. \tag{7.3}$$

First, if the values ϕ_i^* satisfy the above inequalities and the agents outside of N^* play their minimal actions, then the agents from N^* will get positive payoffs. If they did not get the positive payoffs, it would mean that there exists a feasible cheaper coalition $\mathcal{C}'$. However, the inequalities ensure that the agents from $N^* \setminus N_{\mathcal{C}'}$ induce the lower total cost than the total cost of the agents from $N_{\mathcal{C}'} \setminus N^*$; this ensures that agents N^* with actions ϕ_i^* form a cheaper coalition than $\mathcal{C}'$.

Next, we show that no set of agents $N_{\mathcal{C}'}$ can make a collaborative action $\overline{\phi}$ after which the payoff of all agents from $N_{\mathcal{C}'}$ will be greater than previously. For the sake of contradiction let us assume that there exists such a set of agents $N_{\mathcal{C}'}$ and such an action $\overline{\phi}$. First we consider the case when the payoff of some agent $i \notin N^*$ would change. This means that after $\overline{\phi}$ there would be a new cheapest feasible coalition $\mathcal{C}'$, where $i \in N_{\mathcal{C}'}$. However, we know that the total cost of the agents from $N^* \setminus N_{\mathcal{C}'}$ is lower than the total cost of the agents from $N_{\mathcal{C}'} \setminus N^*$. This means that $\mathcal{C}'$ cannot be cheaper than the coalition consisting of the agents from N^*. Finally, consider the case when only the payoffs of the agents from N^* change (and thus $N_{\mathcal{C}'} \subseteq N^*$). However, if the strict subset of N^* could form a feasible coalition, then $\mathcal{C}^*(N)$ would not be the cheapest. Thus, $N_{\mathcal{C}'} = N^*$. This means that every agent from N^* must have played a higher action (and others must have not changed their actions). Since ϕ_i^* were maximal, this means that after the action $\overline{\phi}$ some inequality, for some feasible coalition $\mathcal{C}''$, would not hold any more. Thus, we infer that $\mathcal{C}''$ is cheaper than $\mathcal{C}'$.

To check that there always exists a solution, we see that the definition of N^* ensures that the values $\phi_i^* = \phi_i^{\min}$ satisfy all inequalities.

Finally, by contradiction we prove the N^* is formed by the same agents as forming the cheapest coalition. Assume that the set of the agents that get positive payoffs in some SNE is $N' \neq N^*$. However, if the agents from $(N^* \setminus N')$ play their minimal actions, then the coalition consisting of the agents from N^* would be cheaper than the coalition consisting of the agents from N'. Thus, the agents from $(N^* \setminus N')$ can deviate, getting better payoffs. This completes the proof. □

Interestingly, there is no analogous result for hourly salary model.

Proposition 7.4. *In the hourly salary model there may not exist a Strong Nash Equilibrium even though there exists a feasible coalition.*

Proof. Let us consider the following instance. The budget is $v = 49$. There are 3 agents: a, b, and c; their minimal hourly salaries are $\phi_a^{\min} = \phi_b^{\min} = \phi_c^{\min} = 1$. All two-agent coalitions can complete the project: if a and b cooperate they can complete the project spending on it $t_a = 10$ and $t_b = 10$ time units, respectively; if a and c

cooperate they must spend $t_a = 22$ and $t_c = 2$ time units; if b and c cooperate they must spend $t_b = 2$ and $t_c = 38$ time units.

For the sake of contradiction let us assume that there exists a Strong Nash Equilibrium. First, consider the case when the agents a and b get positive payoffs in SNE. By the budget constraint, $\phi_b \leq 3$. If $\phi_b = 3$, then $\phi_a = 1$. The total cost of $\{a, b\}$ is 40. However, c, by playing $\phi_c = 1$ can form a cheaper coalition $\{a, c\}$ with the total cost 24. If $\phi_b \leq 2$ and $\phi_a = 1$, then a has an incentive to play higher. If $\phi_b \leq 2$ and $\phi_a \geq 2$, then b and c are better off by playing a collaborative action with $\phi_b = 3$ and $\phi_c = 1$ — after such an action a coalition $\{b, c\}$ is cheaper ($c_{b,c} = 44$) than $\{a, b\}$ ($c_{a,b} \geq 50$) and $\{a, c\}$ ($c_{a,c} \geq 46$). Thus, a and b cannot both have positive payoffs in SNE.

Second, assume that the agents a and c get positive payoffs in SNE. The total cost of $\{a, c\}$ is $22\phi_a + 2\phi_c$. In such case, if b plays ϕ_a then the new coalition $\{a, b\}$ with total cost $10\phi_a + 10\phi_a$ forms a new cheapest coalition.

Finally consider the case when b and c get positive payoffs in SNE. This means that $\phi_c = 1$. But a, by playing 1 can form a coalition $\{a, c\}$ with the total cost 24. This completes the proof. □

Proposition 7.5. *Checking whether a given vector of the asking salaries $\langle \phi_i \rangle, i \in N$ is a Strong Nash Equilibrium can be solved in time $O(fcfc)$, where fcfc is the complexity of the problem* FCFC.

Proof. First, we find a winning coalition $\mathcal{C}$ for $\langle \phi_i \rangle$. According to Proposition 7.2 we can do this by solving an instance of the FCFC problem (with $\forall i : \phi_i^{\min} := \phi_i$) . Next, we solve another instance I_2 of the FCFC problem with the parameters set as follows. We set minimal salaries of the agents from $N_{\mathcal{C}}$ to their asking salaries ($\forall_{i \in N_{\mathcal{C}}} \phi_i^{\min} := \phi_i$). The minimal salaries of the agents outside of $N_{\mathcal{C}}$ are left unmodified. If the solution to I_2 consists of the members of $N_{\mathcal{C}}$ only, we claim that a vector $\langle \phi_i \rangle, i \in N$ is a Strong Nash Equilibrium. Otherwise, it is not. □

The proof of Theorem 7.3 is constructive, but it requires considering all feasible coalitions and, so, leads to potentially high computational complexity. On the other hand, if the salaries of the agents can be rational numbers, we can find the salary function in SNE by a polynomial reduction to the FCFC problem. This result is particularly interesting if the salaries of the agents have high granularity; rounding such a rational solution gives an integral solution which is nearly perfect.

Proposition 7.6. *In the project salary model, if the salaries of the agents are rational, then finding a Strong Nash Equilibrium can be solved in time $O(n^3 \log(nv) fcfc))$, where fcfc is the complexity of the problem* FCFC.

Proof. First, we solve a single instance of the FCFC problem to find $N^* = N_{\mathcal{C}^*(N)}$. Next, similarly as in the proof of Theorem 7.3, we introduce the variables $\phi_i, i \in N^*$ and the inequalities (also the same as in the proof of Theorem 7.3). If we find the values $\phi_i, i \in N^*$ satisfying all the inequalities, then the values $\phi_i, i \in N^*$, together with the minimal salaries of the agents outside of N^*, will form a Strong Nash Equilibrium.

The set of inequalities given in the proof of Theorem 7.3 is a linear program; there are, however, exponentially many constraints (a constraint for each possible coalition). We construct a separation oracle by a polynomial reduction to the FCFC problem. Since ellipsoid method [163] requires $O(n^3 L)$ calls to the separation oracle [104] (where L is the size of the representation of the problem; here $L = O(\log(nv))$), this allows us to solve the linear program in time $O(n^3 \log(nv) fcfc)$.

To check whether all the inequalities are satisfied, it is sufficient to solve FCFC with the following parameters. The minimal salaries of the agents from N^* are set to the values of the variables ϕ_i ($\forall_{i \in N^*} \phi_i^{\min} := \phi_i$). The minimal salaries of the agents outside of N^* are left unmodified. Let $\mathcal{C}$ denote the solution of such instance of the FCFC problem. There exists a not-satisfied inequality if and only if $N_{\mathcal{C}} \neq N^*$. The not-satisfied inequality is the inequality that corresponds to the coalition $\mathcal{C} \neq \mathcal{C}^*$. This completes the proof. □

7.4 Decentralized Formation of Coalitions

Let us assume that the agents can communicate and agree their strategies. Consequently, they can form coalitions and bid for the project as consortiums. We show the concept of a (rigorously) strongly winning coalition, in which no subset of agents can successfully deviate. We show how to characterize (rigorously) strongly winning coalitions and how to reduce the problem of finding them to the FCFC problem. We show that the strongly winning coalitions may not exist, and so we introduce the concept of a weakly winning coalition. We prove that a weakly winning coalition always exists. We demonstrate how to reduce the problem of finding winning coalitions to the FCFC problem.

We model the behavior of the agents as a strategic game. Agent i's action is a triple $\langle N_{\mathcal{C}}, \phi_{\mathcal{C}}, b_{\mathcal{C}} \rangle$. Intuitively, such an action means that the agent i decides to enter the coalition $\mathcal{C} = \langle N_{\mathcal{C}}, \phi_{\mathcal{C}}, b_{\mathcal{C}} \rangle$. The payoff of the agent is equal to $\phi_{\mathcal{C}}(i)$ if (i) $\mathcal{C}$ is feasible, (ii) each agent $j \in N_{\mathcal{C}}$ agrees to participate in $\mathcal{C}$ (i.e., they all play $\mathcal{C}$, and their payoffs are consistent with the bid of the coalition $b_{\mathcal{C}}$), and (iii) there is no feasible cheaper coalition $\mathcal{C}'$ such that all the agents from $N_{\mathcal{C}'}$ agree to participate in $\mathcal{C}'$. Otherwise, the payoff of i is 0.

7.4.1 Strongly Winning Coalitions

In this game the payoffs depend on whether the others agree to cooperate, thus the Strong Nash Equilibrium (SNE) rather than the Nash Equilibrium [217] should be considered. In the following definition we propose an even more stable equilibrium concept than the SNE—the Rigorously Strong Nash Equilibrium (RSNE). The RSNE requires that no subset of agents can deviate in a way that each agent would get a payoff *at least as good* as (instead of strictly better) than prior to deviation. Our approach is motivated by considering careful agents. In an SNE, the agents have no incentive to deviate if they get the same payoff; however they also have no incentive not to deviate. Yet, any deviation will result in a serious payoff loss for some agents (changing their payoffs from a positive ϕ to zero). A careful agent will prefer not to be exposed to the possibility of such loss.

Definition 7.3. *The vector of actions π is a* Rigorously Strong Nash Equilibrium (RSNE) *if and only if there is no subset of agents $N_{\mathcal{C}}$ such that the agents from $N_{\mathcal{C}}$ can make a collaborative action $\mathcal{C}$ (a set of actions played by agents) after which the payoff of each agent i from $N_{\mathcal{C}}$ would be at least equal to her payoff under π and the payoff of at least one agent $i \in N$ would improve.*

In the above definition the requirement that the payoff of at least one agent $i \in N$ must change after the coalition deviates ensures that we treat as equivalent the coalitions with the same payoffs. For instance, assuming a system with three agents, a, b and c, if the coalition $\{a, b\}$ gets a positive payoff, it does not matter whether c plays $\langle \{c\}, v+1 \rangle$ or $\langle \emptyset, v+1 \rangle$: in both cases all payoffs are the same.

Below we introduce additional definitions that help to characterize the Rigorously Strong Nash Equilibria in our games.

Definition 7.4. *A feasible coalition $\mathcal{C}$ is* explicitly endangered *by a coalition $\mathcal{C}'$ if (i) $\mathcal{C}'$ is feasible, (ii) $N_{\mathcal{C}} \cap N_{\mathcal{C}'} = \emptyset$ and (iii) $\mathcal{C}'$ is cheaper than $\mathcal{C}$.*

A feasible coalition $\mathcal{C}$ is implicitly endangered *by a coalition $\mathcal{C}'$ if (i) $\mathcal{C}'$ is feasible, (ii) $N_{\mathcal{C}} \cap N_{\mathcal{C}'} \neq \emptyset$ and each agent from $N_{\mathcal{C}} \cap N_{\mathcal{C}'}$ gets in $\mathcal{C}'$ at least as good a salary as in $\mathcal{C}$, and (iii) either $N_{\mathcal{C}} \neq N_{\mathcal{C}'}$ or $\phi_{\mathcal{C}} \neq \phi_{\mathcal{C}'}$.*

If there are agents belonging to both coalitions ($N_{\mathcal{C}} \cap N_{\mathcal{C}'} \neq \emptyset$), we do not consider the total cost of the alternative coalition $\mathcal{C}'$, as the decision whether $\mathcal{C}'$ will be formed depends solely on the agents from $N_{\mathcal{C}} \cap N_{\mathcal{C}'}$: if they decide to form $\mathcal{C}'$, $\mathcal{C}$ will not be formed, thus the client won't be able to choose between $\mathcal{C}$ and $\mathcal{C}'$.

Informally, a coalition is (rigorously) strongly winning if it constitutes a (rigorous) Strong Nash Equilibrium, i.e., the members will not deviate to other coalitions.

Definition 7.5. *A feasible coalition $\mathcal{C}$ is* rigorously strongly winning *if and only if there is a Rigorously Strong Nash Equilibrium in which the agents from $N_{\mathcal{C}}$ get positive payoffs $\phi_{\mathcal{C}}$.*

Definition 7.6. *A feasible coalition $\mathcal{C}$ is* strongly winning *if and only if there is a Strong Nash Equilibrium in which the agents from $N_{\mathcal{C}}$ get positive payoffs $\phi_{\mathcal{C}}$.*

The following theorem connects the intuitive notion of endangerment with the notion of a winning coalition.

Theorem 7.7. *The coalition $\mathcal{C}$ is rigorously strongly winning if and only if $\mathcal{C}$ is not explicitly nor implicitly endangered by any coalition.*

Proof. $\Longleftarrow$ Assume that there exists a rigorously strongly winning coalition $\mathcal{C}$; thus there exists a Rigorously Strong Nash Equilibrium *RSNE* in which the agents from $N_{\mathcal{C}}$ get positive payoffs. This implies that the agents from $N_{\mathcal{C}}$ agree on the action $\langle N_{\mathcal{C}}, \phi_{\mathcal{C}}, b_{\mathcal{C}} \rangle$; other agents $(N \setminus N_{\mathcal{C}})$ have zero payoffs. For the sake of contradiction let us assume that there exists a feasible coalition $\mathcal{C}'$ such that $\mathcal{C}$ is explicitly or implicitly endangered by $\mathcal{C}'$.

If $N_{\mathcal{C}} \cap N_{\mathcal{C}'}$ is empty ($\mathcal{C}$ is explicitly endangered by $\mathcal{C}'$), then $N_{\mathcal{C}'}$ must be cheaper. This however contradicts the assumption that the agents from $N_{\mathcal{C}}$ get positive payoffs.

Assume thus that $N_{\mathcal{C}} \cap N_{\mathcal{C}'}$ is non-empty (i.e., $\mathcal{C}$ is implicitly endangered by $\mathcal{C}'$). Consider the following collaborative action of agents $(N \setminus N_{\mathcal{C}}) \cup N_{\mathcal{C}'}$. All the agents from $N_{\mathcal{C}'}$ make action $\mathcal{C}'$. Each agent i from $N \setminus (N_{\mathcal{C}} \cup N_{\mathcal{C}'})$ makes an action $\langle \{\}, \phi_{\emptyset} \rangle$, where $\phi_{\emptyset}$ is an empty function. We show that after playing this action no agent from $(N \setminus N_{\mathcal{C}}) \cup N_{\mathcal{C}'}$ will get lower payoff and that some agents will get a strictly better payoff (which will contradict the assumption that *RSNE* is a Rigorously Strong Nash Equilibrium). Clearly each agent from $N \setminus (N_{\mathcal{C}} \cup N_{\mathcal{C}'})$ does not decrease her payoff (as previously it was equal to 0). Now, we show that the agents from $N_{\mathcal{C}'}$ will get at least the same payoff as before. Since we know that $\mathcal{C}$ is implicitly endangered by $\mathcal{C}'$ (and thus the agents from $N_{\mathcal{C}} \cap N_{\mathcal{C}'}$ get in $\mathcal{C}'$ at least as good payoff as in $\mathcal{C}$) it is sufficient to show that the agents from $N_{\mathcal{C}'}$ will get positive payoffs. Indeed, there is no feasible coalition that includes some agents from $N \setminus (N_{\mathcal{C}} \cup N_{\mathcal{C}'})$ (as these agents play $\{\}$). Also, the agents from $N_{\mathcal{C}} \setminus N_{\mathcal{C}'}$ do not agree on the collaborative action (they still play $\mathcal{C}$) and thus, cannot form a feasible coalition. Thus, after such change of played actions $\mathcal{C}'$ is the only feasible coalition that the members agreed on. Finally, we can show that at least one agent will get a strictly better payoff. Either $N_{\mathcal{C}} = N_{\mathcal{C}'}$ (and since $\phi_{\mathcal{C}} \neq \phi_{\mathcal{C}'}$, some agent must get a different payoff) or $N_{\mathcal{C}} \neq N_{\mathcal{C}'}$ (and the agents from $N_{\mathcal{C}'} \setminus N_{\mathcal{C}}$ will get a positive payoff).

$\Longrightarrow$ Assume that $\mathcal{C}$ is not explicitly nor implicitly endangered by any coalition. First, if the agents from $N_{\mathcal{C}}$ make the collaborative action $\mathcal{C}$, then they will all get positive payoffs. Indeed, the agents in $N_{\mathcal{C}}$ could not get positive payoffs only if there would exist a cheaper feasible coalition $\mathcal{C}'$ such that $N_{\mathcal{C}} \cap N_{\mathcal{C}'} = \emptyset$. This would, however mean that $\mathcal{C}$ is explicitly endangered by $\mathcal{C}'$. Next, we show that the state in which the agents from $N_{\mathcal{C}}$ make the collective decision $\mathcal{C}$ and the other agents play arbitrary actions is RSNE. For the sake of contradiction let us assume that there exists a subset of agents $N_{\mathcal{C}'}$ which can make a collaborative action $\mathcal{C}'$ after which the payoff of everyone from $N_{\mathcal{C}'}$ would be at least equal to her payoff in $\mathcal{C}$. This would, however mean that $\mathcal{C}$ is either implicitly or explicitly endangered by $\mathcal{C}'$. This completes the proof. $\square$

The result in Theorem 7.8 stated for Rigorously Strong Nash Equilibria transfers to Strong Nash Equilibria with a slight modification of the model. It is sufficient to assume that the payoff of an agent playing an empty coalition receives slightly higher payoff than by playing non-empty losing coalition. In other words, this modification associates some small costs with the preparation of a bid by the agents. Hereinafter, whenever we mention a strictly winning coalition we assume that the agents have small but positive cost of preparing the offer. To state the result for Strong Nash Equilibria we also need to use the definition of a coalition $\mathcal{C}$ being *strictly implicitly endangered* by $\mathcal{C}'$. This definition differs from being implicitly endangered only by the fact that we do not require the agents from $N_{\mathcal{C}} \cap N_{\mathcal{C}'}$ to have at least as good payoffs, but strictly better payoffs in $\mathcal{C}'$ than in $\mathcal{C}$.

Theorem 7.8. *If there are small but positive costs of preparing the offer by the agents then the coalition $\mathcal{C}$ is strongly winning if and only if $\mathcal{C}$ is not explicitly nor strictly implicitly endangered by any coalition.*

Proof. The proof is analogous to the proof of Theorem 7.7. □

Theorems 7.7 and 7.8 give us a better understanding of the concept of Rigorously Strong Nash Equilibrium (and Strong Nash Equilibrium) in our model. They also lead to a simple brute-force algorithm for checking whether the coalition can be a part of some RSNE. Below, we provide the analysis that allows to characterize RSNEs in the project salary model even more precisely.

Theorem 7.9. *In the project salary model, the set of agents participating in a rigorously strongly winning coalition is the same as the set of agents participating in the cheapest feasible coalition.*

Proof. Let $\mathcal{C}$ denote the cheapest feasible coalition. We show that for any other coalition $\mathcal{C}'$, such that $N_{\mathcal{C}} \neq N_{\mathcal{C}'}$, $\mathcal{C}'$ cannot be rigorously strongly winning. For the sake of contradiction let us assume that $\mathcal{C}'$ is rigorously strongly winning. Let $N_{\cap} = N_{\mathcal{C}} \cap N_{\mathcal{C}'}$. Since $\mathcal{C}$ is the cheapest, the sum of salaries of the agents from $N_{\mathcal{C}} \setminus N_{\cap}$ in $\mathcal{C}$ is lower or equal to the sum of salaries of the agents from $N_{\mathcal{C}'} \setminus N_{\cap}$ in $\mathcal{C}'$. Consider a coalition $\mathcal{C}''$ consisting of the set of agents $N_{\mathcal{C}}$ and the following salary function. The salary of each agent from $N_{\mathcal{C}} \setminus N_{\mathcal{C}'}$ is the same as in $\mathcal{C}$ and the salary of each agent from $N_{\cap}$ is the same as in $\mathcal{C}'$. Since the bid $c_{\mathcal{C}'}$ of $\mathcal{C}'$ was below v, the bid of $\mathcal{C}''$ is also below v. Thus, $\mathcal{C}''$ is feasible. Also, $\mathcal{C}'$ is implicitly endangered by $\mathcal{C}''$, which leads to contradiction and completes the proof. □

Interestingly, there is no analogous result for the hourly salary model.

Proposition 7.10. *In the hourly salary model the set of the agents participating in a rigorously strongly winning coalition may not be the same as the set of the agents participating in the cheapest feasible coalition.*

Proof. Let us consider 3 agents a, b, and c with equal minimal salaries $\phi_a^{\min} = \phi_b^{\min} = \phi_c^{\min} = 1$. Agents a and b can complete the project spending $t_a = 10$ and $t_b = 10$

time units on it. Also, the agents b and c can complete the project spending $t_b = 4$ and $t_c = 20$ time units on it. The maximal budget is $v = 28$. The cheapest coalition is $\{a, b\}$ with salaries $\phi_a = \phi_b = 1$ and bid equal to 20. However, the coalition $\{b, c\}$ with salaries $\phi_b = 2$ and $\phi_c = 1$ is rigorously strongly winning. □

Proposition 7.11. *In the project salary model the bid of a strongly winning coalition is equal to the maximal allowed price v.*

Proof. Let $\mathcal{C}$ be a strongly winning coalition. If $b_{\mathcal{C}} < v$ we could increase the salaries of some participating agents. The resulting coalition would implicitly endanger $\mathcal{C}$. □

Proposition 7.11 is a consequence of the fact that the coalition $\mathcal{C}$ can be endangered by a coalition $\mathcal{C}'$ having the same set of the participants but different salaries. An alternative characterization disallows such behavior; we do not explore this path further in this chapter.

Theorem 7.9 and Proposition 7.11 show that the problem of finding a strongly winning coalition collapses to the problem of finding a feasible coalition. The problem, thus, becomes an optimization problem; the strategic behavior of agents does not have an influence on this procedure.

Proposition 7.12. *Checking whether a coalition is rigorously strongly winning can be solved in time $O(n^2 \cdot fcfc)$, where fcfc is the complexity of the problem* FCFC.

Proof. Let us assume that we want to check whether the coalition $\mathcal{C}$ is rigorously strongly winning. First, we check whether we can increase the salary of any agent so that the coalition would still be feasible. If we can, $\mathcal{C}$ is not rigorously strongly winning. Otherwise, we solve FCFC for the set of agents $N \setminus N_{\mathcal{C}}$. If there exists a non-empty solution $\mathcal{C}'$ with the cost $c_{\mathcal{C}'} < c_{\mathcal{C}}$ or such that $c_{\mathcal{C}'} = c_{\mathcal{C}}$ and $\mathcal{C}' \prec \mathcal{C}$, this means that $\mathcal{C}$ is explicitly endangered by $\mathcal{C}'$, and thus is not rigorously strongly winning. Otherwise, $\mathcal{C}$ is not explicitly endangered by any coalition.

Next, we check whether $\mathcal{C}$ is implicitly endangered by some coalition $\mathcal{C}'$. We change the names of the agents so that the agents from $N_{\mathcal{C}}$ were the first $\|N_{\mathcal{C}}\|$ agents in the lexicographic order. Now, for each agent i from $N_{\mathcal{C}}$ we do the following procedure. We solve FCFC for the set of agents $N \setminus \{i\}$, for the minimal salaries of the agents from $N_{\mathcal{C}}$ changed to their salaries in $\mathcal{C}$, and for the budget v set to $c_{\mathcal{C}}$. If there exists a feasible $\mathcal{C}'$ to FCFC such that the set $N_{\mathcal{C}'}$ overlaps with $N_{\mathcal{C}}$ (overlapping can be tested in time $O(n)$), then $\mathcal{C}$ is implicitly endangered by $\mathcal{C}'$. We already know that there is no non-overlapping coalition with the cost lower than $c_{\mathcal{C}}$. Thus, if for no agent i from $N_{\mathcal{C}}$ we find such implicitly endangering coalition, this means that there is no feasible coalition $\mathcal{C}'$ such that $N_{\mathcal{C}} \cap N_{\mathcal{C}'} \neq \emptyset$. Thus, in such case we conclude that $\mathcal{C}$ is rigorously strongly winning. □

Proposition 7.13. *In the project salary model, if the salaries of the agents can be rational numbers, finding a rigorously strongly winning coalition can be solved in time $O(n^5 \log(nv) fcfc)$, where fcfc is the complexity of the problem* FCFC.

Proof. First we solve FCFC to find the cheapest coalition $\mathcal{C}$. We know that the set of the agents participating in a rigorously strongly winning coalition is $N_{\mathcal{C}}$ (Theorem 7.9) and the total cost of such a coalition is v (Proposition 7.11). We only need to find the salary function of such a coalition. For every agent i from $N_{\mathcal{C}}$, we introduce a variable $\phi_{\mathcal{C}}(i)$. We will show the linear program for the variables $\phi_{\mathcal{C}}(i)$, to which the solution is a rigorously strongly winning coalition. At the same time we will show how to implement the separation oracle for the linear program.

First equality states that the salaries of the agents satisfy the feasibility constraint:

$$\sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}(i) = v \tag{7.4}$$

Next two inequalities model explicit endangerment. For each coalition $\mathcal{C}'$, such that $N_{\mathcal{C}} \cap N_{\mathcal{C}'} = \emptyset$ and $\mathcal{C}' \prec \mathcal{C}$:

$$\sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}(i) < \sum_{i \in N_{\mathcal{C}'}} \phi_i^{\min}. \tag{7.5}$$

For each coalition $\mathcal{C}'$, such that $N_{\mathcal{C}} \cap N_{\mathcal{C}'} = \emptyset$ and $\mathcal{C} \prec \mathcal{C}'$:

$$\sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}(i) \leq \sum_{i \in N_{\mathcal{C}'}} \phi_i^{\min}. \tag{7.6}$$

Note that we can check the above two inequalities by solving FCFC problem for the set of agents $N \setminus N_{\mathcal{C}}$. If the resulting coalition $\mathcal{C}'$ is cheaper than $\mathcal{C}$, this means that the inequality constraint for $\mathcal{C}'$ was violated. Otherwise, all the above inequalities are satisfied.

Last, for each coalition $\mathcal{C}'$, such that $N_{\mathcal{C}} \cap N_{\mathcal{C}'} \neq \emptyset$ and $N_{\mathcal{C}} \neq N_{\mathcal{C}'}$ we introduce the inequality modeling implicit endangerment:

$$\sum_{i \in N_{\mathcal{C}} \setminus N_{\mathcal{C}'}} \phi_{\mathcal{C}}(i) + \sum_{i \in N_{\mathcal{C}'} \setminus N_{\mathcal{C}}} \phi_i^{\min} > v. \tag{7.7}$$

We can check this inequality in the same way as we checked whether the coalition was implicitly endangered in the proof of Proposition 7.12: by swapping the names of the agents, for each $i \in N_{\mathcal{C}}$ solving FCFC for the set of agents $N \setminus \{i\}$, and checking the overlapping of the appropriate sets. The whole procedure requires the time $O(n^2 \cdot fcfc)$.

As the result, we showed the reduction of the problem of finding a rigorously strongly winning coalition to the linear program with n variables and a separation oracle running in time $O(n^2 \cdot fcfc)$. □

It is desired to know Rigorously Strong Nash Equilibrium provided they exist. However a RSNE (and even a Strong Nash Equilibrium) may not exist in some instances.

Proposition 7.14. *Both in the project salary and in the hourly salary model, there may not exist a strongly winning coalition even though there exists a feasible coalition.*

Proof. Consider a project with budget $v = 5$; and three identical agents a, b, c with minimal salaries $\phi_i^{\min} = 2$ (in the hourly salary model, assume that each agent spends exactly 1 time unit on the project); a coalition of any two agents is feasible (able to complete the project on time and within the budget).

For the sake of contradiction assume there exists a coalition $\mathcal{C}$ that gets positive payoffs. Without loss of generality we assume that $N_{\mathcal{C}} = \{a, b\}$. At least one of the agents, let us say a has to get salary equal to 2. However, the agents a and c, with the salaries equal to 3 and 2 respectively, can form a feasible coalition in which both a and c get better payoffs (note that we use here the fact that the payoffs are discrete). $\square$

7.4.2 Weakly Winning Coalitions

We are not fully satisfied with the example from Proposition 7.14. Indeed, the coalition $\{a, c\}$ can profit by deviating, e.g., by playing $\phi(a) = 2$ and $\phi(c) = 1$. On the other hand, a should not be wiling to deviate—the reason is that $\{a, c\}$ with payoffs $\phi(a) = 2$ and $\phi c = 1$ is not stable by its own (for instance, the coalition $\{b, c\}$ can play $\phi(b) = 1$ and $\phi(c) = 2$, and successfully deviate). In the above example no coalition will be formed even though intuitively we feel that there are coalitions that would agree to work. Thus, we propose a weaker notion of a winning coalition.

Definition 7.7. *A feasible coalition $\mathcal{C}$ is* weakly winning *if it is not explicitly endangered by any coalition and for each feasible coalition $\mathcal{C}'$ such that $\mathcal{C}$ is implicitly endangered by $\mathcal{C}'$, there exists a feasible coalition $\mathcal{C}''$ such that $\mathcal{C}'$ is explicitly or implicitly endangered by $\mathcal{C}''$.*

Proposition 7.15. *There exists a weakly winning coalition if and only if there exists a feasible coalition.*

Proof. Consider a feasible coalition $\mathcal{C}$ that is not explicitly endangered (such a coalition exists provided there exists a feasible coalition). Let $\mathcal{E}$ denote a set of feasible coalitions implicitly endangering $\mathcal{C}$. If $\mathcal{E} = \emptyset$, $\mathcal{C}$ is strongly winning and, thus also, weakly winning. If there exists $\mathcal{C}' \in \mathcal{E}$ such that $\mathcal{C}'$ is not (implicitly or explicitly) endangered by any feasible coalition, then $\mathcal{C}'$ is strongly winning (and, thus also, weakly winning). Otherwise, $\mathcal{C}$ is weakly winning.

If there is no feasible coalition then there is no weakly winning coalition. $\square$

Proposition 7.16. *In the project salary model, if the salaries of the agents can be rational numbers, the problem of finding a weakly winning coalition can be solved in time $O(n^5 \log(nv) fcfc)$, where fcfc is the complexity of the problem* FCFC.

Proof. First, we look for a rigorously strongly winning coalition. If there is one, it is also weakly winning, and so the procedure is complete. If there is no rigorously strongly winning coalition it is sufficient to find a coalition that is not explicitly endangered by any other coalition. We can do this by solving a single instance of the FCFC problem. □

Proposition 7.17. *In the project salary model, if the salaries of the agent can be rational numbers, the problem of checking whether a coalition $\mathcal{C}'$ is weakly winning coalition can be solved in time $O(n^5 \log(nv) fcfc)$, where fcfc is the complexity of the problem* FCFC.

Proof. We first check whether the coalition is explicitly endangered by any other coalition. We can do this by solving a single instance of the FCFC problem for the set of agents $N \setminus N_{\mathcal{C}'}$.

Next we look for a rigorously strongly winning coalition that endangers $\mathcal{C}'$. We do this in the same way as in the proof of Proposition 7.13. The only difference is that we additionally introduce the following inequalities. We assume the same notation as in the proof of Proposition 7.13. For each $i \in N_{\mathcal{C}} \cap N_{\mathcal{C}'}$ we require: $\phi_{\mathcal{C}}(i) \geq \phi_{\mathcal{C}'}(i)$. □

7.4.3 Other Solution Concepts

In this section we give a brief overview of other solution concepts that can be applied to describe winning coalitions in our games. Most of these solution concepts have their drawbacks and they do not allow to determine winning coalitions. On the other hand, we point out two ideas that, we believe, are interesting for further analysis. The first idea is to apply the concept of the Coalitional Farsighted Conservative Stable Set [82] to our setting. The second is to apply the concepts inspired by the graph interpretations. These two solution concepts are, however, more involved, and, so, we believe that our definition of a weakly winning coalition is the natural simplification, and the first step to understand the complexity of the agents' interactions.

In the following subsections we present the discussion on the application of different solution concepts to our model.

Cooperative Game Theory Approach

It may seem that our solution concepts are closely related to solution concepts from the cooperative game theory. For instance, the definition of Rigorously Strong Nash Equilibrium is close in spirit to the concept of the core from the cooperative games. However, there are some substantial differences. In cooperative game theory it is commonly assumed that the value of a coalition depends only on the members of this coalition. The following example shows that this is not the case in our problem.

Example 7.18. Consider 2 agents a and b with the minimal salaries $\phi_a^{\min} = 1$ and $\phi_b^{\min} = 2$. The maximal budget of the issuer is $v = 2$. Consider two coalitions formed by single agents $\mathcal{C}_1 = \{a\}$, and $\mathcal{C}_2 = \{b\}$. Let us assume that $\mathcal{C}_2$ is feasible. The value of $\mathcal{C}_2$ depends on whether the agent $\mathcal{C}_1$ is feasible or not.

The above example encourages one to consider our problem as a cooperative game with externalities. However, in such games the values of the coalitions depend only on the partition of the agents into coalitions. In our case, however, the whole coalitions are strategic, and their values depend on the actions (the bids) of the other coalitions. We provide a detailed discussion regarding applicability of selected concepts from cooperative game theory in the two following subsections.

The Core

Although the notion of *the core* is initially known from the cooperative game theory, there is a natural generalization to strategic games. In this generalization we say that coalition $\mathcal{C}$ with payoff function ϕ is in the core if and only if there is no feasible coalition $\mathcal{C}'$ with payoff function ϕ' such that every agent in $\mathcal{C}'$ gets, according to ϕ', a better payoff than according to ϕ.

Although, in cooperative game theory we use a simplified model in which feasibility means just that the total payoff of the agents does not exceed the value of the coalition (i.e., the bid of the coalition, in our approach), we may use the more demanding notion of feasibility from our model. As a result, a coalition $\mathcal{C}$ is in the core if and only if it is not implicitly endangered by any other coalition.

Intuitively, the notion of the core in our games is missing an important element. Indeed, a coalition $\mathcal{C}$ might be in the core even though some other coalition $\mathcal{C}'$, disjoint with $\mathcal{C}$, can offer a better price and, consequently, win the auction and be awarded the project.

The (Farsighted) Stability

Another notion known from the cooperative game theory that is worth considering is the von Neumann-Morgenstern stable set. The stable set is the set of all payoff vectors such that (i) no payoff vector in the stable set is dominated by another vector in the set, and (ii) all payoff vectors outside the set are dominated by at least one vector in the set.

In the light of our previous example from Proposition 7.14, it is even more appealing to consider the farsighted von Neumann-Morgenstern stable set [61]. A farsighted coalition is more deliberative, it considers that if it makes a deviation, the second coalition might react as a consequence of the first coalition's action, next the third coalition might react, and so on without the limit. In the original formulation the agents are considered to be optimistic—they are willing to deviate if the deviation starts some sequence of deviations that would lead to a better outcome.

In our games the vN-M stable set, and the farsighted vN-M stable set, might be empty.

Example 7.19. Consider the example from Proposition 7.14. There is a project with the budget $v = 5$; and three identical agents a, b, c with minimal salaries $\phi_i^{\min} > 2$. Every coalition formed by any two agents is feasible. For the sake of clarity of the presentation let us assume that the payoffs of the agents can be the natural numbers only. Let us consider the coalition $\mathcal{C}_1 = \{a, b\}$ with the payoffs $\phi^a = 3$, and $\phi^a = 2$. If the coalition $\mathcal{C}_1$ is in the stable set, then the coalition $\mathcal{C}_2\{b, c\}$ with the payoffs $\phi^b = 3$, and $\phi^c = 3$, which dominates $\mathcal{C}_2$, must not be in the stable set (otherwise it would contradict the internal stability requirement). Since $\mathcal{C}_2$ does not belong to the stable set, and it is dominated only by the coalition $\mathcal{C}_3 = \{a, c\}$ with the payoffs $\phi^a = 2$, and $\phi^c = 3$, we infer that $\mathcal{C}_3$ must belong to stable set. However, $\mathcal{C}_3$ is dominated by $\mathcal{C}_1$, which leads to contradiction. By symmetry, we see that the stable set is empty.

The same reasoning as given in the example above applies to the farsighted vN-M stable sets. The alternative definition in which the agents are conservative, the Coalitional Farsighted Conservative Stable Set, was proposed by Diamantoudi and Xue [82]. Intuitively, in this definition the agents are willing to deviate only if every sequence starting from this deviation leads to a better outcome for them.

We believe that these two cases consider too extreme behavior of the agents. Nevertheless, we think that considering coalitional farsighted conservative stable sets in our game is a very appealing direction for the future work.

Coalition-Proof Nash Equilibria

Another way of weakening the notion of the (rigorously) strongly winning coalition is to consider Coalition-Proof Nash Equilibria [23]. Intuitively, in the Coalitional-Proof Nash Equilibrium we first assume that all players are in a common room, where they can freely discuss their strategies. Then the agents, one by one, leave the room. Once an agent leaves the room, she cannot change her strategy. The agents that are left in the room are allowed to discuss and (cooperatively) change their strategies.

Unfortunately, these equilibria are not guaranteed to exist. This is what we expect since a Coalition-Proof Nash Equilibrium must be essentially a Nash Equilibrium. For the sake of completeness of the presentation, below we show an appropriate example in which there is no Coalition-Proof Nash Equilibrium.

Example 7.20. Consider the example from Proposition 7.14. There is a project with the budget $v = 5$; and three identical agents a, b, c with minimal salaries $\phi_i^{\min} > 2$. Every coalition formed by any two agents is feasible. There is no Coalition-Proof Nash Equilibrium in this example (independently whether the salaries of the agents are natural or rational numbers). Indeed, consider any vector of payoffs $\langle \phi^a, \phi^b, \phi^c \rangle$. If $\phi^a > 2$, we infer that a forms a winning coalition with one of the agents b, or c. Without loss of generality we assume that $\{a, b\}$ is the winning team. Thus, $\phi^b < 3$ and $\phi^c = 0$. If we consider the subgame formed by the agents b and c, we see, however, that their payoff vector $\langle \phi^b, \phi^c \rangle$ is Pareto-dominated by $\langle 3, 2 \rangle$. Now, let us consider the case when $\phi^a < 2$. One of the agents b and c needs to have payoff lower than 3 (w.l.o.g let us assume that this is the agent b). But, if we consider the subgame formed by the agents a and b, their payoff vector $\langle \phi^a, \phi^b \rangle$ is Pareto-dominated by $\langle 2, 3 \rangle$. Finally, let us assume that $\phi^a = 2$. We infer that one of the agents b and c gets zero payoff (let us assume that this is the agent b). However, the payoff vector $\langle \phi^a, \phi^b \rangle$ is Pareto-dominated by $\langle 3, 2 \rangle$.

Graph Interpretations

Let us consider a directed multi-graph in which the vertices are the strategy profiles. Each pair of vertices can be connected with at most two edges, corresponding to implicit and explicit endangerment. Thus, vertices v and u are connected by an edge corresponding to the implicit endangerment if and only if v is implicitly endangered by u. Analogously, v and u are connected by an edge corresponding to the explicit endangerment if and only if v is explicitly endangered by u.

Clearly, in such a graph, strong Nash equilibria correspond to the sinks, the vertices with no outgoing edges. Also, the edges corresponding to explicit endangerment do not form cycles. Consequently, we can restrict our graphs to these induced by the vertices that do not have outgoing edges corresponding to the explicit endangerment. We believe that every connected component in such restricted graphs defines an interesting set of stable solutions. We plan to analyze this idea in our future work. For instance, thus defined set of stable solutions is always non-empty and its elements correspond to weakly winning coalitions.

7.5 Mechanism Design

In this section we take a look at two mechanisms that a project issuer can apply to find a winning coalition: the first one sets the job's budget v; the second one uses a first-price auction.

First, we show that if the client is allowed to change the value v there exists a simple mechanism ensuring the existence of a strongly winning coalition.

Theorem 7.21. *If there exists a feasible coalition, then there exists a budget v^* for which there exists a strongly winning coalition. The problem of finding such v^* can be solved in time $O(\log v \cdot \mathit{ffc})$, where ffc is the complexity of the problem* FFC.

Proof. Let v^* be the smallest value such that there exists a feasible coalition. We show that for v^* there exists a strongly winning coalition. Let $\mathcal{C}^*$ be the most preferred (according to the tie-breaking rule $\prec$) feasible coalition for v^*. For the sake of contradiction let us assume that there exists a coalition $\mathcal{C}'$ such that $\mathcal{C}^*$ is strictly implicitly or explicitly endangered by $\mathcal{C}'$. Of course $b_{\mathcal{C}'} \leq v^*$ (otherwise $\mathcal{C}'$ would not be feasible). If $\mathcal{C}^*$ is explicitly endangered by $\mathcal{C}'$ ($N_{\mathcal{C}^*} \cap N_{\mathcal{C}'} = \emptyset$), it means $\mathcal{C}'$ is cheaper than $\mathcal{C}^*$; and we get a contradiction with the definition of v^*. Otherwise

($\mathcal{C}^*$ is strictly implicitly endangered by $\mathcal{C}'$), let $i \in N_{\mathcal{C}^*} \cap N_{\mathcal{C}'}$. Now, i must get strictly better salary in $\mathcal{C}'$ than in $\mathcal{C}^*$. Thus if we change the salary of i in the coalition $\mathcal{C}'$ to $\phi_{\mathcal{C}'}(i) = \phi_{\mathcal{C}^*}(i)$ we get a contradiction—a cheaper feasible coalition.

To find such a v^*, one has to run a binary search over v. □

In the second approach we use the first-price auction in which coalitions participate. In a standard first-price auction, an item's price starts from some minimal value (the least preferred outcome for the owner of the item). Bidders place bids for the current price. The asking price is gradually increased until there are no further bids; the last bidder wins the auction. Similarly, in our proposed auction, the auction starts from the original budget v (the least preferred outcome for the client); the asking price is gradually *decreased*. Coalitions place bids for the current asking price (as in the standard first-price auction, multiple bids for the same asking price are not allowed). The auction stops if there is no feasible coalition that can propose a lower bid than the current asking price. This leads to the concept of an auction-winning coalition.

Definition 7.8. *A coalition $\mathcal{C}$ is* auction-winning *if and only if there is no feasible coalition $\mathcal{C}'$ such that $b_{\mathcal{C}'} < b_{\mathcal{C}}$ and for each agent $i \in N_{\mathcal{C}} \cap N_{\mathcal{C}'}$ the agent gets better salary in $\mathcal{C}'$, $\phi_{\mathcal{C}'}(i) \geq \phi_{\mathcal{C}}(i)$.*

Proposition 7.22. *The problem of checking whether a feasible coalition $\mathcal{C}$ is auction-winning can be solved in time $O(ffc)$. The problem of finding an auction-winning coalition can be solved in time $O(v \cdot ffc)$; ffc is the complexity of the problem* FFC.

Proof. To check whether a coalition $\mathcal{C}$ is auction-winning one has to solve the problem of existence of the feasible coalition for the asking price: $v = b_{\mathcal{C}} - 1$ (representing the next asking price in the first-price auction); and for each $i \in N_{\mathcal{C}}$ set $\phi_i^{\min} = \phi_{\mathcal{C}}(i)$ (these agents must get at least the same payoffs as in $\mathcal{C}$). If no such coalition exists, $\mathcal{C}$ is auction-winning.

To find an auction-winning coalition one can simply simulate the auction. □

Here, once again, we saw that these problems of finding auction-winning coalitions require solving the problem of finding a feasible coalition. We note that the procedure of finding an auction-winning coalition from Proposition 7.22 might be exponential with respect to the representation of v.

The summary of our results in the general model, i.e., the model where we assume a general oracle capable of solving the FFC problem, is given in Table 7.2. We believe that the computational results favor the concept of the auction winning coalition (or the centralized model). First, the weakly winning coalition is guaranteed to exist. Second, the computational power needed to find an auction winning coalition seems much smaller in comparison with other concepts. In the centralized model, finding the winning coalition (when we already have the asking salaries of the agents) has also a straightforward reduction to FCFC.

Table 7.2: The summary of the results in general model. The column "Existence" contains the information whether a coalition/equilibrium always exists. The column "Checking" contains the complexity of checking whether a given coalition satisfies the definition corresponding to the row. The column "Finding" contains the complexity of finding a coalition/equilibrium (*ffc* and *fcfc* are the complexities of the problems FFC and FCFC, respectively). The values marked as (*) are valid only in the project salary model. The values marked as (+) are valid only in the hourly salary model. The values marked as (-) are valid only if the salaries of the agents can be rational numbers.

		Existence	Checking	Finding
Decentralized	rigorously strongly winning coalition	Not always	$O(n^2 \cdot fcfc)$	$O(n^5 \log(nv) fcfc)$ (*)(-)
	strongly winning coalition	Not always	open problem	
	weakly winning coalition	Always	$O(n^5 \log(nv) fcfc)$ (*)(-)	
	auction winning coalition	Always	$O(ffc)$	$O(v \cdot ffc)$
Central.	winning coalition (having asking salaries)	N/A	$O(fcfc)$	
	Strong Nash Equilibrium	Always (*) Not always (+)	$O(fcfc)$	$O(n^3 \log(nv) fcfc))$ (*)(-)

7.6 Finding Feasible Coalitions in a Scheduling Model

In Sections 7.4 and 7.5 we show that many problems of finding the (weakly/strongly) winning coalitions or determining whether a given coalition is (weakly/strongly) winning require solving the subproblem of finding the feasible coalition. The general model (Section 7.2) assumed that given a coalition there is an oracle deciding whether there exists a feasible coalition.

By specifying an oracle, our results can be applied to two different problems known in the literature: the commodity auctions and the path auctions.

In the commodity auctions setting, the project can be seen as a set of items $I = \{i_1, i_2, \ldots, i_q\}$, where each agent owns a certain subset of the items. A coalition is feasible if the agents have together all the items from I.

In the path auctions setting [223] we are given a graph G with two distinguished vertices: a source s and a target t. The agents correspond to the vertices in the graph. A coalition is feasible if the participating agents form a path from s to t.

In this section we show a possible concrete instance of this model in which a project is a set of indivisible, independent, tasks and agents are processors who process these tasks with varying speeds.

7.6.1 The Scheduling Model

A project consists of a set $\mathcal{T} = \{t_1, t_2, \ldots, t_q\}$ of q independent tasks. The tasks can be processed sequentially or in parallel. The tasks are indivisible: a task must be processed on a single processor. Once started, a task cannot be interrupted. All tasks must be completed before a given time d, the project's deadline.

Agents correspond to processors (in this section we use terms "agent" and "processor" interchangeably). Each agent has certain skills which are represented as the speed of executing the tasks. Thus, for each agent i we define the skill vector $s_i = \langle s_{i,1}, s_{i,2}, \ldots s_{i,q} \rangle$ which has the following meaning: agent i is able to finish task t_j within $s_{i,j}$ time units (with $s_{i,j} = \infty$ when the agent is unable to finish the task). We assume that s_i is known for each agent (it can be well approximated, e.g., from past behavior of the agents certified by clients in form of reviews). An agent can process only a single task at each time moment—if she wants to process more than one task, she must execute the tasks sequentially. We assume that only a single agent can work on a given task. This assumption is not as restrictive as it may appear; if the task t_i is large and can be processed by multiple agents in parallel, the project client will rather replace t_i by a number of smaller tasks.

For a coalition $\mathcal{C}$ we define $\Phi_{\mathcal{C}} : \mathcal{T} \to N_{\mathcal{C}}$ to be an assignment function (assigning tasks to agents). The assignment function $\Phi_{\mathcal{C}}$ enables us to formalize the notion of a coalition completing the project before the deadline and also the total cost of the coalition. Specifically, a project is finished before the deadline d if and only if all the agents finish their assigned tasks before d, $\forall i \in N_{\mathcal{C}} : \sum_{\ell : \Phi(t_\ell) = i} s_{i,\ell} \le d$. In the hourly salary model, the cost of the coalition is equal to $c_{\mathcal{C}} = \sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}(i) \sum_{\ell : \Phi(t_\ell) = i} s_{i,\ell}$.

In the scheduling model we define the problem of finding a feasible coalition as follows.

Problem 7.9 (FFCSM: Find Feasible Coalitions, Scheduling Model)**.** *Let $\mathcal{T}$ be the set of q tasks and N be the set of processors (or equivalently, agents). For each task $t_j \in \mathcal{T}$ and each processor (agent) $i \in N$ we define $s_{i,j}$ as the processing time of t_j on i. Let $\phi_i^{\min}$ be the cost of renting processor i (hiring agent i). The budget of the project is v and the deadline is d. The FFCSM problem consists of selecting a subset of the processors $N' \subseteq N$ and the assignment function $\Phi : \mathcal{T} \to N'$ such that the budget is not exceeded ($c_{N',\Phi} \le B$) and the project's makespan does not exceed the deadline d.*

In the hourly salary model, the problem of finding the feasible coalition reduces to the problem of scheduling on unrelated processors with costs [270]. Specifically, Shmoys and Tardos [270] show a 2-approximation algorithm for approximating the makespan (the deadline d in our model).

Problem 7.10 (FFCHS: Find Feasible Coalitions, Hourly Salary). *The instances of the problem are the same as in the* FFCSM *problem, except that in the* FFCHS *problem we additionally specify that the cost of the coalition $c_{N',\Phi}$ is defined as $c_{\mathcal{C}} = \sum_{i \in N_{\mathcal{C}}} \phi_{\mathcal{C}}(i) \sum_{\ell : \Phi(t_\ell) = i} s_{i,\ell}$.*

The project salary model is a generalization of the problem of minimizing makespan on unrelated processors [293]. To the best of our knowledge, this problem has not been stated before; thus we formally define it below.

Problem 7.11 (FFCPS: Find Feasible Coalitions, Project Salary). *The instances of the problem are the same as in the* FFCSM *problem, except that in the* FFCPS *problem we additionally specify that the cost of the coalition $c_{N',\Phi}$ is defined as $c_{N',\Phi} = \sum_{i \in N'} \phi_i^{\min}$.*

An easier variant of the problem, in which the goal is to optimize the assignment only (assuming that the processors are already selected) has a 2-approximation algorithm [293]. However, adding the notion of the budget usually significantly increases the complexity. We believe that the approximability of FFCPS is a very appealing problem.

7.6.2 FFCPS: Hardness Results

First, we show the NP-hardness of FFC-Scheduling in restricted special cases.

Theorem 7.23. FFCPS *and* FFCHS *are NP-hard even for two agents.*

Proof. The proof is by reduction from the partition problem. In the partition problem, we are given a set of integers $\{n_j\}$; we ask whether there exists a partition of this set into two subsets S_1, S_2, such that $\sum_{n_j \in S1} n_j = \sum_{n_j \in S_2} n_j$. To construct an instance of the feasible coalition problem, we construct a project that has a task for each n_j, an unlimited budget and a deadline $d = 1/2 \sum n_j$. We take two agents a and b with processing speeds $s_{a,j} = s_{b,j} = n_j$ and unit costs: $\phi_a^{\min} = \phi_b^{\min} = 1$. A feasible coalition corresponds with partitioning numbers into two with equal sums. □

Theorem 7.24. FFCPS *is NP-hard even if the agents can be assigned no more than 3 tasks, if each agent has no more than 3 skills (for each j we have that $\|\{i : s_{i,j} \neq \infty\}\| \leq 3$), if the deadline is constant, and if the minimal salaries of the agents are equal 1.*

Proof. The proof is by reduction from the exact set cover problem. In the exact set cover problem we are given a set of elements $T = \{t_1, t_2, \ldots, t_q\}$ and family $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of 3-element subsets of T. We ask whether there exist $\frac{q}{3}$ subsets from $\mathcal{S}$ that cover all the elements from T. The exact set cover problem is NP-hard even if each member of T appears in at most 3 sets from $\mathcal{S}$.

We build an instance of the feasible coalition problem in the following way. There are q tasks and n agents; for each agent i and each task t_j we have that $s_{i,j} = 1$ if and only if $t_j \in S_i$. Otherwise, $s_{i,j} = \infty$. The deadline d is equal to 3. The minimal salary of each agent is 1 and the budget v to $\frac{q}{3}$. It is easy to check that there exists a feasible coalition if and only if there exists a cover of T with $\frac{q}{3}$ sets. □

Theorem 7.25. FFCHS *is NP-hard even if the agents can be assigned no more than 4 tasks, if each agent has no more than 4 skills (for each j we have that $\|\{i : s_{i,j} \neq \infty\}\| \leq 4$), if the deadline is constant, and if the minimal salaries of the agents are equal 1.*

Proof. The proof is by reduction from the exact set cover problem. We are given a set of elements $T = \{t_1, t_2, \ldots, t_q\}$ and family $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of 3-element subsets of T. We assume that each member of T appears in at most 3 sets from $\mathcal{S}$.

We build an instance I of the feasible coalition problem in the following way. There are $q + n$ tasks and $2n$ agents. The first q tasks $t_1, t_2, \ldots, t_q$ correspond to the elements in T. The next n tasks $t_{q+1}, t_{q+2}, \ldots t_{q+n}$ are the dummy tasks needed by our construction. The first n agents $1, 2, \ldots, n$ correspond to the subsets from $\mathcal{S}$ and the next n agents $(n+1), (n+2), \ldots, 2n$ are the dummy agents. The minimal salaries of all agents are equal to 1.

For each agent i, $i \leq n$ and each task t_j, $j \leq q$, we set $s_{i,j} = 2$ if and only if $t_j \in S_i$; otherwise $s_{i,j} = \infty$. Also, for each agent i, $i \leq n$ and each task t_j, $j > q$ we set $s_{i,j} = 5$ if and only if $i = j - q$; otherwise $s_{i,j} = \infty$. For each agent i, $i > n$ and each task t_j we set $s_{i,j} = 6$ if and only if $i - n = j - q$; otherwise $s_{i,j} = \infty$. The deadline d is equal to 6 and the budget v is equal to $v = \frac{7}{3}q + 5n$. Clearly, each agent has no more than 4 skills and so, in any feasible solution, cannot be assigned more than 4 tasks.

We will show that the answer to the original instance of the exact set cover problem is "yes" if and only if there exists a feasible coalition in the our constructed instance I.

$\Longleftarrow$ Let us assume there exists a feasible coalition $\mathcal{C}$. The cost of this coalition is at most equal to $v = \frac{7}{3}q + 5n$. Each non-dummy task (there are q such tasks) takes 2 time units, and thus implies the cost equal to 2. The dummy tasks can be assigned either to non-dummy agents (implying the cost 5) or to dummy agents (implying the cost 6). Thus, we infer that at most $\frac{q}{3}$ dummy agents are assigned a task $(2q + \frac{1}{3}q \cdot 6 + (n - \frac{1}{3}q) \cdot 5 = v)$. As the result at least $(n - \frac{q}{3})$ dummy tasks must be assigned to non-dummy agents. A non-dummy agent, who is assigned a dummy task cannot be assigned any other task (otherwise the completion time would exceed the deadline). Thus, at most $\frac{q}{3}$ non-dummy agents can be assigned non-dummy tasks. The non-dummy tasks can be assigned only to non-dummy agents. We see the subsets corresponding to these non-dummy agents who are assigned non-dummy tasks form the solution to the initial exact set cover problem.

$\Longrightarrow$ Let us assume that there exists the exact set cover in the initial problem. The agents corresponding to the subsets from the cover can be assigned tasks so that the deadline is not exceeded and the total cost of completing these tasks is equal to $2q$. The other $(n - \frac{q}{3})$ non-dummy agents can be assigned one dummy task each. Finally, not-yet assigned dummy tasks can be assigned to dummy agents. The total cost of such assignment is equal to $2q + (n - \frac{1}{3}q) \cdot 5 + \frac{1}{3}q \cdot 6 = v$.

This completes the proof. □

Unfortunately, FFCPS is not approximable for makespan, for budget, and even for the combination of both these parameters.

Theorem 7.26. *For any $\alpha, \beta \geq 1$ there is no polynomial α-β-approximation algorithm for* FFCPS *that approximates makespan with the ratio α and budget with the ratio β, unless P=NP. This result holds even if the costs of all processors are equal 1.*

Proof. For the sake of contradiction let us assume that there exists α-β-approximation algorithm A. We provide a reduction showing that A can be used as β-approximation algorithm for SetCover. This will however contradict the result of Feige [98]. Let I be an instance of SetCover, where $T = \{t_1, t_2, \ldots, t_q\}$ is the set of elements and $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ is the set of the subsets of T. We ask whether there exists K subsets from $\mathcal{S}$ that together cover all elements from T.

From I we construct an instance of FFCPS in the following way. There are q tasks corresponding to q elements in I. There are n agents $1, 2, \ldots, n$ corresponding to the subsets in $\mathcal{S}$. The duration $s_{i,j}$ of the task t_i when processed by the agent j is defined in the following way. If $t_i \in S_j$ then $s_{i,j} = 1$. Otherwise, $s_{i,j} = \alpha q + 1$. The minimal salary of each agent is equal to 1 and the total budget is K. We show that if there exist K subsets from $\mathcal{S}$ covering T then we can use A to find βK subsets covering T.

Let C denote the covering using K subsets. If we assign each task t_i to any agent j such that $S_j \in C$ and $t_i \in S_j$, then the completion time of the tasks on each processor will be at most equal to q. In such case we will use only K processors. Thus A returns the solution with the makespan at most equal to αq using at most βK processors. This, however, means that each task t_i is assigned to such agent j that $t_i \in S_j$. Thus, the subsets corresponding to the selected processors form the solution of I. Of course, there is at most βK such processors. This completes the proof. □

Theorems 7.23, 7.24, and 7.25 show that the problems FFCPS and FFCHS remain NP-hard even if various parameters are constant. Although Theorem 7.23 gives us NP-hardness even for 2 agents, it is somehow not satisfactory as we used the fact that the deadline d can be very large. If the deadline is given in unary encoding, we can solve the case for 2 agents by dynamic programming. Thus, it is interesting if we can solve the problem efficiently for small numbers of agents, if the input is given in unary encoding. We use parameterized complexity theory [87] to approach this problem. We ask if FFCPS and FFCHS have FPT algorithms for the parameter n, the number of the agents, provided the input is given in unary encoding.

Theorem 7.27. *Consider the number of agents as the parameter.* FFCPS *and* FFCHS *are W[1]-hard, even if all the agents have minimal salaries equal to 1, and if the size of the input is given in unary encoding.*

Proof. We show the reduction from Unary Bin Packing (which is W[1]-hard [147]). In the instance of the unary bin packing problem we are given a set T of q items $T = \{t_1, t_2, \dots, t_q\}$ (the size of the item t_i is equal to s_i) and a set N of n bins, each having a capacity d. We ask whether it is possible to pack all the items to the bins.

From this instance we can construct the instance of FFCPS (or FFCHS) in the following way. Here T will be the set of tasks, N will be the set of agents. The minimal salaries of the agents are equal to 1; the speed of processing the task t_j by the agent i is equal to $s_{i,j} = s_j$. In FFCPS we set the total budget v to be equal to n. In FFCHS we set v to $\sum_{t_i \in T} s_i$. Of course, there exists a feasible schedule if and only if there exists a feasible bin-packing. □

7.6.3 Integer Programming Formulation

In the hourly salary model, Shmoys and Tardos [270] show an integer programming formulation. In this subsection we state the FFCPS problem as an integer programming problem.

$$\begin{aligned}
&\text{minimize } d && (7.8)\\
&\text{subject to } \sum_{i \in N} a_i \phi_i^{\min} \le v && (7.9)\\
&x_{i,j} \le a_i \qquad , i \in N && (7.10)\\
&\sum_{t_j \in T} x_{i,j} s_{i,j} \le d \text{ , } i \in N \le d && (7.11)\\
&x_{i,j} \in \{0,1\} \qquad , i \in N; t_j \in T && (7.12)\\
&a_i \in \{0,1\} \qquad , i \in N && (7.13)
\end{aligned}$$

In the above formulation, a binary variable a_i denotes whether agent i is a part of the solution (is assigned some tasks, Equation 7.13). A binary variable $x_{i,j}$ is equal to 1 if and only if the task t_j is assigned to the agent i (Equation 7.12). We minimize the makespan d (Equation 7.8), which is the maximal completion time of the tasks over all the agents (Inequality 7.11). We cannot exceed the budget v (Inequality 7.9), and the tasks can be assigned only to the selected agents (Inequality 7.10).

7.7 Conclusions

In this chapter we presented a new class of coalitional games that model cooperation and competition for the employment in a complex project. We believe that this is an interesting setting that relates to other natural problems, such as coalition formation, coalitional auctions, auctions for sharable items, etc. We consider two models of the organization of the market. First, the winning coalition is selected by the client based on bids from individual agents; the agents are strategic about the salaries they request. Second, the coalition formation process is decentralized—the already-formed coalitions bid for the project, thus the agents are strategic both regarding their salaries and regarding their cooperation partners.

We propose concepts of stability for each of our models. These concepts are of interest both to the agents and to the client. The client gains an insight into agents' strategies and can thus establish a relation between the cost of organizing the market and the cost of the winning coalition. The agents can optimize their strategies according to their beliefs about other agents (an agent can ask, e.g., whether she can increase her asking salary and still participate in the winning coalition). In the centralized model we show that the Strong Nash Equilibrium always exists. In the decentralized model an SNE may not exist, but we prove the existence of weakly winning coalitions. We show how to reduce the problem of finding a winning coalition to the problem of finding a feasible one. Our results are summarized in Table 7.2. Finally, to show that the abstract model can be applied in practice, we presented a concrete model in which the project is represented as a set of independent tasks and the agents have certain skills (expressed as processing speeds).

There are many natural open questions. The first interesting direction is to apply other solution concepts to the decentralized variant of our model (see Section 7.4.3): the Coalitional Farsighted Conservative Stable Set and the concepts inspired by the graph interpretations. Both the computational complexity and their game-theoretic properties are open. Second, it is natural to consider other types of auctions in the decentralized setting. In particular, it is an open question whether there exists a truthful mechanism. This question is especially appealing in the light of the vast literature on designing truthful mechanisms for the centralized setting. Third, it is interesting to analyze how incomplete knowledge of the agents and the inconsistencies of their beliefs affect the equilibria. How should an agent play when she is aware of these inconsistencies?

Chapter 8

Non-Monetary Fair Scheduling—A Cooperative Game Theory Approach

We consider a multi-organizational system in which each organization contributes processors to the global pool but also jobs to be processed on the common resources. The fairness of the scheduling algorithm is essential for the stability and even for the existence of such systems (as organizations may refuse to join an unfair system).

We consider on-line, non-clairvoyant scheduling of sequential jobs. The started jobs cannot be stopped, canceled, preempted, or moved to other processors. We consider identical processors, but most of our results can be extended to related or unrelated processors.

We model the fair scheduling problem as a cooperative game and we use the Shapley value to determine the ideal fair schedule. In contrast to the previous works, we do not use money to assess the relative utilities of the jobs. Instead, to calculate the contribution of an organization, we determine how the presence of this organization influences the performance of other organizations. Our approach can be used with arbitrary utility function (e.g., flow time, tardiness, resource utilization), but we argue that the utility function should be strategy resilient. The organizations should be discouraged from splitting, merging or delaying their jobs. We present the unique (to within a multiplicative and additive constants) strategy resilient utility function.

We show that the problem of fair scheduling is NP-hard and hard to approximate. However, we show that the problem parameterized with the number of organizations is fixed parameter tractable (FPT). Also, for unit-size jobs, we present a fully polynomial-time randomized approximation scheme (FPRAS). Although for the large number of the organizations the problem is computationally hard, the presented exponential algorithm can be used as a fairness benchmark.

All our algorithms are greedy, i.e., they don't leave free processors if there are waiting jobs. We show that any greedy algorithm results in at most 25% loss of the resource utilization in comparison with the globally optimal algorithm. As a corollary we conclude that the resource underutilization, being the result of the fairness requirement, is (tightly) upper bounded by 25%.

We propose a heuristic scheduling algorithm for the fair scheduling problem. We experimentally evaluate the heuristic and compare its fairness to fair share, round robin and the exact exponential algorithm. Our results show that fairness of the heuristic algorithm is close to the optimal one. The difference between our heuristic and the fair share algorithm is more visible on longer traces with more organizations. These results show that assigning static target shares (as in the fair share algorithm) is not fair in multi-organizational systems and that, instead, dynamic measures of organizations' contributions should be used.

8.1 Introduction

In multi-organizational systems, participating organizations give access to their local resources; in return their loads can be processed on other resources. The examples of such systems include PlanetLab[1], grids (Grid5000[2], EGEE[3]) or organizationally distributed storage systems [130]. There are many incentives for federating into consortia: the possibility of decreasing the costs of management and maintenance (one large system can be managed more efficiently than several smaller ones), but also the willingness to utilize resources more efficiently. Peak loads can be offloaded to remote resources. Moreover, organizations can access specialized resources or the whole platform (which permits, for example, testing on a large scale).

In the multi-organizational and multi-user systems, fairness of the resource allocation mechanisms is equally important as its efficiency. For example, efficiency of BitTorrent depends on users' collaboration, which in turn requires the available download bandwidth to be distributed fairly [250]. Fairness has been also discussed in storage systems [51,124,125,139,300,301,311] and computer networks [303]. In scheduling, for instance, a significant part of the description of Maui [144], perhaps the most common cluster scheduler, focuses on the fair-share mechanism. Nevertheless, there is no universal agreement on the meaning of fairness; next, we review approaches most commonly used in literature: distributive fairness and game theory.

Under distributive fairness, organizations are ensured a fraction of the resources according to *predefined (given) shares*. The share of an organization may depend on the perceived importance of the workload, payments [51,124,125,300]; or calculated to satisfy (predefined) service level agreements [139,158,311]. The literature on distributive fairness describes algorithms distributing resources according to the given shares, but does not describe how the shares should be set. In scheduling,

[1]www.planet-lab.org/
[2]www.grid5000.fr
[3]egee-technical.web.cern.ch

distributive fairness is implemented through fair queuing mechanisms: YFQ [35], SFQ and FSFQ [115,150], or their modifications [51,124,125,139,300,301,311,315].

A different approach is to optimize directly the performance (the utility) of users, rather than just the allocated resources. Kostreva et al. [167] propose an axiomatic characterization of fairness based on multi-objective optimization; [263] applies this concept to scheduling in a multi-organizational system. Inoie et al. [141] proposes a similar approach for load balancing: a fair solution must be Pareto-optimal and the revenues of the players must be proportional to the revenues in Nash equilibrium.

While distributive fairness might be justified in case of centrally-managed systems (e.g., Amazon EC2 or a single HPC center), in our opinion it is inappropriate for consortia (e.g., PlanetLab or non-commercial scientific systems such as Grid5000 or EGEE) in which there is no single "owner" and the participating organizations may take actions (e.g., reschedule jobs on their resources, add local resources, or isolate into subsystems). In such systems, the shares of the participating organizations should depend both on their workload and on the owned resources; intuitively an organization that contributes many "useful" machines should be favored; similarly an organization that has only a few jobs.

If agents may form binding agreements, cooperative game theory studies the stability of resulting agreements (coalitions and revenues). Shapley value [265], the established solution concept that characterizes what is a *fair* distribution of the total revenue of a coalition between the participating agents, has been used in scheduling theory. However, all the models we are aware of use the concept of money. The works of Carroll et at. [45], Mishra et al. [209], Mashayekhy and Grosu [197] and Moulin et al. [214] describe algorithms and the process of forming the coalitions for scheduling. These works assume that each job has a certain *monetary* value for the issuing organization and each organization has its initial monetary budget.

Money may have negative consequences on the stakeholders of resource-sharing consortia. Using (or even mentioning) money discourages people from cooperating [296]. This stays in sharp contrast with the idea behind the academic systems—sharing the infrastructure is a step towards closer cooperation. Additionally, we believe that using money is inconvenient in non-academic systems as well. In many contexts, it is not clear how to valuate the completion of the job or the usage of a resource (especially when workload changes dynamically). We think that the accurate valuation is equally important (and perhaps equally difficult) as the initial problem of fair scheduling. Although auctions [38] or commodity markets [160] have been proposed to set prices, these approaches implicitly require to set the reference value to determine profitability. Other works on *monetary* game-theoretical models for scheduling include [110,111,120,123,240]; monetary approach is also used for other resource allocation problems, e.g. network bandwidth allocation [310]. However, none of these works describes how to valuate jobs and resources.

In a non-monetary approach proposed by Dutot el al. [90], the jobs are scheduled to minimize the global performance metric (the makespan) with an additional requirement—the utility of each player cannot be worse than if the player would act alone. Such approach ensures the stability of the system against actions of any single user (it is not profitable for the user to leave the system and to act alone) but not to the formation of sub-coalitions.

In the selfish job model [224], the agents are the jobs that selfishly choose processors on which to execute. Similarly to our model, the resources are shared and treated as common goods; however, no agent contributes resources.

An alternative to scheduling is to allow jobs to share resources concurrently. In congestion games [56,225,258] the utility of the player using a resource R depends on the number of the players concurrently using R; the players are acting selfishly. Congestion games for divisible load scheduling were analyzed by Grosu and Chronopoulos [119] (see also Chapter 9 of this dissertation).

In this chapter we propose fair scheduling algorithms for systems composed of multiple organizations (in contrast to the case of multiple organizations using a system owned by a single entity). We model the organizations, their machines, and their jobs as a cooperative game. In this game we do not use the concept of money. When measuring the contribution of organization O, we analyze how the presence of O in the grand coalition influences the completion times of the jobs of all participating organization. This contribution is expressed in the same units as the utility of the organization. In the design of the fair algorithm we use the concept of Shapley value. In contrast to simple cooperative games, in our case the value of a coalition (the total utility of the organizations in this coalition) depends on the underlying scheduling algorithm. This makes the problem of calculating the contributions of the organizations more involved. First, we develop algorithms for arbitrary utilities (e.g., resource utilization, tardiness, flow time, etc.). Next, we argue that designing the scheduling mechanism itself is not enough; we show that the utility function must be chosen to discourage organizations from manipulating their workloads (e.g., merging or splitting the jobs—similar ideas have been proposed for the money-based models [214]). We present an exponential scheduling algorithm for the strategy resilient utility function. We show that the fair scheduling problem is NP-hard and difficult to approximate. For a simpler case, when all the jobs are unit-size, we present a fully polynomial-time randomized approximation scheme (FPRAS). According to our experiments, this algorithm is close to the optimum when used as a heuristics for workloads with different sizes of the jobs.

Our contribution is the following:

1. We derive the definition of a fair algorithm from the cooperative game theory axioms (Definitions 8.1 and 8.2, Algorithm 8.1, and Theorem 8.3). The algorithm uses only the notions regarding the performance of the system (no money-based mechanisms).

2. We present the axioms for (Section 8.4), and the definition of, the fair utility function (Theorem 8.4)—this function is similar to the flow-time metric but the differences make it strategy-resilient (Proposition 8.5).

3. We show that the fair scheduling problem is NP-complete (Theorem 8.6) and hard to approximate (Theorem 8.7). However, the problem parameterized with the number of organizations is fixed parameter tractable (FPT).

4. We present an FPRAS for a special case with unit-size jobs (Algorithm 8.6, Theorems 8.10 and 8.11).

5. We show the tight bounds on the resource underutilization due to the fairness of the algorithm. Our result is even more general and applies to all greedy algorithms (i.e., such algorithms that at every time moment in which there is a free processor and a non-empty set of ready, but not scheduled, jobs, schedules some job on some free processor).

6. We propose a practical heuristic that schedules jobs according to an estimated Shapley value. The heuristic estimates the contribution of an organization by the number of CPU-timeunits an organization contributes for computing jobs of other organizations; the algorithm schedules the jobs to minimize the maximal difference between the utility and the contribution over all organizations.

7. Finally, we conduct simulation experiments to verify fairness of commonly-used scheduling algorithms (Section 8.7). The experiments show that although the fair share algorithm is considerably better than round robin (which does not aim to optimize fairness), our heuristic constantly outperforms fair share, being close to the optimal algorithm and the randomized approximation algorithm. The main conclusion from the experimental part of this chapter is that ensuring that each party is given a fair share of resources (the distributive fairness approach) might not be sufficient in systems with dynamic job arrival patterns. An algorithm based on the Shapley value, that explicitly considers the organization's impact on other organizations' utilities, produces more fair schedules.

In this chapter we use very mild assumptions about the jobs. We do not require to know their valuations, durations, or their future incoming pattern. Thus, we believe the presented results have practical consequences for real-life job schedulers. Also, our exponential algorithm forms a benchmark for comparing the fairness of other polynomial-time scheduling algorithms. The experimental comparison of some real scheduling algorithms suggests that the polynomial-time heuristic algorithms inspired by the ones presented in this chapter often result in a better fairness than the currently most popular fair share algorithm.

8.2 Preliminaries

Organizations, machines, jobs. We consider a system built by a set of independent *organizations* $\mathcal{O} = \{O^{(1)}, O^{(2)}, \dots O^{(k)}\}$. Each organization $O^{(u)}$ owns a *computational cluster* consisting of $m^{(u)}$ *machines* (*processors*) denoted as $M_1^{(u)}, M_2^{(u)}, \dots M_{m^{(u)}}^{(u)}$ and produces its *jobs*, denoted as $J_1^{(u)}, J_2^{(u)}, \dots$. Each job $J_i^{(u)}$ has *release time* $r_i^{(u)} \in \mathbb{T}$, where $\mathbb{T}$ is a discrete set of time moments. We consider an on-line problem in which each job is unknown until its release time. We consider a non-clairvoyant model, i.e., the job's processing time is unknown until the job completes (hence we do not need to use imprecise [172] run-time estimates). For the sake of simplicity of the presentation, we assume that machines are identical, i.e., each job $J_i^{(u)}$ can be executed at any machine and its processing always takes $p_i^{(u)}$ time units; $p_i^{(u)}$ is the *processing time*. Most of the results, however, can be extended to the case of related machines, where $p_i^{(u)}$ is a function of the schedule—the only exception are the results in Section 8.5.1, where we rely on the assumption that each job processed on each machine takes exactly one time unit. The results even generalize to the case of unrelated machines, however if we assume non-clairvoyant model with unrelated machines (i.e., we do not know the processing times of the jobs on any machine) then we cannot optimize the assignment of jobs to machines.

The jobs are sequential (this is a standard assumption in many scheduling models and, particularly, in the selfish job model [224]; an alternative is to consider the parallel jobs, which we plan to do in the future). Once a job is started, the scheduler cannot preempt it or migrate it to another machine (this assumption is usual in HPC scheduling because of high migration costs). Finally, we assume that the jobs of each individual organization should be started in the order in which they are presented. This allows organizations to have an internal prioritization of their jobs.

Cooperation, schedules. Organizations can cooperate and share their infrastructure; in such case we say that organizations form a *coalition*. Formally, a coalition $\mathcal{C}$ is a subset of the set of all organizations, $\mathcal{C} \subseteq \mathcal{O}$. We also consider a specific coalition consisting of all organizations, which we call a *grand coalition* and denote as $\mathcal{C}_g$ (formally, $\mathcal{C}_g = \mathcal{O}$, but in some contexts we use the notation $\mathcal{C}_g$ to emphasize that we are referring to the set of the organizations that cooperate). The coalition must agree on the schedule $\sigma = \bigcup_{(u)} \bigcup_i \{(J_i^{(u)}, s_i^{(u)}, M(J_i^{(u)}))\}$ which is a set of triples; a triple $(J_i^{(u)}, s_i^{(u)}, M(J_i^{(u)}))$ denotes a job $J_i^{(u)}$ started at time moment $s_i^{(u)} \geq r_i^{(u)}$ on machine $M(J_i^{(u)})$. We assume that a machine executes at most one job at any time moment. We often identify a job $J_i^{(u)}$ with a pair $(s_i^{(u)}, p_i^{(u)})$; and a schedule with $\bigcup_{(u)} \bigcup_i \{(s_i^{(u)}, p_i^{(u)}))\}$ (we do so for a more compact presentation of our results). Each coalition uses all the machines of its participants and schedules consecutive tasks on available machines. We consider only greedy schedules: at any time moment if there is a free processor and a non-empty set of ready, but not scheduled, jobs, some job must be assigned to the free processor. Since we do not know neither the characteristics of the future workload nor the duration of the started but not yet completed jobs, any non-greedy policy would result in unnecessary delays in processing jobs. Also, such greedy policies are used in real-world schedulers [144].

Let $\mathfrak{J}$ denote the set of all possible sets of jobs. An *online scheduling algorithm* (in short, a scheduling algorithm) $\mathcal{A} : \mathfrak{J} \times \mathbb{T} \to \mathcal{O}$ is an online algorithm that continuously builds a schedule: for a given time moment $t \in \mathbb{T}$ such that there is a free machine in t and a set of jobs released before t but not yet scheduled: $\mathcal{J} \in \mathfrak{J}$, $\mathcal{A}(\mathcal{J}, t)$ returns the organization the task of which should be started. The set of all possible schedules produced by such algorithms is the set of *feasible schedules* and denoted by Γ. We recall that in each feasible schedule the tasks of a single organization are started in the FIFO order.

Objectives. We consider a *utility function* $\psi : \Gamma \times \mathcal{O} \times \mathbb{T} \to \mathbb{R}$ that for a given schedule $\sigma \in \Gamma$, an organization $O^{(u)}$, and a time moment t gives the value corresponding to the $O^{(u)}$ organization's satisfaction from a schedule σ until time moment t. The examples of such utility functions that are common in scheduling theory are: flow time, resource utilization, turnaround, etc. Our scheduling algorithms will only use the notions of the utilities and do not require any external payments.

Since a schedule σ is fully determined by a scheduling algorithm $\mathcal{A}$ and a coalition of organizations $\mathcal{C}$, we often identify $\psi(\mathcal{A}, \mathcal{C}, O^{(u)}, t)$ with appropriate $\psi(\sigma, O^{(u)}, t)$. Also, we use a shorter notation $\psi^{(u)}(\mathcal{C})$ instead of $\psi(\mathcal{A}, \mathcal{C}, O^{(u)}, t)$ whenever $\mathcal{A}$ and t are known from the context. We define the *characteristic function* $v : \Gamma \times \mathbb{T} \to \mathbb{R}$ describing the total utility of the organizations from a schedule: $v(\mathcal{A}, \mathcal{C}, t) = \sum_{O^{(u)} \in \mathcal{C}} \psi(\mathcal{A}, \mathcal{C}, O^{(u)}, t)$. Analogously as above, we can use an equivalent formulation: $v(\sigma, t) = \sum_{O^{(u)} \in \mathcal{C}} \psi(\sigma, O^{(u)}, t)$, also using a shorter notation $v(\mathcal{C})$ whenever it is possible. Note that the utilities of the organizations, $\psi^{(u)}(\mathcal{C})$, constitute a division of the total value of the coalition $v(\mathcal{C})$.

8.3 Fair Scheduling Based on the Shapley Value

In this section our goal is to find a scheduling algorithm $\mathcal{A}$ that in each time moment t ensures a fair distribution of the value of the coalition $v(\mathcal{C})$ between the participating organizations. We will denote this desired fair division of the value v as $\phi^{(1)}(v), \phi^{(2)}(v), \ldots, \phi^{(k)}(v)$ meaning that $\phi^{(u)}(v)$ denotes the ideally fair revenue (utility) obtained by organization $O^{(u)}$. We would like the values $\phi^{(u)}(v)$ to satisfy the fairness properties, first proposed by Shapley [265] (below we give intuitive motivations; see [265] for further arguments).

1. Efficiency—the total value $v(\mathcal{C})$ is distributed:

$$\sum_{O^{(u)} \in \mathcal{C}} \phi^{(u)}(v(\mathcal{C})) = v(\mathcal{C}).$$

2. Symmetry—the organizations $O^{(u)}$ and $O^{(u')}$ having indistinguishable contributions obtain the same profits:

$$\left(\forall_{\mathcal{C}' \subset \mathcal{C}: O^{(u)}, O^{(u')} \notin \mathcal{C}'} \; v(\mathcal{C}' \cup \{O^{(u)}\}) = v(\mathcal{C}' \cup \{O^{(u')}\})\right) \Rightarrow \phi^{(u)}(v(\mathcal{C})) = \phi^{(u')}(v(\mathcal{C})).$$

3. Additivity—for any two characteristic functions v and w and a function $(v+w)$: $\forall_{\mathcal{C}' \subseteq \mathcal{C}} \; (v+w)(\mathcal{C}') = v(\mathcal{C}') + w(\mathcal{C}')$ we have that $\forall_{\mathcal{C}' \subseteq \mathcal{C}} \forall_u$:

$$\phi^{(u)}((v+w)(\mathcal{C})) = \phi^{(u)}(v(\mathcal{C})) + \phi^{(u)}(w(\mathcal{C})).$$

   Consider any two independent schedules σ_1 and σ_2 that together form a schedule $\sigma_3 = \sigma_1 \cup \sigma_2$ (σ_1 and σ_2 are independent iff removing any subset of the jobs from σ_1 does not influence the completion time of any job in σ_2 and vice versa). The profit of an organization that participates only in one schedule (say σ_1) must be the same in case of σ_1 and σ_3 (intuitively, the jobs that do not influence the current schedule also do not influence the current profits). The profit of every organization that participates in both schedules should in σ_3 be the sum of the profits in σ_1 and σ_2. Intuitively: if the schedules are independent then the profits are independent too.

4. Null player—an organization that does not increase the value of any coalition $C' \subset C$ gets nothing:

$$\left(\forall_{\mathcal{C}' \subset \mathcal{C}} : v(\mathcal{C}' \cup \{O^{(u)}\}) = v(\mathcal{C}')\right) \Rightarrow \phi^{(u)}(v(\mathcal{C})) = 0.$$

Since the four properties are actually the axioms of the Shapley value [265], they fully determine the single mapping between the coalition values and the profits of organizations (known as the Shapley value). In game theory the Shapley value is considered the classic mechanism ensuring the fair division of the revenue of the coalition[4]. The Shapley value is defined and described in preliminaries of this dissertation in Section 2.1.2. Here we only recall one of the definitions, showing an algorithm for its computation.

Let $\mathcal{L}_{\mathcal{C}}$ denote all orderings of the organizations from coalition $\mathcal{C}$. Each ordering $\prec_{\mathcal{C}}$ can be associated with a permutation of the set $\mathcal{C}$, thus $\|\mathcal{L}_{\mathcal{C}}\| = \|\mathcal{C}\|!$. For the ordering $\prec_{\mathcal{C}} \in \mathcal{L}_C$ we define $\prec_{\mathcal{C}} (O^{(i)}) = \{O^{(j)} \in \mathcal{C} : O^{(j)} \prec_{\mathcal{C}} O^{(i)}\}$ as the set of all organizations from $\mathcal{C}$ that precede $O^{(i)}$ in the order $\prec_{\mathcal{C}}$. Shapley value can be computed by the following formula [231]:

$$\phi^{(u)}(v(\mathcal{C})) = \frac{1}{\|\mathcal{C}\|!} \sum_{\prec_{\mathcal{C}} \in \mathcal{L}_{\mathcal{C}}} \left(v\left(\prec_{\mathcal{C}} (O^{(u)}) \cup \{O^{(u)}\}\right) - v\left(\prec_{\mathcal{C}} (O^{(u)})\right)\right). \tag{8.1}$$

This formulation has an interesting interpretation. Consider the organizations joining the coalition $\mathcal{C}$ in order $\prec_{\mathcal{C}}$. Each organization $O^{(u)}$, when joining, contributes to

[4]The Shapley value has other interesting axiomatic characterizations [305].

the current coalition value equal to $\big(v(\prec_{\mathcal{C}} (O^{(u)}) \cup \{O^{(u)}\}) - v(\prec_{\mathcal{C}} (O^{(u)})\big)$. Thus, $\phi^{(u)}(v(\mathcal{C}))$ is the expected contribution to the coalition $\mathcal{C}$ when the expectation is taken over the order in which the organizations join $\mathcal{C}$. Hereinafter we will call the value $\phi^{(u)}(v(\mathcal{C}))$ (or using a shorter notation $\phi^{(u)}$) as the *contribution* of the organization $O^{(u)}$.

Let us consider a specific scheduling algorithm $\mathcal{A}$, a specific time moment t, and a specific coalition $\mathcal{C}$. Ideally, the utilities of the organizations should be equal to the reference fair values, $\forall_u \; \psi^{(u)}(\mathcal{C}) = \phi^{(u)}(v(\mathcal{C}))$, (meaning that the utility of the organization is equal to its contribution), but our scheduling problem is discrete so an algorithm guaranteeing this property may not exist. Thus, we will call "fair" an algorithm that results in utilities close to the contributions. The following definition of a fair algorithm is in two ways recursive. A fair algorithm for a coalition $\mathcal{C}$ and time t must be also fair for all subcoalitions $\mathcal{C}' \subset \mathcal{C}$ and for all previous $t' < t$. An alternative to being fair for all previous $t' < t$ would be to ensure asymptotic fairness; however, our formulation is more responsive and more relevant for the online case. We want to avoid the case in which an organization is disfavored in one, possibly long, time period and then favored in the next one.)

Definition 8.1. *Set an arbitrary metric $\| \cdot \|_d : 2^k \times 2^k \to \mathbb{R}_{\geq 0}$; and set an arbitrary time moment $t \in \mathbb{T}$. $\mathcal{A}$ is a fair algorithm in t for coalition $\mathcal{C}$ in metric $\| \cdot \|_d$ if and only if:*

$$\mathcal{A} \in \operatorname{argmin}_{\mathcal{A}' \in \mathcal{F}(<t)} \|\vec{\phi}(\mathcal{A}', \mathcal{C}, t) - \vec{\psi}(v(\mathcal{A}', \mathcal{C}, t)\|_d$$

where:

1. *$\mathcal{F}(< t)$ is a set of algorithms fair in each time moment $t' < t$; $\mathcal{F}(< 0)$ is a set of all greedy algorithms,*
2. *$\vec{\psi}(v(\mathcal{A}', \mathcal{C})$ is a vector of utilities $\langle \psi^{(u)}(v(\mathcal{A}', \mathcal{C})) \rangle$,*
3. *$\vec{\phi}(\mathcal{A}', \mathcal{C})$ is a vector of contributions $\langle \phi^{(u)}(v(\mathcal{A}', \mathcal{C})) \rangle$, where $\phi^{(u)}(v(\mathcal{A}', \mathcal{C}))$ is given by Equation 8.1,*
4. *In Equation 8.1, for any $\mathcal{C}' \subset \mathcal{C}$, $v(\mathcal{C}')$ denotes $v(\mathcal{A}_f, \mathcal{C}')$, where $\mathcal{A}_f$ is any fair algorithm for coalition $\mathcal{C}'$.*

Definition 8.2. *$\mathcal{A}$ is a fair algorithm for coalition $\mathcal{C}$ if and only if it is fair in each time moment $t \in \mathbb{T}$.*

Further on, we consider algorithms fair in the Manhattan metric: $\|\vec{v_1}, \vec{v_2}\|_M = \sum_{i=1}^{k} |v_1[i] - v_2[i]|$. However, our analysis can be generalized to other distance functions.

Based on Definition 8.2, we construct a reference fair algorithm for an arbitrary utility function ψ (Algorithm Ref; the pseudo-code is presented in Figure 8.1).

```
Notation:
$\mathbf{jobs}[\mathcal{C}][O^{(u)}]$ — list of waiting jobs of organization $O^{(u)}$.
$\boldsymbol{\phi}[\mathcal{C}][O^{(u)}]$ — the contribution of $O^{(u)}$ in $\mathcal{C}$, $\phi^{(u)}(\mathcal{C})$.
$\boldsymbol{\psi}[\mathcal{C}][O^{(u)}]$ — utility of $O^{(u)}$ from being in $\mathcal{C}$, $\psi(\mathcal{C}, O^{(u)})$.
$\mathbf{v}[\mathcal{C}]$ — value of a coalition $\mathcal{C}$.
$\boldsymbol{\sigma}[\mathcal{C}]$ — schedule for a coalition $\mathcal{C}$.
FreeMachine($\sigma, t$) — returns true if and only if there is a free machine in $\sigma$ in time $t$.

ReleaseJob($O^{(u)}$, $J$):
    for $\mathcal{C} : O^{(u)} \in \mathcal{C}$ do
        $\text{jobs}[\mathcal{C}][O^{(u)}].\text{push}(J)$

Distance($\mathcal{C}$, $O^{(u)}$, $t$):
    $old \leftarrow \sigma[\mathcal{C}]$;
    $new \leftarrow \sigma[\mathcal{C}] \cup \{(\text{jobs}[\mathcal{C}][O^{(u)}].\text{first}, t)\}$;
    $\Delta\psi \leftarrow \psi(new, O^{(u)}, t) - \psi(old, O^{(u)}, t)$;
    return $\left|\phi[\mathcal{C}][O^{(u)}] + \frac{\Delta\psi}{\|\mathcal{C}\|} - \psi[\mathcal{C}][O^{(u)}] - \Delta\psi\right|$
    $+ \sum_{O^{(u')}} \left|\phi[\mathcal{C}][O^{(u')}] + \frac{\Delta\psi}{\|\mathcal{C}\|} - \psi[\mathcal{C}][O^{(u')}]\right|$;

SelectAndSchedule($\mathcal{C}$, $t$):
    $u \leftarrow \text{argmin}_{O^{(u)}}(\texttt{Distance}(\mathcal{C}, O^{(u)}, t))$ ;
    $\sigma[\mathcal{C}] \leftarrow \sigma[\mathcal{C}] \cup \{(\text{jobs}[\mathcal{C}][u].\text{first}, t)\}$;
    $\psi[\mathcal{C}][O^{(u)}] \leftarrow \psi(\sigma[\mathcal{C}], O^{(u)}, t)$;

UpdateVals($\mathcal{C}$, $t$):
    foreach $O^{(u)} \in \mathcal{C}$ do
        $\psi[\mathcal{C}][O^{(u)}] \leftarrow \psi(\sigma[\mathcal{C}], O^{(u)}, t)$;
        $\phi[\mathcal{C}][O^{(u)}] \leftarrow 0$;
    $\text{v}[\mathcal{C}] \leftarrow \sum_{O^{(u)}} \psi(\sigma[\mathcal{C}], O^{(u)}, t)$;
    foreach $\mathcal{C}_{sub}$: $\mathcal{C}_{sub} \subseteq \mathcal{C}$ do
        foreach $O^{(u)} \in \mathcal{C}_{sub}$ do
            $\phi[\mathcal{C}][O^{(u)}] \leftarrow \phi[\mathcal{C}][O^{(u)}] +$
                $(\text{v}[\mathcal{C}_{sub}] - \text{v}[\mathcal{C}_{sub} \setminus \{O^{(u)}\}])$
                $\cdot \frac{(\|\mathcal{C}_{sub}\|-1)!(\|\mathcal{C}\|-\|\mathcal{C}_{sub}\|)!}{\|\mathcal{C}\|!}$ ;

FairAlgorithm($\mathcal{C}$):
    foreach time moment $t$ do
        foreach job $J_i^{(u)}$: $r_i^{(u)} = t$ do
            ReleaseJob($O_i^{(u)}, J_i^{(u)}$);
        for $s \leftarrow 1$ to $\|C\|$ do
            foreach $\mathcal{C}' \subset \mathcal{C}$, such that $\|C'\| = s$ do
                UpdateVals($\mathcal{C}', t$);
                while FreeMachine($\sigma[\mathcal{C}']$, $t$) do
                    SelectAndSchedule($\mathcal{C}', t$);
                $\text{v}[\mathcal{C}] \leftarrow \sum_{O^{(u)}} \psi(\sigma[\mathcal{C}], O^{(u)}, t)$;
```

Figure 8.1: Algorithm REF: a fair algorithm for arbitrary utility function ψ.

Algorithm REF keeps a schedule for every subcoalition $\mathcal{C}' \subset \mathcal{C}$. For each time moment the algorithm complements the schedule starting from the subcoalitions of the smallest size. The values of all smaller coalitions $v[\mathcal{C}_s]$ are used to update the contributions of the organizations (lines 19-23) in the procedure `UpdateVals`). Before scheduling any job of each coalition $\mathcal{C}'$, the contribution and the utility of each organization in $\mathcal{C}'$ is updated (procedure `UpdateVals`). If there is a free machine and a set of jobs waiting for execution, the algorithm selects the job according to Definition 8.1, thus it selects the organization that minimizes the distance of the utilities $\vec{\psi}$ to their ideal values $\vec{\phi}$ (procedure `SelectAndSchedule`). Assuming the first job of the organization $O^{(u)}$ is tentatively scheduled, procedure `Distance` computes a distance between the new values of $\vec{\psi}$ and $\vec{\phi}$.

Procedure `Distance` works as follows. Assuming $O^{(u)}$ is selected, the value $\Delta\psi$ denotes the increase of the utility of $O^{(u)}$ due to scheduling its first waiting job. This is also the increase of the value of the whole coalition. When procedure `Distance`$(\mathcal{C}, O^{(u)}, t)$ is executed, the schedules (and, thus, the values) in time t for all subcoalitions $\mathcal{C}' \subset \mathcal{C}$ are known. The schedule, for coalition $\mathcal{C}$ is known only in time $(t-1)$, as we have not yet decided which job should be scheduled in time t. Thus, scheduling the job will change the schedule (and the value) only for a coalition $\mathcal{C}$. From the definition of the Shapley value, it follows that if the value $v(\mathcal{C})$ of the coalition $\mathcal{C}$ increases by $\Delta\psi$ and the value of all subcoalitions remains the same, then the contribution $\phi^{(u')}$ of each organization $O^{(u')} \in \mathcal{C}$ to $\mathcal{C}$ will increase by the same value equal to $\Delta\psi/\|\mathcal{C}\|$. Thus, for each organization $O^{(u')} \in \mathcal{C}$, the new contribution of $O^{(u')}$ is $(\phi[\mathcal{C}][O^{(u')}] + \frac{\Delta\psi}{\|\mathcal{C}\|})$. The new utility for each organization $O^{(u')} \in \mathcal{C}$, such that $O^{(u')} \neq O^{(u)}$ is equal to $\psi[\mathcal{C}][O^{(u')}]$. The new utility of the organization $O^{(u)}$ is equal to $(\psi[\mathcal{C}][O^{(u)}]| + \Delta\psi)$.

Theorem 8.1. *Algorithm* REF *from Figure 8.1 is a fair algorithm.*

Proof. Algorithm REF is a straightforward implementation of Definition 8.2. □

Proposition 8.2. *In each time moment t the time complexity of Algorithm* REF *from Figure 8.1 is $O(\|\mathcal{O}\|(2^{\|\mathcal{O}\|}\sum m^{(u)} + 3^{\|\mathcal{O}\|}))$.*

Proof. Once the contribution is calculated, each coalition in t may schedule at most $\sum m^{(u)}$ jobs. The time needed for selecting each such a job is proportional to the number of the organizations. Thus, we get the $\|\mathcal{O}\|2^{\|\mathcal{O}\|}\sum m^{(u)}$ part of the complexity. For calculating the contribution of the organization $O^{(u)}$ to the coalition $\mathcal{C}$ the algorithm considers all subsets of $\mathcal{C}$—there are $2^{\|\mathcal{C}\|}$ such subsets. Since there are $\binom{\|\mathcal{O}\|}{k}$ coalitions of size k, the number of the operations required for calculating the contributions of all organizations is proportional to:

$$\sum_{(u)}\sum_{k=0}^{\|\mathcal{O}\|}\binom{\|\mathcal{O}\|}{k}2^k = \|\mathcal{O}\|\sum_{k=0}^{\|\mathcal{O}\|}\binom{\|\mathcal{O}\|}{k}1^{\|\mathcal{O}\|-k}2^k = \|\mathcal{O}\|(1+2)^{\|\mathcal{O}\|} = \|\mathcal{O}\|3^{\|\mathcal{O}\|}.$$

This gives the $\|\mathcal{O}\|3^{\|\mathcal{O}\|}$ part of the complexity and completes the proof. □

Corollary 8.3. *The problem of finding fair schedule parameterized with the number of organizations is FPT.*

8.4 Strategy-Proof Utility Functions

There are many utility functions considered in scheduling, e.g., the flow time, the turnaround time, resource utilization, makespan, tardiness. However, it is not sufficient to design a fair algorithm for an arbitrary utility function ψ. Some functions may create incentives for organizations to manipulate their workload: to divide the tasks into smaller pieces, to merge, or to delay them. This is undesired as an organization should neither profit nor suffer from the way it presents its workload. An organization should present its jobs in the most convenient way; it should not focus on playing against other organizations. We show that in organizationally distributed systems, where we have to take into account such manipulations, the choice of the utility functions is restricted.

We introduce additional notation for this section: let us fix an arbitrary organization $O^{(u)}$ and let σ_t denote a schedule of the jobs of $O^{(u)}$ in time t. The jobs $J_i(s_i, p_i)$ of $O^{(u)}$ are characterized by their start times s_i and processing times p_i. We are considering envy-free utility functions that for a given organization $O^{(u)}$ depend only on the schedule of the jobs of $O^{(u)}$. This means that there is no external economical relation between the organizations (organization O^u cares about O^v only if the jobs of O^v influence the jobs of O^u—in contrast to looking directly at the utility of O^v). We also assume the non-clairvoyant model—the utility in time t depends only on the jobs, or the parts of the jobs, completed before or at t. Let us assume that our goal is to maximize the utility function.[5] We start from presenting the desired properties of the utility function ψ (when presenting the properties we use the shorter notation $\psi(\sigma_t)$ for $\psi(\sigma_t, t)$):

1. Tasks anonymity (starting times)—improving the completion time of a single task with a certain processing time p by one unit of time is for each task equally profitable – for $s, s' \leq t - 1$, we require:

$$\psi(\sigma_t \cup \{(s,p)\}) - \psi(\sigma_t \cup \{(s+1,p)\}) =$$
$$\psi(\sigma'_t \cup \{(s',p)\}) - \psi(\sigma'_t \cup \{(s'+1,p)\}) > 0.$$

2. Tasks anonymity (number of tasks)—in each schedule increasing the number of completed tasks is equally profitable—for $s \leq t - 1$, we require:

$$\psi(\sigma_t \cup \{(s,p)\}) - \psi(\sigma_t) = \psi(\sigma'_t \cup \{(s,p)\}) - \psi(\sigma'_t) > 0.$$

[5]We can easily transform the problem to the minimization form by taking the inverse of the standard maximization utility function

3. Strategy-resistance—no organization cannot profit from merging multiple smaller jobs into one larger job or from dividing a larger job into smaller pieces:

$$\psi(\sigma_t \cup \{(s, p_1)\}) + \psi(\sigma_t \cup \{(s + p_1, p_2)\}) = \psi(\sigma_t \cup \{(s, p_1 + p_2)\}).$$

   In spite of dividing and merging the jobs, each organization can delay the release time of their jobs and artificially increase the size of the jobs. Delaying the jobs is, however, never profitable for the organization (by property 1). Also, the strategy-resistance property discourages the organizations from increasing the sizes of their jobs (The utility coming from processing a larger job is always greater and, thus, the scheduling algorithm would think that the organization gained more that it actually did.).

To within a multiplicative and additive constants, there is only one utility function satisfying the aforementioned properties.

Theorem 8.4. *Let ψ be a utility function that satisfies the 3 properties: task anonymity (starting times), task anonymity (number of tasks), strategy-resistance. ψ is of the following form:*

$$\psi(\sigma, t) = \sum_{(s,p)\in\sigma_t} \min(p, t - s)(K_1 - K_2 \frac{s + \min(s + p - 1, t - 1)}{2}) + K_3,$$

where

1. $K_1 = \psi(\sigma \cup \{(0, 1)\}, t) - \psi(\sigma) > 0$
2. $K_2 = \psi(\sigma \cup \{(s, p)\}, t) - \psi(\sigma \cup \{(s + 1, p)\}, t) > 0$
3. $K_3 = \psi(\emptyset)$.

Proof.

$$\psi(\sigma, t) = \psi(\bigcup_{(s,p)\in\sigma}(s, p), t) = \psi(\bigcup_{(s,p)\in\sigma}(s, \min(p, t - s)), t)$$

(non-clairvoyance)

$$= \psi(\bigcup_{(s,p)\in\sigma} \bigcup_{i=s}^{\min(s+p-1,t-1)} (i, 1), t)$$

(strategy-resistance)

$$= \psi(\bigcup_{(s,p)\in\sigma} \bigcup_{i=s}^{\min(s+p-1,t-1)} (0, 1), t) - K_2 \sum_{(s,p)\in\sigma_t} \sum_{i=s}^{\min(s+p-1,t-1)} i$$

(starting times anonymity)

$$= \psi(\emptyset) + \sum_{(s,p)\in\sigma_t} \sum_{i=s}^{\min(s+p-1,t-1)} K_1$$

(number of tasks anonymity)

$$- K_2 \sum_{(s,p)\in\sigma_t} \min(p, t-s) \frac{s + \min(s+p-1, t-1)}{2}$$

(sum of the arithmetic progression)

$$= K_3 + \sum_{(s,p)\in\sigma_t} \min(p, t-s)(K_1 - K_2 \frac{s + \min(s+p-1, t-1)}{2})$$

□

We use the constants K_1, K_2, K_3 to simplify the form of the utility function and ensure that the utility is always positive. With $K_1 = 1$, $K_2 = t$ and $K_3 = 0$, we get the following strategy-proof utility function:

$$\psi_{sp}(\sigma, t) = \sum_{(s,p)\in\sigma: s\leq t} \min(p, t-s) \left(t - \frac{s + \min(s+p-1, t-1)}{2} \right). \tag{8.2}$$

Function ψ_{sp} can be interpreted as the task throughput. A task with processing time p_i can be identified with p_i unit-sized tasks starting in consecutive time moments. Intuitively, the function ψ_{sp} assigns to each such unit-sized task starting at time t_s a utility value equal to $(t - t_s)$; the higher the utility value, the earlier this unit-sized task completes. The utility of the schedule is the sum of the utilities over all such unit-sized tasks. Function ψ_{sp} is similar to the flow time except for two differences: (i) Flow time is a minimization objective, but increasing the number of completed jobs increases its value. E.g., scheduling no jobs results in zero (optimal) flow time, but, of course, an empty schedule cannot be considered optimal (breaking the second axiom); (ii) Flow time favors short tasks, which is an incentive for dividing tasks into smaller pieces (this breaks strategy-resistance axiom). The differences between the flow time and ψ_{sp} is also presented in the example in Figure 8.2. The similarity of ψ_{sp} to the flow time is quantified by Proposition 8.5 below.

Proposition 8.5. *Let $\mathcal{J}$ be a fixed set of jobs, each having the same processing time p and each completed before time moment t. Then, maximization of the ψ_{sp} utility is equivalent to minimization of the flow time of the jobs.*

Proof. Let σ denote an arbitrary schedule of $\mathcal{J}$. Since the flow time uses the release times of the jobs, we will identify the jobs with the triples (s, p, r) where s, p and r denote the start time, processing time and release time, respectively. Let $\psi_{ft}(\sigma)$ denote the total flow time of the jobs from $\mathcal{J}$ in schedule σ. We have:

$$\psi_{sp}(\sigma, t) = \sum_{(s,p,r)\in\sigma: s\leq t} \min(p, t-s) \left(t - \frac{s + \min(s+p-1, t-1)}{2} \right)$$

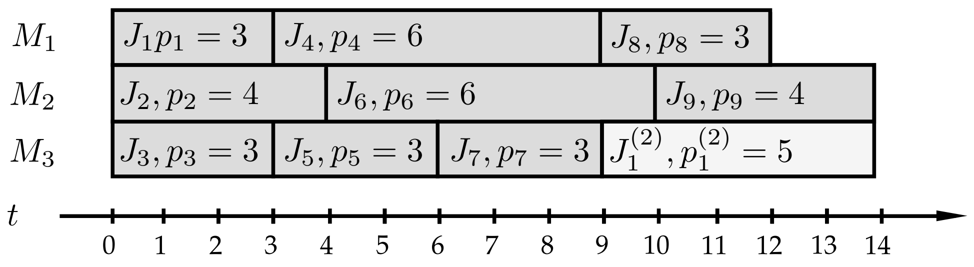

Figure 8.2: Consider 9 jobs owned by $O^{(1)}$ and a single job owned by $O^{(2)}$, all scheduled on 3 processors. We assume all jobs were released in time 0. In this example all jobs finish before or at time $t = 14$. The utility ψ_{sp} of the organization $O^{(1)}$ in time 13 does not take into account the last uncompleted unit of the job J_9, thus it is equal to: $3 \cdot (13 - \frac{0+2}{2}) + 4 \cdot (13 - \frac{0+3}{2}) + \cdots + 3 \cdot (13 - \frac{9+11}{2}) + 3 \cdot (13 - \frac{10+12}{2}) = 262$. The utility in time 14 takes into account all the parts of the jobs, thus it is equal to $3 \cdot (14 - \frac{0+2}{2}) + 4 \cdot (14 - \frac{0+3}{2}) + \cdots + 3 \cdot (14 - \frac{9+11}{2}) + 4 \cdot (14 - \frac{10+13}{2}) = 297$. The flow time in time 14 is equal to $3 + 4 + \cdots + 14 = 70$. If there was no job $J_1^{(2)}$, then J_9 would be started in time 9 instead of 10 and the utility ψ_{sp} in time 14 would increase by $4 \cdot (\frac{10+13}{2} - \frac{9+12}{2}) = 4$ (the flow time would decrease by 1). If, for instance, J_6 was started one time unit later, then the utility of the schedule would decrease by 6 (the flow time would decrease by 1), which shows that the utility takes into account the sizes of the jobs (in contrast to the flow time). If the job J_9 was not scheduled at all, the utility ψ_{sp} would decrease by 10, which shows that the schedule with more tasks has higher, i.e., better utility (the flow time would decrease by 14; since flow time is a minimization metric, this breaks the second axiom regarding the tasks anonymity).

$$= \sum_{(s,p,r)\in\sigma} p\left(t - \frac{2s+p-1}{2}\right)$$

(each job is completed before t)

$$= \sum_{(s,p,r)\in\sigma} (pt + \frac{p^2+p}{2} - r) - p \sum_{(s,p,r)\in\sigma} ((p+s) - r)$$

$$= \|\mathcal{J}\|(pt + \frac{p^2+p}{2}) - \sum_{(s,p,r)\in\sigma}(r) - p\psi_{ft}(\sigma)$$

Since p, $\|\mathcal{J}\|(pt + \frac{p^2+p}{2})$ and $\sum_{(s,p,r)\in\sigma} r$ are constants we get the thesis. $\square$

```
: SelectAndSchedule
    $u \leftarrow \mathrm{argmin}_{O^{(u)}}(\psi[\mathcal{C}][O^{(u)}] - \phi[\mathcal{C}][O^{(u)}])$ ;
    $\sigma[\mathcal{C}] \leftarrow \sigma[\mathcal{C}] \cup \{(\mathrm{jobs}[\mathcal{C}][u].\mathrm{first}, t)\}$;
    $\psi[\mathcal{C}][O^{(u)}] \leftarrow \psi(\sigma[\mathcal{C}], O^{(u)}, t)$;
```

Figure 8.3: Function `SelectAndSchedule` for utility function ψ_{sp}.

8.5 Fair Scheduling with Strategy-Proof Utility

For the concrete utility function ψ_{sp}, we can simplify the `SelectAndSchedule` function in Algorithm Ref. The simplified version is presented in Figure 8.3.

The algorithm selects the organization $O^{(u)}$ that has the largest difference $(\phi^{(u)} - \psi^{(u)})$, that is, the organization that has the largest contribution in comparison to the obtained utility. One can wonder whether we can select the organization in polynomial time—without keeping the $2^{\|\mathcal{C}\|}$ schedules for all subcoalitions. Unfortunately, the problem of calculating the credits for a given organization is NP-hard.

Theorem 8.6. *The problem of calculating the contribution $\phi^{(u)}(\mathcal{C}, t)$ for a given organization $O^{(u)}$ in coalition $\mathcal{C}$ in time t is NP-hard.*

Proof. We present the reduction of the SubsetSum problem (which is NP-hard) to the problem of calculating the contribution for an organization. Let I be an instance of the SubsetSum problem. In I we are given a set of k integers $S = \{x_1, x_2, \ldots, x_k\}$ and a value x. We ask whether there exists a subset of S with the sum of elements equal to x. From I we construct an instance I_{con} of the problem of calculating the contribution for a given organization. Intuitively, we construct the set of $(\|S\| + 2)$ organizations: $\|S\|$ of them will correspond to the appropriate elements from S. The two dummy organizations a and b are used for our reduction. One dummy organization a has no jobs. The second dummy organization b has a large job that dominates the value of the whole schedule. The instance I_{con} is constructed in such a way that for each coalition $\mathcal{C}$ such that $b \in \mathcal{C}$ and such that the elements of S corresponding to the organizations from $\mathcal{C}$ sum up to the value lower than x, the marginal contribution of a to $\mathcal{C}$ is $L + O(L)$, where $O(L)$ is small in comparison with L. The marginal contribution of a to other coalitions is small ($O(L)$). Thus, from the contribution of a, we can count the subsets of S with the sum of the elements lower than x. By repeating this procedure for $(x+1)$ we can count the subsets of S with the sum of the elements lower than $(x+1)$. By comparing the two values, we can find whether there exists the subset of S with the sum of the elements equal to x. The precise construction is described below.

Let $\mathcal{S}_{<x} = \{S' \subset S : \sum_{x_i \in S'} s_i < x\}$ be the set of the subsets of S, each having the sum of the elements lower than x. Let $n_{<x}(S) = \sum_{S' \in \mathcal{S}_{<x}} (\|S'\| + 1)!(\|S\| - \|S'\|)!$ be the number of the orderings (permutations) of the set $S \cup \{a, b\}$ that starts with some permutation of the sum of exactly one element of $\mathcal{S}_{<x}$ (which is some subset of

S such that the sum of the elements of this subset is lower than x) and $\{b\}$ followed by the element a. In other words, if we associate the elements from $S \cup \{a, b\}$ with the organizations and each ordering of the elements of $S \cup \{a, b\}$ with the order of the organizations joining the grand coalition, then $n_{<x}(S)$ is the number of the orderings corresponding to the cases when organization a joins grand coalition just after all the organizations from $S' \cup \{b\}$, where S' is some element of $\mathcal{S}_{<x}$. Of course $\mathcal{S}_{<x} \subseteq \mathcal{S}_{<(x+1)}$. Note that there exists $S' \subset S$, such that $\sum_{x_i \in S'} x_i = x$ if and only if the set $\mathcal{S}_{<x}$ is a proper subset of $\mathcal{S}_{<(x+1)}$ (i.e. $\mathcal{S}_{<x} \subset \mathcal{S}_{<(x+1)}$). Indeed, there exists S' such that $S' \notin \mathcal{S}_{<x}$ and $S' \in \mathcal{S}_{<(x+1)}$ if and only if $\sum_{x_i \in S'} x_i < x + 1$ and $\sum_{x_i \in S'} x_i \geq x$ from which it follows that $\sum_{x_i \in S'} x_i = x$. Also, $\mathcal{S}_{<x} \subset \mathcal{S}_{<(x+1)}$ if and only if $n_{<(x+1)}(S)$ is greater than $n_{<(x)}(S)$ (we are doing a summation of the positive values over the larger set).

In I_{con} there is a set of $(k + 2)$ machines, each owned by a different organization. We will denote the set of first k organizations as $\mathcal{O}_S$, the (k+1)-th organization as a and the (k+2)-th organization as b. Let $x_{tot} = \sum_{j=1}^{k} x_j + 2$. The i-th organization from $\mathcal{O}_S$ has 4 jobs: $J_1^{(i)}, J_2^{(i)}, J_3^{(i)}$ and $J_4^{(i)}$, with release times $r_1^{(i)} = r_1^{(i)} = 0$, $r_3^{(i)} = 3$ and $r_4^{(i)} = 4$; and processing times $p_1^{(i)} = p_2^{(i)} = 1$, $p_3^{(i)} = 2x_{tot}$ and $p_4^{(i)} = 2x_i$. The organization a has no jobs; the organization b has two jobs $J_1^{(b)}$ and $J_2^{(b)}$, with release times $r_1^{(b)} = 2$ and $r_2^{(b)} = (2x + 3)$; and processing times $p_1^{(b)} = (2x + 2)$ and $p_2^{(b)} = L = 4\|S\|x_{tot}^2((k + 2)!) + 1$ (intuitively L is a large number).

Until time $t = 2$ only the organizations from $\mathcal{O}_S$ have some (unit-size) jobs to be executed. The organization b has no jobs till time $t = 2$, so it will run one or two unit-size jobs of the other organizations, contributing to all such coalitions that include b and some other organizations from $\mathcal{O}_S$. This construction allows to enforce that in the first time moment after $t = 2$ when there are jobs of some of the organizations from $\mathcal{O}_S$ and of b available for execution, the job of b will be selected and scheduled first.

Let us consider a contribution of a to the coalition $\mathcal{C}$ such that $a \notin \mathcal{C}$ and $b \in \mathcal{C}$. There are $(\|\mathcal{C} \cap \mathcal{O}_S\| + 2)$ machines in the coalition $\mathcal{C} \cup \{a\}$. The schedule in $\mathcal{C} \cup \{a\}$ after $t = 2$ looks in the following way (this schedule is depicted in Figure 8.4). In time $t = 2$ one machine (let us denote this machine as M') starts the job $J_1^{(b)}$ In time $t = 3$ some $\|\mathcal{C} \cap \mathcal{O}_S\|$ machines start the third jobs (the one with size $2x_{tot}$) of the organizations from $\mathcal{C} \cap \mathcal{O}$ and one machine (denoted as M'') starts the fourth jobs of the organizations from $\mathcal{C} \cap \mathcal{O}_S$; the machine M'' completes processing all these jobs in time $2y + 4$, where $y = \sum_{i: O^{(i)} \in \mathcal{C} \wedge O^{(i)} \in \mathcal{O}_S} x_i$ (of course $2y + 4 \leq 2x_{tot}$). In time $(2x + 3)$, if $y < x$ the machine M'' starts processing the large job $J_2^{(b)}$ of the organization b; otherwise machine M'' in time $(2x+3)$ still executes some job $J_4^{(i)}$ (as the jobs $J_4^{(i)}$ processed on M'' start in even time moments). In time $2x + 4$, if $y \geq x$, the large job $J_2^{(b)}$ is started by machine M' just after the job $J_1^{(b)}$ is completed, ($J_1^{(b)}$ completes in $(2x + 4)$); here we use the fact that after $t = 2$, b will be prioritized over

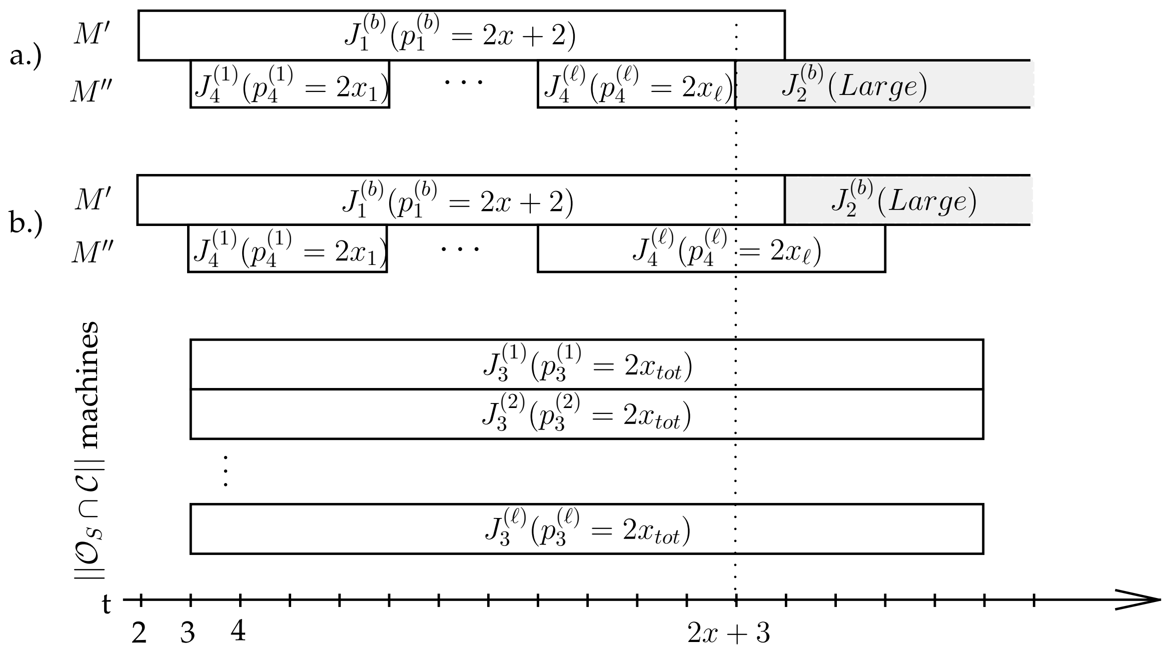
a.)
b.)
M'
M''
$J_1^{(b)}(p_1^{(b)} = 2x+2)$
$J_4^{(1)}(p_4^{(1)} = 2x_1)$
$\cdots$
$J_4^{(\ell)}(p_4^{(\ell)} = 2x_\ell)$
$J_2^{(b)}(Large)$
$\|\mathcal{O}_S \cap \mathcal{C}\|$ machines
$J_3^{(1)}(p_3^{(1)} = 2x_{tot})$
$J_3^{(2)}(p_3^{(2)} = 2x_{tot})$
$J_3^{(\ell)}(p_3^{(\ell)} = 2x_{tot})$
t
2
3
4
$2x+3$

Figure 8.4: The schedules for the coalition $\mathcal{C} \cup \{a\}$ for two cases: a) $\sum_{i:O^{(i)} \in \mathcal{C} \wedge O^{(i)} \in \mathcal{O}_S} x_i \leq x$, b) $\sum_{i:O^{(i)} \in \mathcal{C} \wedge O^{(i)} \in \mathcal{O}_S} x_i > x$. The two cases a) and b) differ only in the schedules on machines M' and M''. In the case a) the large job $J_2^{(b)}$ (marked as a light gray) is started one time unit earlier than in case b).

the organizations from $\mathcal{O}_S$. To sum up: if $y < x$ then the large job $J_2^{(b)}$ is started in time $(2x+3)$, otherwise it is started in time $(2x+4)$.

If $y < x$ then by considering only a decrease of the starting time of the largest job, the contribution of a to the coalition $\mathcal{C}$ can be lower bounded by c_1:

$$c_1 = L\left(t - \frac{(2x+3) + (2x+3+L)}{2}\right) - L\left(t - \frac{(2x+4) + (2x+4+L))}{2}\right) = L,$$

The organization a causes also a decrease of the starting times of the small jobs (the jobs of the organizations from $\mathcal{O}_S$); each job of size smaller or equal to $2x_{tot}$. The starting time of each such small job is decreased by at most $2x_{tot}$ time units. Thus, the contribution of a in case $y < x$ can be upper bounded by c_2:

$$c_2 \leq L + 4\|S\|x_{tot}^2.$$

If $y \geq x$ then a causes only a decrease of the starting times of the small jobs of the organizations from $\mathcal{O}_S$, so the contribution of a to $\mathcal{C}$ in this case can be upper bounded by c_3:

$$c_3 \leq 4\|S\|x_{tot}^2.$$

By similar reasoning we can see that the contribution of a to any coalition $\mathcal{C}'$ such that $b \notin \mathcal{C}'$ is also upper bounded by $4\|S\|x_{tot}^2$.

The contribution of organization a, $\phi^{(a)}$, is given by Equation 8.1, with $u = a$ and $C = \{O^{(1)} \dots O^{(k+2)}\}$. Thus:

$$\phi^{(a)} = \sum_{\mathcal{C}' \subseteq \mathcal{C} \setminus \{a\}} \frac{\|\mathcal{C}'\|!(k+1-\|\mathcal{C}'\|)!}{(k+2)!} \mathrm{marg\_}\phi(\mathcal{C}', a),$$

where $\mathrm{marg\_}\phi(\mathcal{C}', a)$ is the contribution of a to coalition $\mathcal{C}'$. All the coalitions $\mathcal{C}'$ such that $a \notin \mathcal{C}'$, $b \in \mathcal{C}'$ and $\sum_{i:O^{(i)} \in \mathcal{C}' \cap \mathcal{O}_S} x_i < x$ will contribute to $\phi^{(a)}$ the value at least equal to $\frac{n_{<x}(S)}{(k+2)!}c_1 = \frac{n_{<x}(S)L}{2(k+2)!}$ (as there is exactly $n_{<x}(S)$ orderings corresponding to the the case when a is joining such coalitions $\mathcal{C}'$) and at most equal to $\frac{n_{<x}(S)}{(k+2)!}c_2 \leq \frac{n_{<x}(S)(L+8\|S\|x_{tot}^2)}{2(k+2)!}$. The other $(k+2)! - n_{<x}(S)$ orderings will contribute to $\phi^{(a)}$ the value at most equal to $\frac{((k+2)!-n_{<x}(S))}{(k+2)!}c_3 = \frac{((k+2)!-n_{<x}(S))(4\|S\|x_{tot}^2)}{(k+2)!}$. Also:

$$\frac{((k+2)! - n_{<x}(S))(4\|S\|x_{tot}^2)}{(k+2)!} + \frac{n_{<x}(S)(4\|S\|x_{tot}^2)}{(k+2)!} = 4\|S\|x_{tot}^2 < \frac{L}{(k+2)!},$$

which means that $\phi^{(a)}$ can be stated as $\phi^{(a)} = \frac{n_{<x}(S)L}{(k+2)!} + R$, where $0 \leq R \leq \frac{L}{(k+2)!}$. We conclude that $\lfloor \frac{(k+2)!\phi^{(a)}}{L} \rfloor = n_{<x}(S)$. We have shown that calculating the value of $\phi^{(a)}$ allows us to find the value $n_{<x}(S)$. Analogously, we can find $n_{<(x+1)}(S)$. By comparing $n_{<x}(S)$ with $n_{<(x+1)}(S)$ we find the answer to the initial SubsetSum problem, which completes the proof.

□

We propose the following definition of the approximation of the fair schedule (similar definitions of the approximation ratio are used for multi-criteria optimization problems [95]):

Definition 8.3. *Let σ be a schedule and let $\vec{\psi}$ be a vector of the utilities of the organizations in σ. We say that σ is an α-approximation fair schedule in time t if and only if there exists a truly fair schedule σ^*, with the vector $\vec{\psi}^* = \langle \psi^{(u),*} \rangle$ of the utilities of the organizations, such that:*

$$\|\vec{\psi} - \vec{\psi}^*\|_M \leq \alpha \|\vec{\psi}^*\|_M = \alpha \sum_u \psi^{(u),*} = \alpha \cdot v(\sigma^*, \mathcal{C}).$$

Unfortunately, the problem of finding a fair schedule is difficult to approximate. There is no algorithm with approximation ratio better than 1/2 (see the proof below). This means that the problem is not approximable in practice (the ratio 1/2 is too low to be useful). Consider two schedules of jobs of m organizations on a single machine. Each organization has one job; all the jobs are identical. In the first schedule σ_{ord}

the jobs are scheduled in order: $J_1^{(1)}, J_1^{(2)}, \dots J_1^{(m)}$ and in the second schedule σ_{rev} the jobs are scheduled in exactly reverse order: $J_1^{(m)}, J_1^{(m-1)}, \dots J_1^{(1)}$. The relative distance between σ_{ord} and σ_{rev} tends to 1 (with increasing m), so $(\frac{1}{2})$-approximation algorithm does not allow to decide whether σ_{ord} is truly better than σ_{rev}. In other words, $\frac{1}{2}$-approximation algorithm cannot distinguish whether a given order of the priorities of the organizations is more fair then the reverse order.

Theorem 8.7. *For every $\epsilon > 0$, there is no polynomial algorithm for finding a $(\frac{1}{2} - \epsilon)$-approximate fair schedule, unless* $\mathrm{P} = \mathrm{NP}$.

Proof. Intuitively, we divide time in $(\|\mathcal{B}\|^2 + 3)$ independent batches. The jobs in the last batch are significantly larger than all the previous ones. We construct the jobs in all first $(\|\mathcal{B}\|^2 + 2)$ batches so that the order of execution of the jobs in the last batch depends on whether there exists a subset $S' \subset S$ such that $\sum_{x_i \in S'} x_i = x$. If the subset does not exist the organizations are prioritized in some predefined order σ_{ord}; otherwise, the order is reversed σ_{rev}. The sizes of the jobs in the last batch are so large that they dominate the values of the utilities of the organizations. The relative distance between the utilities in σ_{ord} and in σ_{rev} is $(1-\epsilon)$ so any $(\frac{1}{2}-\epsilon)$-approximation algorithm $\mathcal{A}$ would allow to infer the true fair schedule for such constructed instance, and so the answer to the initial SubsetSum problem. The precise construction is described below.

We show that if there is an $(\frac{1}{2} - \epsilon)$-approximation algorithm $\mathcal{A}$ for calculating the vector of the contributions, then we would be able to use $\mathcal{A}$ for solving the SubsetSum problem (which is NP-hard). This proof is similar in a spirit to the proof of Theorem 8.6. Let I be an instance of the SubsetSum problem, in which we are given a set $S = \{x_1, x_2, \dots, x_k\}$ of k integers and a value x. In the SubsetSum problem we ask for the existence of a subset $S' \subset S$ such that $\sum_{x_i \in S'} x_i = x$; we will call the subsets $S' \subset S$ such that $\sum_{x_i \in S'} x_i = x$ the x-sum subsets.

From I we construct the instance of the problem of calculating the vector of contributions in the following way. We set $\mathcal{O} = \mathcal{O}_S \cup \{a\} \cup \mathcal{B}$ to be the set of all organizations where $\mathcal{O}_S = \{O_1, \dots, O_k\}$ ($\|\mathcal{O}_S\| = k$) is the set of the organizations corresponding to the appropriate elements of S and $\{a\} \cup \mathcal{B}$, where $\mathcal{B} = \{B_1, \dots, B_\ell\}$ ($\ell = \|\mathcal{B}\|$ will be defined afterwards; intuitively $\ell \gg k$), is the set of dummy organizations needed for our construction.

We divide the time into $(\|\mathcal{B}\|^2 + 3)$ independent batches. The batches are constructed in such a way that the $(j+1)$-th batch starts after the time in which all the jobs released in j-th batch are completed in every coalition (thus, the duration of the batch can be just the maximum release time plus the sum of the processing times of the jobs released in this batch). As the result, the contribution $\phi^{(u)}$ of each organization $O^{(u)}$ is the sum of its contributions in the all $(\|\mathcal{B}\|^2 + 3)$ batches. For the sake of the clarity of the presentation we assume that time moments in each batch are counted from 0.

We start from the following observation: if the sum of the processing times of the jobs in a batch is equal to p_{sum}, then the contribution of each organization can be upper bounded by p_{sum}^2. This observation follows from the fact that any organization, when joining a coalition, cannot decrease the completion time of any job by more than p_{sum}. As the total number of unit-size parts of the jobs is also p_{sum}, we infer that the joining organization cannot increase the value of the coalition by more than p_{sum}^2. The second observation is the following: if the joining organization causes decrease of the completion time of the task with processing time p, then its contribution is at least equal to $\frac{p}{\|\mathcal{O}\|!}$ (as it must decrease the start time of the job by at least one time unit in at least one coalition).

Let $x_{tot} = \sum_{j=1}^{k} x_j$. In our construction we use 4 large numbers L, XL, H and XH, where $L = (\|\mathcal{O}\|+1+4\|\mathcal{B}\|^2 x_{tot}^2)\cdot\|\mathcal{O}\|!$; $XL = (\mathcal{O}!\cdot L\cdot\|\mathcal{O}\|(\|\mathcal{O}\|+1))^2+\mathcal{O}!4\|\mathcal{B}\|^2x_{tot}^2+1$, $H = \|\mathcal{B}\|^2(2\|\mathcal{O}\|(1+x_{tot})+2x+XL)^2+1$ and XH is a very large number that will be defined afterwards. Intuitively: $XH \gg H \gg XL \gg L \gg x_{tot}$.

In the first batch only the organizations from $\mathcal{B}$ release their jobs. The i-th organization from $\mathcal{B}$ releases $2i$ jobs in time 0, each of size L. This construction is used to ensure that after the first batch the i-th organization from $\mathcal{B}$ has the difference $(\phi^{(i)} - \psi^{(i)})$ greater than the difference $(\phi^{(i+1)} - \psi^{(i+1)})$ of the $(i+1)$-th organization from $\mathcal{B}$ of at least $\frac{L}{\|\mathcal{O}\|!} = (\|\mathcal{O}\|+1+4\|\mathcal{B}\|^2x_{tot}^2)$ and of at most $p_{sum}^2 = (L\cdot\frac{\|\mathcal{B}\|(\|\mathcal{B}\|+1)}{2})^2 < \frac{XL}{\mathcal{O}!} - 4\|\mathcal{B}\|^2x_{tot}^2$.

In the second batch, at time 0, all the organizations except for a release 2 jobs, each of size H. This construction is used to ensure that after the second batch the contribution (and so the the difference $(\phi-\psi)$) of the organization a is large (at least equal to H, as a joining any coalition causes the job of size H to be scheduled at least one time unit earlier). Since in each of the next $\|\mathcal{B}\|^2$ batches the total size of the released jobs will be lower than $(2\|\mathcal{O}\|(1+x_{tot})+2x+XL)$, we know that in each of the next $\|\mathcal{B}\|^2$ batches the jobs of a will be prioritized over the jobs of the other organizations.

Each of the next $\|\mathcal{B}\|^2$ batches is one of the $2\|\mathcal{B}\|$ different types. For the organization B_i ($1 \leq i \leq \|\mathcal{B}\|$) there is exactly i batches of type $\texttt{Bch}(B_i,\ 2x+1)$ and $(\|\mathcal{B}\|-i)$ batches of type $\texttt{Bch}(B_i,\ 2x)$. The order of these $\|\mathcal{B}\|^2$ batches can be arbitrary.

The batches $\texttt{Bch}(B_i,\ 2x)$ and $\texttt{Bch}(B_i,\ 2x+1)$ are similar. The only difference is in the jobs of the organization a. In the batch $\texttt{Bch}(B_i,\ 2x)$ the organization a has two jobs $J_1^{(a)}$ and $J_2^{(a)}$, with release times $r_1^{(a)} = 0$ and $r_2^{(a)} = 2x$ and processing times $p_1^{(a)} = 2x+1$ and $p_2^{(a)} = XL$. In the batch $\texttt{Bch}(B_i,\ 2x+1)$ the organization a has two jobs $J_1^{(a)}$ and $J_2^{(a)}$, with release times $r_1^{(a)} = 0$ and $r_2^{(a)} = 2x+1$ and processing times $p_1^{(a)} = 2x+2$ and $p_2^{(a)} = XL$. All other organizations have the same jobs in batches $\texttt{Bch}(B_i,\ 2x)$ and $\texttt{Bch}(B_i,\ 2x+1)$. The organization B_i has no jobs and all the other organizations from $\mathcal{B}$ release a single job of size $(2x_{tot}+2)$ in time 0. The

j-th organization from $\mathcal{O}_S$ has two jobs $J_1^{(j)}$ and $J_2^{(j)}$, with release times $r_1^{(j)} = 0$ and $r_2^{(j)} = 1$ and processing times $p_1^{(j)} = 2x_{tot} + 1$ and $p_2^{(j)} = 2x_j$.

Finally, in the last $(\|\mathcal{B}\|^2 + 3)$-th batch only the organizations from $\mathcal{B}$ release their jobs. Each such organization releases $\|\mathcal{O}\|$ jobs in time 0, each of size XH.

Now let us compare the schedules for the batches $\texttt{Bch}(B_i,\, 2x)$ and $\texttt{Bch}(B_i,\, 2x+1)$ (see Figure 8.5). Let us consider a schedule for a coalition $\mathcal{C}'$. Let $\mathcal{O}_{S,\mathcal{C}'} = \mathcal{O}_S \cap \mathcal{C}'$; let $\mathcal{B}_{\mathcal{C}'} = \mathcal{B} \cap \mathcal{C}' \setminus \{B_i\}$. Let $\mathcal{J}_1$ denote the set of $\|\mathcal{O}_{S,\mathcal{C}'}\|$ jobs of sizes $2x_{tot} + 1$ (these are the first jobs of the organizations from $\mathcal{O}_{S,\mathcal{C}'}$). Let $\mathcal{J}_2$ denote the set of $\|\mathcal{O}_{S,\mathcal{C}'}\|$ jobs of sizes from S (the second jobs of the organizations from $\mathcal{O}_{S,\mathcal{C}'}$). Let $\mathcal{J}_3$ denote the $\|\mathcal{B}_{\mathcal{C}'}\|$ jobs of the organizations from $\mathcal{B}_{\mathcal{C}'}$ of sizes $2x_{tot} + 2$ (the single jobs of these organizations).

If $\sum_{x_i : O_i \in \mathcal{O}_{S,\mathcal{C}'}} x_i > x$ or $\sum_{x_i : O_i \in \mathcal{O}_{S,\mathcal{C}'}} x_i < x$ the schedules for any $\mathcal{C}'$ in batches $\texttt{Bch}(B_i,\, 2x)$ and $\texttt{Bch}(B_i,\, 2x + 1)$ looks similarly. In time 0, $\|\mathcal{O}_{S,\mathcal{C}'}\|$ machines will schedule the $\|\mathcal{O}_{S,\mathcal{C}'}\|$ jobs from $\mathcal{J}_1$ (let us denote these machines as $\mathcal{M}$) and $\|\mathcal{B}_{\mathcal{C}'}\|$ machines will schedule the $\|\mathcal{B}_{\mathcal{C}'}\|$ jobs from $\mathcal{J}_3$. If $B_i \notin \mathcal{C}'$ then the jobs from $\mathcal{J}_2$ will be scheduled on the machines from $\mathcal{M}$ just after the jobs from $\mathcal{J}_1$. If $B_i \in \mathcal{C}'$ and $a \notin \mathcal{C}'$, then the coalition $\mathcal{C}'$ has $(\|\mathcal{O}_{S,\mathcal{C}'}\| + \|\mathcal{B}_{\mathcal{C}'}\| + 1)$ machines; one machine will execute the jobs from $\mathcal{J}_2$. If $B_i \in \mathcal{C}'$ and $a \in \mathcal{C}'$ then the coalition $\mathcal{C}'$ has $(\|\mathcal{O}_{S,\mathcal{C}'}\| + \|\mathcal{B}_{\mathcal{C}'}\| + 2)$ machines. One machine (denoted as M') will execute the jobs from $\mathcal{J}_2$ and one other machine (denoted as M'') will execute the job $J_1^{(a)}$. Now, if $\sum_{x_i : O_i \in \mathcal{O}_{S,\mathcal{C}'}} x_i < x$ then $J_2^{(a)}$ will be scheduled on M'; otherwise on M'' (this follows from the construction in the second batch – we recall that the jobs of a should be prioritized). Thus, as explained in Figure 8.5, if $\sum_{x_i : O_i \in \mathcal{O}_{S,\mathcal{C}'}} x_i > x$ or $\sum_{x_i : O_i \in \mathcal{O}_{S,\mathcal{C}'}} x_i < x$ the contribution and the utility of each organization from $\mathcal{B}$ in two batches $\texttt{Bch}(B_i,\, 2x)$ and $\texttt{Bch}(B_i,\, 2x + 1)$ differ by at most $4x_{tot}^2$.

If $\sum_{x_i : O_i \in \mathcal{O}_{S,\mathcal{C}'}} x_i = x$, then the schedules for the cases: (i) $B_i \notin \mathcal{C}'$ (ii) ($B_i \in \mathcal{C}'$ and $a \notin \mathcal{C}'$) remain the same as in case $\sum_{i : O_i \in \mathcal{O}_{S,\mathcal{C}'}} x_i \neq x$. For the last case ($B_i \in \mathcal{C}'$ and $a \in \mathcal{C}'$) the jobs from $\mathcal{J}_1$, from $\mathcal{J}_2$ and $J_1^{(a)}$ are scheduled in the same way as previously. However, the job $J_2^{(a)}$ will be scheduled in $\texttt{Bch}(B_i,\, 2x+1)$ on machine M' (in the moment it is released) and in $\texttt{Bch}(B_i,\, 2x)$ on machine M' or M'' (one time unit later than it was released). As explained in Figure 8.5, if there exists an x-sum subset $S' \subset S$, then the contribution of B_i in $\texttt{Bch}(B_i,\, 2x + 1)$ will be greater by at least of $\frac{XL}{\mathcal{O}!} - 4x_{tot}^2$ than in $\texttt{Bch}(B_i,\, 2x)$.

As the result, if there does not exist an x-sum subset $S' \subset S$, then the difference $(\phi^{(i)} - \psi^{(i)})$ for the i-th organization from $\mathcal{B}$ will be greater than the difference $(\phi^{(i+1)} - \psi^{(i+1)})$ for the $(i + 1)$-th organization from $\mathcal{B}$ by at least $(\|\mathcal{O}\| + 1)$ (from the construction in the first batch the difference $(\phi^{(i)} - \psi^{(i)})$ was greater than $(\phi^{(i+1)} - \psi^{(i+1)})$ by at least $(\|\mathcal{O}\| + 1 + 4\|\mathcal{B}\|^2 x_{tot}^2)$, and as explained in Figure 8.5 the difference $(\phi^{(i+1)} - \psi^{(i+1)} - \phi^{(i)} + \psi^{(i)})$ could change by at most $4\|\mathcal{B}\|^2 x_{tot}^2$). Otherwise, the difference $(\phi^{(i)} - \psi^{(i)})$ for the i-th organization will be lower than for the $(i+1)$-th

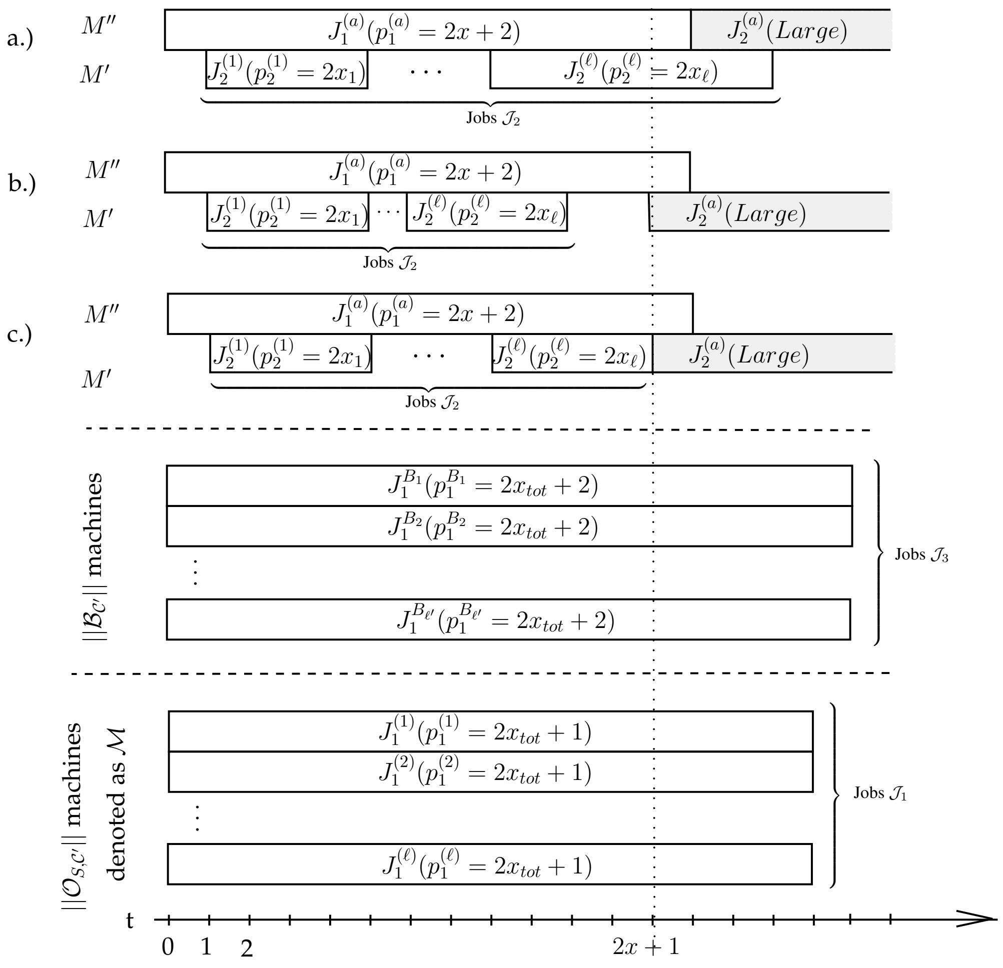
a.)
M''
$J_1^{(a)}(p_1^{(a)} = 2x+2)$
$J_2^{(a)}(Large)$
M'
$J_2^{(1)}(p_2^{(1)} = 2x_1)$
$\cdots$
$J_2^{(\ell)}(p_2^{(\ell)} = 2x_\ell)$
Jobs $\mathcal{J}_2$
b.)
M''
$J_1^{(a)}(p_1^{(a)} = 2x+2)$
M'
$J_2^{(1)}(p_2^{(1)} = 2x_1)$
$J_2^{(\ell)}(p_2^{(\ell)} = 2x_\ell)$
$J_2^{(a)}(Large)$
Jobs $\mathcal{J}_2$
c.)
M''
$J_1^{(a)}(p_1^{(a)} = 2x+2)$
M'
$J_2^{(1)}(p_2^{(1)} = 2x_1)$
$J_2^{(\ell)}(p_2^{(\ell)} = 2x_\ell)$
$J_2^{(a)}(Large)$
Jobs $\mathcal{J}_2$
$\|\mathcal{B}_{\mathcal{C}'}\|$ machines
$J_1^{B_1}(p_1^{B_1} = 2x_{tot}+2)$
$J_1^{B_2}(p_1^{B_2} = 2x_{tot}+2)$
$J_1^{B_{\ell'}}(p_1^{B_{\ell'}} = 2x_{tot}+2)$
Jobs $\mathcal{J}_3$
$\|\mathcal{O}_{S,\mathcal{C}'}\|$ machines denoted as $\mathcal{M}$
$J_1^{(1)}(p_1^{(1)} = 2x_{tot}+1)$
$J_1^{(2)}(p_1^{(2)} = 2x_{tot}+1)$
$J_1^{(\ell)}(p_1^{(\ell)} = 2x_{tot}+1)$
Jobs $\mathcal{J}_1$
t
0
1
2
$2x+1$

Figure 8.5: The schedule for the coalition $\mathcal{C}'$ such that $B_i \in \mathcal{C}'$ and $a \in \mathcal{C}'$ in batch $\texttt{Bch}(B_i,\, 2x+1)$, for 3 cases: a) $\sum_{x_i : O_i \in \mathcal{O}_{S,C'}} x_i > x$, b) $\sum_{x_i : O_i \in \mathcal{O}_{S,C'}} x_i < x$, c) $\sum_{x_i : O_i \in \mathcal{O}_{S,C'}} x_i = x$. We compare B_i's contribution ϕ on this schedule to schedule $\texttt{Bch}(B_i,\, 2x)$ (not shown; the only differences are that $p_1^{(a)} = 2x+1$ and $r_2^{(a)} = 2x$). Other organizations $B_j \neq B_i$ have utility equal to contribution in all cases considered here. As B_i has no jobs, it contributes only a single machine (corresponding to M'). Thanks to M', the small jobs $J_2^{(i)}$ execute at most $2x_{tot}$ earlier (if there is no machine M', these jobs are executed at $\mathcal{M}$). The total size of these small jobs is $2x_{tot}$. Regarding small jobs, the resulting contribution of B_i to $\mathcal{C}'$ is bounded by $4x_{tot}^2$.

In case a) M' does not decrease the start time of the large job $J_2^{(a)}$; the same happens in batch $\texttt{Bch}(B_i,\, 2x)$. In case b) M' speeds up $J_2^{(a)}$ by 1; the same happens in batch $\texttt{Bch}(B_i,\, 2x)$. In case c) M' also speeds up $J_2^{(a)}$ by 1; however, in batch $\texttt{Bch}(B_i,\, 2x)$ M' does not decrease $J_2^{(a)}$'s start time ($J_2^{(a)}$ is always started at $(2x+1)$). To summarize, B_i contribution to $\mathcal{C}'$ in both a) and b) differs by at most $4x_{tot}^2$ between $\texttt{Bch}(B_i,\, 2x)$ and $\texttt{Bch}(B_i,\, 2x+1)$. In contrast, in c) the contribution in $\texttt{Bch}(B_i,\, 2x+1)$ is greater by at least $XL - 4x_{tot}^2$ compared to the contribution in $\texttt{Bch}(B_i,\, 2x)$.

As a consequence, considering B_i's contribution to all coalitions, if there exists an x-sum subset $S' \subset S$ (case c), then the contribution of B_i in $\texttt{Bch}(B_i,\, 2x+1)$ is by at least $\frac{XL}{\|\mathcal{O}\|!} - 4x_{tot}^2$ greater than in $\texttt{Bch}(B_i,\, 2x)$; if there is no such an x-sum subset, then the contribution of B_i in $\texttt{Bch}(B_i,\, 2x+1)$ and in $\texttt{Bch}(B_i,\, 2x)$ differ by no more than $4x_{tot}^2$.

organization (as there are more batches of type $\mathtt{Bch}(B_{i+1}, 2x+1)$ than of type $\mathtt{Bch}(B_i, 2x+1)$).

Thus, if there does not exist an x-sum subset $S' \subset S$, then in the last batch the jobs of B_1 will be scheduled first, than the jobs of B_2, and so on – let us denote such schedule as σ_{ord}. On the other hand, if there exists an x-sum subset $S' \subset S$, the jobs in the last batch will be scheduled in the exactly reverse order – such schedule will be denoted as σ_{rev}.

Now, let us assess the distance between the vector of utilities in case of two schedules σ_{ord} and σ_{rev}. Let us assume that $\|\mathcal{B}\|$ is even. Every job of the organization B_i ($1 \le i \le \frac{\|\mathcal{B}\|}{2}$) in the last batch is started $XH(\|\mathcal{B}\| - 2i + 1)$ time units earlier in σ_{ord} than in σ_{rev}. The jobs of the organization $B_{(\|\mathcal{B}\|+1-i)}$ ($1 \le i \le \frac{\|\mathcal{B}\|}{2}$) are scheduled $XH(\|\mathcal{B}\| - 2i + 1)$ time units later in σ_{ord} than in σ_{rev}. Since each such job consists of XH unit-size elements, the distance between the vector of utilities for σ_{ord} and σ_{rev}, denoted as $\Delta\psi$, can be lower bounded by:

$$\Delta\psi \ge 2\|\mathcal{O}\| \sum_{i=1}^{\|\mathcal{B}\|/2} (2i-1)XH^2 = \|\mathcal{O}\|\|\mathcal{B}\| \frac{(1 + 1 + 2\frac{\|\mathcal{B}\|}{2} - 2)}{2} XH^2 = \frac{1}{2}\|\mathcal{O}\|\|\mathcal{B}\|^2 XH^2.$$

Now, we can define XH to be the total size of the all except the last batch times $\frac{4}{\epsilon}$. Below we show how to bound the total utility ψ_{tot} of the true fair schedule (σ_{ord} or σ_{rev}) in the time t when all the jobs are completed. Each unit size part of the job completed in time t contributes to the utility the value 1. Each unit size part of the job executed in time $t-1$ is worth 2, and so on. Since the jobs in the last batch are executed on $\|\mathcal{O}\|$ machines and the duration of the batch is equal to $\|\mathcal{B}\|XH$, the utility of the jobs from the last batch is equal to $\sum_{i=1}^{\|\mathcal{B}\|XH} i$. The jobs in all previous batches are started no earlier than in $t - \|\mathcal{B}\|XH - \frac{\epsilon}{4}XH$. The duration of the all but the last batch can be upper bounded by $\frac{\epsilon}{4}XH$. There are $\|\mathcal{O}\|$ machines, so the utility of the jobs from the all but the last batch can be upper bounded by $(\|\mathcal{B}\|XH + \frac{\epsilon}{4}XH)\frac{\epsilon}{4}XH$. Thus we get the following bound on ψ_{tot}:

$$\begin{aligned}
\psi_{tot} &< \|\mathcal{O}\| \left(\sum_{i=1}^{\|\mathcal{B}\|XH} i + \left(\|\mathcal{B}\|XH + \frac{\epsilon}{4}XH)(\frac{\epsilon}{4}XH \right) \right) \\
&\le \|\mathcal{O}\| \left(\frac{1 + \|\mathcal{B}\|XH}{2} \|\mathcal{B}\|XH + \frac{\epsilon}{4}\|\mathcal{B}\|XH^2 + \frac{\epsilon}{16}^2 XH^2 \right) \\
&\le \|\mathcal{O}\| \left(\frac{1}{2}(1 + \|\mathcal{B}\|XH)^2 + \frac{\epsilon}{4}\|\mathcal{B}\|^2 XH^2 \right) \\
&\le \|\mathcal{O}\|\|\mathcal{B}\|^2 XH^2 \left(\frac{1}{2} \cdot \left(\frac{1 + \|\mathcal{B}\|}{\|\mathcal{B}\|} \right)^2 + \frac{\epsilon}{4} \right).
\end{aligned}$$

We can chose the size $\|\mathcal{B}\|$ so that $\left(\frac{1+\|\mathcal{B}\|}{\|\mathcal{B}\|}\right)^2 < 1 + \frac{\epsilon}{2}$. As the result we have:

$$\Delta\psi/\psi_{tot} > \frac{1}{2}/\frac{1}{2}\left(\left(\frac{1+\|\mathcal{B}\|}{\|\mathcal{B}\|}\right)^2 + \frac{\epsilon}{2}\right) > \frac{1}{1+\epsilon} > 1 - \epsilon$$

Finally let us assume that there exists $(\frac{1}{2} - \epsilon)$-approximation algorithm $\mathcal{A}$ that returns the schedule σ for our instance. Now, if σ is closer to σ_{ord} than to σ_{rev}, we can infer that σ_{ord} is a true fair solution to our instance (and so the answer to the initial SubsetSum question is "yes"). Otherwise, σ_{rev} is a true solution (and the answer to the SubsetSum problem is "no"). This completes the proof.

□

8.5.1 Special Case: Unit-Size Jobs

In case where the jobs are unit-size, the problem has additional properties that allow us to construct an efficient approximation algorithm (however, the worst-case complexity of this special case is open). The results in this section do not generalize to related or unrelated processors and apply to the setting with identical processors only. For unit-size jobs, the value of each coalition $v(\mathcal{C})$ does not depend on the schedule:

Proposition 8.8. *For any two greedy algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, for each coalition $\mathcal{C}$ and each time moment t, the values of the coalitions $v(\mathcal{A}_1, \mathcal{C}, t)$ and $v(\mathcal{A}_2, \mathcal{C}, t)$ are equal, provided all jobs are unit-size.*

Proof. We prove the following stronger thesis: For every time moment t any two greedy algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ schedule the same number of the jobs till t. We prove this thesis by induction. The base step for $t = 0$ is trivial. Having the thesis proven for $(t-1)$ and, thus knowing that in t in both schedules there is the same number of the jobs waiting for execution (here we use the fact that the jobs are unit-size), we infer that in t the two algorithms schedule the same number of the jobs. Since the value of the coalition does not take into account the owner of the job, we get the thesis for t. This completes the proof. □

As a result, we can use a randomized approximation algorithm for the scheduling problem restricted to unit-size jobs (Algorithm Rand from Figure 8.6). The algorithm is inspired by the randomized approximation algorithm for computing the Shapley value presented by Liben-Nowell et al [177]. However, in our case the game is not supermodular (which is shown in Proposition 8.9 below), and so we have to adapt the algorithm and, thus, we obtain different approximation bounds.

Proposition 8.9. *In case of unit-size jobs, the cooperation game in which the value of each coalition $\mathcal{C}$ is defined by $v(\mathcal{C}) = \sum_{O^{(u)} \in \mathcal{C}} \psi(O^{(u)})$ is not supermodular.*

```
Notation:
$\epsilon$, $\lambda$ — as in Theorem 8.10

Prepare($\mathcal{C}$):
    $N \leftarrow \lceil \frac{\|\mathcal{C}\|^2}{\epsilon^2} \ln\left(\frac{\|\mathcal{C}\|}{1-\lambda}\right) \rceil$;
    $\Gamma \leftarrow$ generate $N$ random orderings (permutations) of the set of all organizations (with
    replacement);
    $Subs \leftarrow Subs' \leftarrow \emptyset$ ;
    foreach $\prec \in \Gamma$ do
        for $u \leftarrow 1$ to $\|C\|$ do
            $\mathcal{C}' \leftarrow \{O^{(i)} : O^{(i)} \prec O^{(u)}\}$ ;
            $Subs \leftarrow Subs \cup \{\mathcal{C}'\}$; $Subs' \leftarrow Subs' \cup \{\mathcal{C}' \cup \{O^{(u)}\}\}$ ;

ReleaseJob($O^{(u)}$, $J$):
    for $\mathcal{C}' \in Subs \cup Subs' : O^{(u)} \in \mathcal{C}'$ do
        jobs$[\mathcal{C}'][O^{(u)}]$.push$(J)$

SelectAndSchedule($\mathcal{C}$, $t$):
    $u \leftarrow \text{argmin}_{O^{(u)}}(\psi[\mathcal{C}][O^{(u)}] - \phi[\mathcal{C}][O^{(u)}])$ ;
    $\sigma[\mathcal{C}] \leftarrow \sigma[\mathcal{C}] \cup \{(\text{jobs}[\mathcal{C}][u].\text{first}, t)\}$;
    finPerOrg$[O^{(u)}] \leftarrow$ finPerOrg$[O^{(u)}] + 1$;
    $\phi[O^{(u)}] \leftarrow \phi[O^{(u)}] + 1$;

FairAlgorithm($\mathcal{C}$):
    Prepare($\mathcal{C}$) ;
    foreach time moment $t$ do
        foreach job $J_i^{(u)}$: $r_i^{(u)} = t$ do
            ReleaseJob($O_i^{(u)}$, $J_i^{(u)}$);
        foreach $\mathcal{C}' \subset Subs \cup Subs'$ do
            v$[\mathcal{C}'] \leftarrow$ v$[\mathcal{C}']$ + finPerCoal$[\mathcal{C}']$ ;
            $n \leftarrow \min(\sum_{O^{(u)} \in \mathcal{C}'} m^{(u)}, \|\text{jobs}[\mathcal{C}][O^{(u)}]\|)$ ;
            remove first $n$ jobs from jobs$[\mathcal{C}][O^{(u)}]$ ;
            finPerCoal$[\mathcal{C}'] \leftarrow$ finPerCoal$[\mathcal{C}'] + n$ ;
            v$[\mathcal{C}'] \leftarrow$ v$[\mathcal{C}'] + n$ ;
        foreach $O^{(u)} \in \mathcal{C}$ do
            $\psi[O^{(u)}] \leftarrow \psi[O^{(u)}]$ + finPerOrg$[O^{(u)}]$;
            $\phi[O^{(u)}] \leftarrow 0$;
            foreach $\mathcal{C}' \in Subs : O^{(u)} \notin \mathcal{C}'$ do
                marg_$\phi \leftarrow$ v$[\mathcal{C}' \cup \{O^{(u)}\}]$ − v$[\mathcal{C}']$ ;
                $\phi[O^{(u)}] \leftarrow \phi[O^{(u)}]$ + marg_$\phi \cdot \frac{1}{N}$;
        while FreeMachine($\sigma[\mathcal{C}]$, $t$) do
            SelectAndSchedule($\mathcal{C}, t$);
```

Figure 8.6: Algorithm RAND: A fair algorithm for the specific utility function ψ_{sp} and for unit-size jobs.

Proof. Consider a following instance with 3 organizations: a, b and c each owning a single machine. Organizations a and b in time $t = 0$ release two unit size jobs each; the organization c has no jobs. We are considering the values of the coalitions in time $t = 2$; $v(\{a, c\}) = 4$ (the two jobs are scheduled in time 0), $v(\{b, c\}) = 4$, $v(\{a, b, c\}) = 7$ (three jobs are scheduled in time 0 and one in time 1) and $v(\{c\}) = 0$ (there is no job to be scheduled). We see that $v(\{a, b, c\}) + v(\{c\}) < v(\{a, c\}) + v(\{b, c\})$, which can be written as:

$$v(\{a, c\} \cup \{b, c\}) + v(\{a, c\} \cap \{b, c\}) < v(\{a, c\}) + v(\{b, c\}).$$

This shows that the game is not supermodular. $\square$

In our algorithm, we keep simplified schedules for a random subset of all possible coalitions. For each organization $O^{(u)}$, the set $Subs[O^{(u)}]$ keeps $N = \frac{\|\mathcal{C}\|^2}{\epsilon^2} \ln\left(\frac{\|\mathcal{C}\|}{1-\lambda}\right)$ random coalitions not containing $O^{(u)}$; for each such random coalition $\mathcal{C}'$ that is kept in $Subs[O^{(u)}]$, $Subs'[O^{(u)}]$ contains the coalition $\mathcal{C}' \cup \{O^{(u)}\}$. For the coalitions kept in $Subs[O^{(u)}]$, we store a simplified schedule (the schedule that is determined by an arbitrary greedy algorithm). The simplified schedule allows us to find the value $v(\mathcal{C}')$ of the coalition $\mathcal{C}'$. (Maintaining the whole schedule would require recursive information about the schedules in the subcoalitions of $\mathcal{C}'$.) As a consequence of Proposition 8.8, we know that the value $v(\mathcal{C}')$ of coalition $\mathcal{C}'$ can be determined by an arbitrary greedy algorithm.[6]

The third *foreach* loop in procedure `FairAlgorithm` (line 22 in Figure 8.6) updates the values of all coalitions kept in *Subs* and *Subs*'. From Equation 8.2, it follows that after one time unit, if no additional job is scheduled, the value of the coalition increases by the number of completed unit-size parts of the jobs (here, as the jobs are unit-size, the number of the completed jobs is finPerCoal[$\mathcal{C}'$]). In time moment t, all waiting jobs (the number of such jobs is $\|\text{jobs}[\mathcal{C}][O^{(u)}]\|$) are scheduled provided there are enough processors (the number of the processors is $\sum_{O^{(u)} \in \mathcal{C}'} m^{(u)}$). If n additional jobs are scheduled in time t then the value of the coalition in time t increases by n.

In the fourth *foreach* loop (line 28 in Figure 8.6), once again we use the fact that the utility of the organization after one time unit increases by the number of finished jobs (finPerOrg[$O^{(u)}$]). In the last *foreach* loop (line 31), the contribution of each organization is approximated by summing the marginal contributions, marg_ϕ: however, we do not sum the marginal contributions of all the coalitions but only the marginal contributions of these coalitions that we initially randomly selected. Theorem 8.10 below gives the bounds for the quality of approximation.

Theorem 8.10. *Let $\vec{\psi}$ denote the vector of utilities in the schedule determined by Algorithm* Rand *from Figure 8.6. If the jobs are unit-size, then $\mathcal{A}$ with the probability*

[6]In this point we use the assumption about the unit size of the jobs. The algorithm cannot be extended to the general case. In a general case, for calculating the value for each subcoalition we would require the exact schedule which cannot be determined polynomially (Theorem 8.6).

λ determines the ϵ-approximation schedule, i.e. gives guarantees for the bound on the distance to the truly fair solution:

$$\|\vec{\psi} - \vec{\psi}^*\|_M \leq \epsilon|\vec{\psi}^*|.$$

Proof. Let us consider an organization $O^{(u)}$ participating in a coalition $\mathcal{C}$ and a time moment t. Let $\phi^{(u),*}$ and $\psi^{(u),*}$ denote the contribution and the utility of the organization $O^{(u)}$ in a coalition $\mathcal{C}$ in time moment t in a truly fair schedule. Let $v^*(\mathcal{C})$ denote the value of the coalition $\mathcal{C}$ in a truly fair schedule. According to notation in Figure 8.6, let $\phi[O^{(u)}]$ and $\psi[O^{(u)}]$ denote the contribution and the utility of the organization $O^{(u)}$ in a coalition $\mathcal{C}$ in time t in a schedule determined by Algorithm RAND; Let $N = \frac{\|\mathcal{C}\|^2}{\epsilon^2}\ln\left(\frac{\|\mathcal{C}\|}{1-\lambda}\right)$. First, note that $|\psi^{(u),*} - \psi[O^{(u)}]| \leq |\phi^{(u),*} - \phi[O^{(u)}]|$. Indeed, if the contribution of the organization $O^{(u)}$ increases by a given value $\Delta\phi$ then Algorithm RAND will schedule $\Delta\phi$ more unit-size jobs of the organization $O^{(u)}$ provided there is enough such jobs waiting for execution.

Let X denote the random variable that with the probability $\frac{1}{\|\mathcal{C}\|!}$ returns the marginal contribution of the organization $O^{(u)}$ to the coalition composed of the organizations preceding $O^{(u)}$ in the random order (of course, there is $\|\mathcal{C}\|!$ such random orderings). We know that $X \in [0, v^*(\mathcal{C})]$ and that $\mathbb{E}(X) = \phi^{(u),*}$. Algorithm RAND is constructed in such a way that $\phi[O^{(u)}] = \sum_{i=0}^{N}\frac{1}{N}X_i$, where X_i are independent copies of X. Thus, $\mathbb{E}(\phi[O^{(u)}]) = \phi^{(u),*}$. From Hoeffding's inequality we get the bound on the probability p_ϵ that $\phi[O^{(u)}] - \phi^{(u),*} > \frac{\epsilon}{\|\mathcal{C}\|}v^*(\mathcal{C})$:

$$\begin{aligned} p_\epsilon &= \mathrm{P}\left(\sum_{i=0}^{N}\frac{1}{N}X_i - \phi^{(u),*} > \frac{\epsilon}{\|\mathcal{C}\|}v^*(\mathcal{C})\right) \\ &< \exp\left(-\frac{\epsilon^2 v^*(\mathcal{C})^2 N^2}{v^*(\mathcal{C})^2 N\|\mathcal{C}\|^2}\right) \\ &= \exp\left(-\frac{\epsilon^2 N}{\|\mathcal{C}\|^2}\right) = \frac{1-\lambda}{\|\mathcal{C}\|}. \end{aligned}$$

The probability that $\vec{\phi} - \vec{\phi}^* > \epsilon v^*(\mathcal{C})$ can be bounded by $p_\epsilon\|\mathcal{C}\| = 1 - \lambda$. As the result, also the probability that $\vec{\psi} - \vec{\psi}^* > \epsilon v^*(\mathcal{C})$ can be bounded by $1 - \lambda$, which completes the proof. □

The complexity of Algorithm RAND is $\|\mathcal{O}\| \cdot N = \|\mathcal{O}\|\frac{\|\mathcal{C}\|^2}{\epsilon^2}\ln\left(\frac{\|\mathcal{C}\|}{1-\lambda}\right)$ times the complexity of the single-organization scheduling algorithm. As a consequence, we get the following result.

Theorem 8.11. *There exists an FPRAS for the problem of finding the fair schedule for the case when the jobs are unit size.*

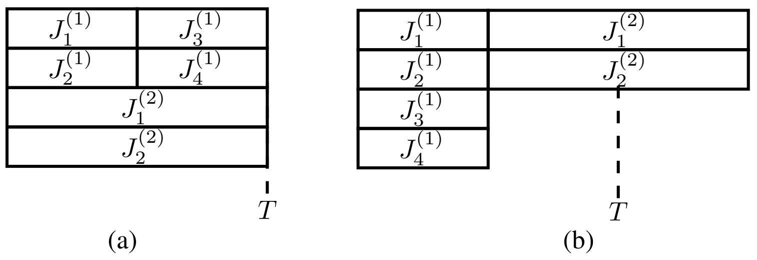

Figure 8.7: The example showing that greedy algorithms might induce suboptimal resource utilization. In the example we have 4 jobs of the organization $O^{(1)}$, each of size 3, and 2 jobs of the organization $O^{(2)}$, each of size 6. All the jobs are released in time 0. Let us consider time moment $T = 6$. In Figure (a) the jobs of $O^{(2)}$ are started first, which results in 100% resource utilization. In Figure (b) the jobs of $O^{(1)}$ are started first, which in time T gives 75% of resource utilization.

8.6 Resource Utilization of Greedy Algorithms

It might appear that in order to ensure the fairness of the algorithm we might be forced to use globally inefficient algorithms. Such algorithms might, for instance, waste resources. We define the resource utilization as the percentage of the time in which, on average, every processor is busy. The resource utilization is an established metric indicating the global efficiency of resource usage. Indeed, even though we use greedy algorithms, some of them might result in suboptimal resource utilization. This problem is shown in Figure 8.7.

Thus, there is a natural question, which in additional to the context of fair scheduling, is interesting on its own. How bad can we do in terms of resource utilization when using a greedy algorithm (with any underlying scheduling policy)? In the next theorem we show that the example from Figure 8.7 is, essentially, the worst possible scenario.

Definition 8.4. *An algorithm $\mathcal{A}$ is an α-competitive online algorithm for resource utilization if and only if in each time moment T the ratio of the resource utilization between the schedule derived by $\mathcal{A}$ and the schedule obtained by any other algorithm is greater or equal to α.*

Theorem 8.12. *Every greedy algorithm for scheduling sequential jobs on identical processors is a $\frac{3}{4}$-competitive online algorithm for resource utilization.*

Proof. Let σ denote the schedule obtained by some greedy algorithm $\mathcal{A}$ until time T, and let σ^* denote the schedule obtained by the optimal (according to the resource utilization metric) algorithm for the same input. Now, we will divide the time axis into blocks in the following way. The first block starts in time 0. The i-th block ($i > 1$) starts in the earliest possible time moment t_i such that (i) $t_i > t_{i-1}$ (the i-th

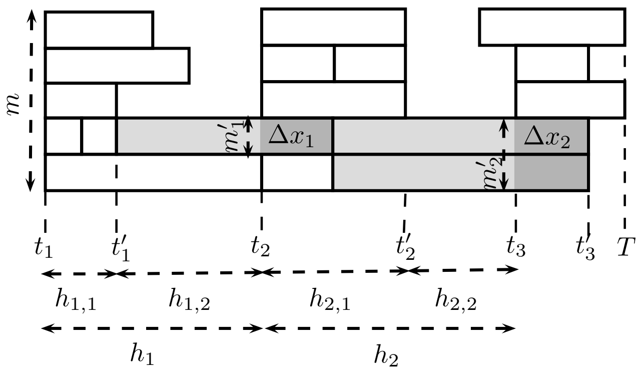

Figure 8.8: Illustration of the notation used in the proof of Theorem 8.12. There are 3 time blocks in this figure: the first one lasts from t_1 until t_2; the second from t_2 until t_3; and the third one from t_3 until T. The parts of jobs that were delayed outside their time block (in comparison with an optimal schedule) are marked in dark gray. The remaining parts of these jobs are marked in light gray. We see that these jobs which are delayed outside the i-th time block are started at or before t'_i.

block starts after the $(i-1)$-th one), and (ii) in t_i there are jobs in σ running on all the processors and in $(t_i - 1)$ at least one processor in σ is idle. Let t_ℓ denote the start time of the last block. By convention we take $t_{\ell+1} = T$. The blocks for the example schedule are depicted in Figure 8.8. Furthermore, let t'_i denote the earliest moment in the i-th time block in which some processor is idle. Let $h_i = t_{i+1} - t_i$ denote the duration of the i-th time block. Let $h_{i,1} = t'_i - t_i$ and let $h_{i,2} = t_{i+1} - t'_i$.

In our proof we will consider the time blocks separately and for each time block we will prove that the total number of the unit-size parts of the jobs completed in this block in schedules σ and σ^* vary by no more than the factor of $\frac{3}{4}$. Throughout this proof we will use the variable V that, intuitively, accumulates the number of unit-size parts of the jobs that in σ were completed in the earlier time block than in σ^*. Let V_i denote the value of V after we completed an analysis for the i-th block, with $V_0 = 0$.

Let us consider the i-th time block. Let x_i and x_i^* denote the number of unit-size parts of the jobs completed in the i-th time block in schedules σ and σ^*, respectively. If $x_i \geq x_i^*$ then we increase the variable V by $(x_i - x_i^*)$. Otherwise, let $\Delta x_i = x_i^* - x_i > 0$. We consider the two following cases:

1. If $V_{i-1} = 0$, then we set $\Delta y_i = \Delta x_i$, and $V_i = 0$.

2. If $V_{i-1} > 0$, then we set $\Delta y_i = \Delta x_i - \min(\Delta x_i, V_{i-1})$, and $V_i = V_{i-1} - \min(\Delta x_i, V_{i-1})$. Intuitively, this means that the unit-size parts of jobs that were computed extra in earlier blocks and accumulated in V, pay for some parts of the jobs that were computed in the later block.

Now, if $\Delta y_i = 0$ this means that from V we managed to pay for the parts that, due to inefficiency of the algorithm $\mathcal{A}$, were not computed in the i-th time block. Otherwise ($\Delta y_i > 0$), we infer that some $\Delta y_i > 0$ parts of the jobs that were released before t_{i+1} were delayed and in σ were not completed in the i-th time block (while they were in σ^*). These jobs were released at or after t_i. Indeed, otherwise the job would be started at time $t_i - 1$ or earlier (the algorithm $\mathcal{A}$ is greedy, and a processor is idle at $t_i - 1$), and so, such a job would be processed for the whole duration of the i-th time block. Consequently, the unit-size parts of this job would not contribute to Δy_i (the number of unit-size parts of this job completed in the i-th time block in σ would be no greater than in σ^*).

Let us consider the jobs the parts of which contributed to Δy_i. Let m'_i denote the number of machines on which these jobs were processed (see Figure 8.8 for an example). From the pigeonhole principle, at least one from the considered jobs, J, was delayed by at least $\frac{\Delta y_i}{m'_i}$. Since the algorithm $\mathcal{A}$ is greedy, $h_{i,1} \geq \frac{\Delta y_i}{m'_i}$ (there were at least $\frac{\Delta y_i}{m'_i}$ time moments in the i-th time block with no idle processors; otherwise J would be started earlier). Also, each from the considered jobs starts in time t'_i at the latest—indeed, if it would be started later, then from the greediness of the algorithm we would infer that the release time of such job is at least $t'_i + 1$, and so, such job would not be delayed in σ. Consequently, through the whole duration of the i-th block some m'_i machines are continuously occupied. Thus, the idle surface of the processors in σ is at most equal to $h_{i,2}(m - m'_i)$. Since, $\Delta y_i \leq \Delta x_i$, and Δx_i denotes the difference in the number of unit-size parts of the jobs computed in σ^* and in σ, we infer that $\Delta y_i \leq h_{i,2}(m - m'_i)$.

Now, let us estimate o_i, the number of the occupied slots in the i-th block in schedule σ.

$$o_i \geq h_{i,1}m + h_{i,2}m'_i \geq \frac{\Delta y_i}{m'_i}m + \frac{\Delta y_i}{m - m'_i}m'_i \geq \Delta y_i \left(\frac{m}{m'_i} + \frac{m'_i}{m - m'_i}\right) \geq 3\Delta y_i.$$

Thus: $\frac{o_i}{o_i + \Delta y_i} \geq \frac{o_i}{o_i + \frac{1}{3}o_i} = \frac{3}{4}$. Since our reasoning can be repeated for every time block, we get the thesis.

□

This result shows that even without any information on jobs' release dates, durations, even with an arbitrary scheduling policy, we waste no more than 25% of the resources. Of course, this loss of efficiency is even smaller when there are many jobs to be computed. For instance, if at any time moment there are jobs waiting for execution, any greedy algorithm achieves 100% resource utilization.

It is also natural to consider the loss of efficiency according to our strategy-proof metric. We have chosen to consider resource utilization as it has more intuitive meaning in terms of the waste of resources. We leave the problem of finding bounds for our strategy-proof metric as a natural follow-up question.

We note that our fair scheduling algorithm is also applicable for parallel jobs (jobs requiring more than one processor). However, for the case of parallel jobs, the loss of the global efficiency of an arbitrary greedy algorithm can be higher. We leave these extensions, as well as generalization of the processor model to related and unrelated machines, for the future work.

8.7 Experimental Evaluation of the Algorithms

In the previous section we showed that the problem of finding a fair schedule is computationally intractable. However, the ideas used in the exponential and the FPRAS algorithms can be used as insights for creating reasonable heuristics. In this section we present experimental evaluation of the fairness of two simple heuristic algorithms, and for several algorithms known in the scheduling theory.

8.7.1 Algorithms

In this section we describe the algorithms that we evaluate.

Ref. We used Algorithm Ref from Figure 8.1 (which is an exponential algorithm) as the reference fair algorithm.

Rand. We used Algorithm Rand from Figure 8.6 as a heuristic for workloads with jobs having different sizes. We verify two versions of the algorithm with $N = 15$ and $N = 75$ random subcoalitions.

DirectContr (the pseudo-code of the algorithm is given in Figure 8.9). The algorithm keeps for each organization O its utility $\psi_{sp}[O]$ and its estimated contribution $\phi[O]$. The estimate of the contribution of each organization is assessed directly (without considering any subcoalitions) by the following heuristic. On each scheduling event t we consider the processors in a random order and assign waiting jobs to free processors. The job that is started on processor m increases the contribution $\tilde{\phi}$ of the owner of m by the utility of this job.

In the pseudo code, $\mathrm{finUt}[O]$ denotes the number of the unit-size parts of the jobs of organization O that are completed before t_{prev}. From Equation 8.2, we know that the utility in time t of the unit-size parts of the jobs of the organization O that are completed before t_{prev} is greater by the additive value of $(t - t_{prev})\mathrm{finUt}[O]$ than this utility in time t_{prev} (line 7); the utility of the unit-size parts of the job completed between t_{prev} and t is equal to $\sum_{i=1}^{t-t_{prev}} i = \frac{1}{2}(t-t_{prev})(t-t_{prev}+1)$ (line 15). Similarly, $\mathrm{finCon}[O]$ denotes the number of completed unit-size parts of the jobs processed on the processors of the organization O. The algorithm updates the utilities and the estimates of the contributions. The waiting jobs are assigned to the processors in the order of decreasing differences $(\phi - \psi)$ of the issuing organizations (similarly as in algorithm Ref).

```
Notation:
own(M), own(J) — the organization owning the processor $M$, the job $J$
$wait(O)$ — the set of released, but not-yet scheduled jobs of the organization $O$ at time $t$

Initialize($\mathcal{C}$):
    foreach $O^{(u)} \in \mathcal{C}$ do
        $\mathrm{finUt}[O^{(u)}] \leftarrow 0$; $\mathrm{finCon}[O^{(u)}] \leftarrow 0$ ;
        $\phi[O^{(u)}] \leftarrow 0$; $\psi[O^{(u)}] \leftarrow 0$ ;

Schedule($t_{prev}$, $t$): // $t_{prev}$ is the time of the previous event
    foreach $O^{(u)} \in \mathcal{C}$ do
        $\phi[O^{(u)}] \leftarrow \phi[O^{(u)}] + (t - t_{prev})\mathrm{finCon}[O^{(u)}]$;
        $\psi[O^{(u)}] \leftarrow \psi[O^{(u)}] + (t - t_{prev})\mathrm{finUt}[O^{(u)}]$;
    $\gamma \leftarrow$ generate a random permutation of the set of all processors;
    foreach $m \in \gamma$ do
        if not FreeMachine($m$, $t$) then
            $J \leftarrow$ RunningJob($m$);
            finUt[own($J$)] $\leftarrow$ finUt[own($J$)] $+ t - t_{prev}$ ;
            finCon[own($m$)] $\leftarrow$ finCon[own($m$)] $+ t - t_{prev}$ ;
            $\phi$[own($J$)] $\leftarrow \phi$[own($J$)] $+ \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$;
            $\psi$[own($m$)] $\leftarrow \psi$[own($m$)] $+ \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$;
    foreach $m \in \gamma$ do
        if FreeMachine($m$, $t$) and $\bigcup_{O^{(u)}} wait(O^{(u)}) \neq \emptyset$ then
            $org \leftarrow \mathrm{argmax}_{O^{(u)}:wait(O^{(u)})\neq\emptyset}(\phi[O^{(u)}] - \psi[O^{(u)}])$ ;
            $J \leftarrow$ first waiting job of $org$ ;
            startJob($J$, $m$) ;
            $\mathrm{finUt}[org] \leftarrow \mathrm{finUt}[org] + 1$ ;
            finCon[own($m$)] $\leftarrow$ finCon[own($m$)] $+ 1$ ;
```

Figure 8.9: Algorithm DirectContr: a heuristic algorithm for fair scheduling.

RoundRobin. The algorithm cycles through the list of organizations to determine the job to be started.

FairShare [159]. This is perhaps the most popular scheduling algorithm that uses the idea of distributive fairness. Each organization is given a target weight (a *share*). The algorithm tries to ensure that the resources used by different organizations are proportional to their shares. More formally, whenever there is a free processor and some jobs waiting for execution, the algorithm sorts the organizations in the ascending order of the following ratios: the total time of the processor already assigned for the jobs of the organization divided by its share. A job from the organization with the lowest ratio is started.

In all versions of fair share, in the experiments we set the target share to the fraction of processors contributed by an organization to the global pool.

UtFairShare. This algorithm uses the same idea as FairShare. The only difference is that UtFairShare tries to balance the utilities of the organizations instead of their resource allocation. Thus, in each step the job of the organization with the smallest ratio of utility to share is selected. We used this algorithm because it uses the allocation mechanism of FairShare, but operates on the strategy-proof metric used by our reference exponential algorithm.

CurrFairShare. This version of the fair share algorithm does not keep any history; it only ensures that, for each organization, the number of currently executing jobs is proportional to its target share. We used this algorithm because it is light and efficient. It has also an interesting property: the history does not influence the current schedule. We were curious to check how this property influences the fairness.

8.7.2 Settings

To run simulations, we chose the following workloads from the Parallel Workload Archive [100]: 1. LPC-EGEE[7] (cleaned version), 2. PIK-IPLEX,[8] 3. RICC,[9] 4. SHARCNET-Whale.[10] We selected traces that closely resemble sequential workloads (in the selected traces most of the jobs require a single processor). We replaced parallel jobs that required $q > 1$ processors with q copies of a sequential job having the same duration.

In each workload, each job has a user identifier (in the workloads there are respectively 56, 225, 176 and 154 distinct user identifiers). To distribute the jobs between the organizations we uniformly distributed the user identifiers between the organizations; each job sent by a given user was assigned to the corresponding organization.

Because Ref is exponential, the experiments are computationally-intensive; in most of the experiments, we simulate 5 organizations only.

The users usually send their jobs in consecutive blocks. We also considered a scenario when the jobs are uniformly distributed between organizations (corresponding to a case when the number of users within organizations is large, in which case the distribution of the jobs should be close to uniform). These experiments led to the same conclusions, so we present only the results from the case where the user identifiers were distributed between the organizations.

For each workload, the total number of processors in the system was equal to the number originally used in the workload (that is 70, 2560, 8192 and 3072, respectively). The processors were assigned to organizations so that the counts follow Zipf and (in different runs) uniform distributions.

For each algorithm, we compared the vector of the utilities (the utilities per organization) at the end of the simulated time period (a fixed time t_{end}), $\vec{\psi}$, with the vector of the utilities in the ideally fair schedule $\vec{\psi}^*$ (computed by Ref). Let

[7]www.cs.huji.ac.il/labs/parallel/workload/l_lpc/index.html
[8]www.cs.huji.ac.il/labs/parallel/workload/l_pik_iplex/index.html
[9]www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html
[10]www.cs.huji.ac.il/labs/parallel/workload/l_sharcnet/index.html

Table 8.1: The average delay (or the speed up) of jobs due to the unfairness of the algorithm $\Delta\psi/p_{tot}$ for different algorithms and different workloads. Each row is an average over 100 instances taken as parts of the original workload. The duration of the experiment is $5 \cdot 10^4$ time units.

	LPC-EGEE		PIK-IPLEX		SHARCNET-Whale		RICC	
	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.
RoundRobin	238	353	6	33	145	38	2839	357
Rand ($N = 15$)	8	21	0.014	0.01	6	6	162	187
DirectContr	5	11	0.02	0.15	10	7	537	303
FairShare	16	25	0.3	1.38	13	8	626	309
UtFairShare	16	25	0.3	1.38	38	67	515	284
CurrFairShare	87	106	0.3	1.58	145	80	1231	243

p_{tot} denote the total number of unit-size parts of the jobs completed in the fair schedule returned by Ref, $p_{tot} = \sum_{(s,p)\in\sigma^*:s\leq t_{end}} \min(p, t_{end} - s)$. We calculated the difference $\Delta\psi = \|\vec{\psi} - \vec{\psi}^*\| = \sum_{O^{(u)}}(\psi^{(u)} - \psi^{(u),*})$ and compared the values $\Delta\psi/p_{tot}$ for different algorithms. The value $\Delta\psi/p_{tot}$ is the measure of the fairness that has an intuitive interpretation. Since delaying each unit-size part of a job by one time moment decreases the utility of the job owner by one, the value $\Delta\psi/p_{tot}$ gives the average unjustified delay (or, unjustified speed-up) of a job due to the unfairness of the algorithm.

8.7.3 Results

We start with experiments on short sub-traces of the original workloads. We randomly selected the start time of the experiment t_{start} and set the end time to $t_{end} = t_{start} + 5 \cdot 10^4$. For each workload we run 100 experiments (on different periods of workloads of length $5 \cdot 10^4$). The average values of $\Delta\psi/p_{tot}$, and the standard deviations are presented in Table 8.1.

From this part of the experiments we conclude that: (i) The algorithm Rand is the most fair algorithm regarding the fairness by the Shapley Value; but Rand is also the second most computationally intensive algorithm (after Ref). (ii) All the other algorithms are about equally computationally efficient. The algorithm DirectContr is the most fair one among these other algorithms. (iii) The algorithm FairShare, which is the algorithm mostly used in real systems, is not much worse than DirectContr. (iv) Arbitrary scheduling algorithms like RoundRobin may result in unfair schedules. (v) The fairness of the algorithms may depend on the workload. In RICC the differences are much more visible than in PIK-IPLEX. Thus, although DirectContr and FairShare are usually comparable, on some workloads the difference is significant.

In the second series of experiments, we verified the effect of the duration of the simulated workload on the resulting fairness measure (the ratio $\Delta\psi/p_{tot}$). As we

Table 8.2: The average delay (or the speed up) of jobs due to the unfairness of the algorithm $\Delta\psi/p_{tot}$ for different algorithms and different workloads. Each row is an average over 100 instances taken as parts of the original workload. The duration of the experiment is $5 \cdot 10^5$.

	LPC-EGEE		PIK-IPLEX		SHARCNET-Whale		RICC	
	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.
RoundRobin	4511	6257	242	1420	404	1221	10850	13773
Rand ($N = 15$)	562	1670	1.3	7	26	158	771	1479
DirectContr	410	1083	0.2	1.4	60	204	1808	3397
FairShare	575	1404	2.3	12	94	307	2746	4070
UtFairShare	888	2101	1.2	5	120	344	4963	6080
CurrFairShare	1082	2091	2.2	11	180	805	5387	9083

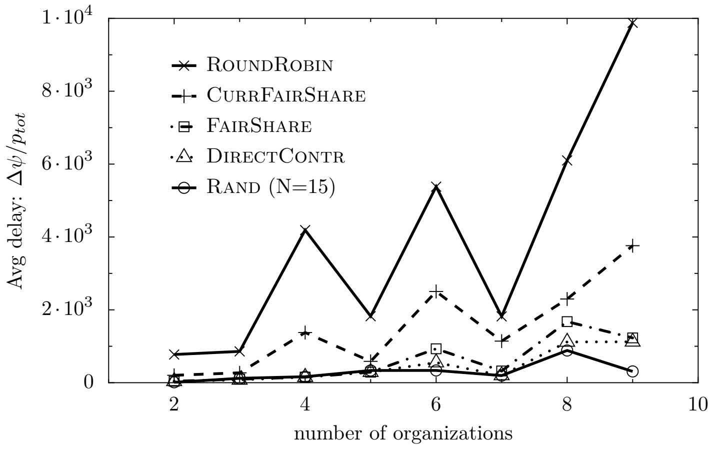

Figure 8.10: The effect of the number of the organizations on ratio $\Delta\psi/p_{tot}$.

changed the duration of the experiments from $5 \cdot 10^4$ to $5 \cdot 10^5$, we observed that the unfairness ratio $\Delta\psi/p_{tot}$ was increasing. The values of the ratio for $t_{end} - t_{start} = 5 \cdot 10^5$ are presented in Table 8.2. The relative quality of the algorithms is the same as in the previous case. Thus, all our previous conclusions hold. However, now all the algorithms are significantly less fair than the exact algorithm. Thus, in long-running systems the difference between the approaches becomes more important. If there are a few organizations, either the exact algorithm Ref or the randomized algorithm Rand should be used. In larger systems, when the computational cost of these algorithms is too high, DirectContr clearly outperforms FairShare.

Last, we verified the influence of the number of organizations on the ratio $\Delta\psi/p_{tot}$. The results from the experiments conducted on LPC-EGEE data set are presented in Figure 8.10. As the number of organizations increases, the unfairness ratio $\Delta\psi/p_{tot}$ grows and the difference between the algorithms is more significant. This confirms our previous conclusions.

8.8 Conclusions

In this chapter we defined the fairness of scheduling algorithms in terms of cooperative game theory, which allows to quantify the impact of an organization on the utilities of others. We presented a non-monetary model in which it is not required that each organization has accurate valuations of its jobs and resources. We show that classic utility functions may create incentives for workload manipulations. We thus proposed a strategy resilient utility function that can be thought of as per-organization throughput.

We analyzed the complexity of the fair scheduling problem. The general problem is NP-hard and hard to approximate. Nevertheless, the problem parameterized with the number of organizations is in FPT. Also, the FPT algorithm can be used as a reference for comparing the fairness of different algorithms on small instances (dozens of organizations). For a special case with unit-size jobs, we proposed an FPRAS. In our experiments, we showed that the FPRAS can be used as a heuristic algorithm; we also showed another efficient heuristic (DirectContr). The main conclusion from the experiments is that in multi-organizational systems, the distributive fairness idea used by the fair share algorithm does not result in truly-fair schedules; our heuristics better approximate the Shapley-fair schedules.

We, further, showed that every greedy algorithm achieves at least $\frac{3}{4}$-times as good resource utilization as the optimal algorithm. Since this result holds even if the durations and the pattern of incoming jobs are unknown and, under arbitrary underlying scheduling policy, the loss of resources utilization due to the fairness is upper-bounded by 25%.

Since we do not require the valuation of the jobs and we consider an on-line, non-clairvoyant scheduling, we believe the presented results have practical consequences for real-life job schedulers.

There are many natural questions for the future work. Although our approach is applicable to parallel jobs and to scheduling on related and unrelated machines, we yet do not know the resulting loss of global efficiency. Determining these bounds is an interesting open question. We suspect that in case of related and unrelated machines the loss of efficiency might be significant. In such case, the next natural question is too look for refinements of our algorithm that would allow to alleviate this problem. Another interesting direction is to explore other game-theoretic notions of fairness.

Chapter 9

We Are Impatient: Algorithms for Geographically Distributed Load Balancing with (Almost) Arbitrary Load Functions

In geographically-distributed systems, communication latencies are non-negligible. The perceived processing time of a request is thus composed of the time needed to route the request to the server and the true processing time. Once a request reaches a target server, the processing time depends on the total load of that server; this dependency is described by a load function. We consider a broad class of load functions; our only requirement is that they are convex and twice differentiable. In particular, our model can be applied to heterogeneous systems in which every server has a different load function. This approach allows us not only to generalize results for queuing theory, but also to use empirically-derived load functions, measured in a system under stress-testing.

Optimal assignment of requests to servers is communication-balanced, i.e., for each pair of non perfectly-balanced servers, the reduction of processing time resulting from moving a single request from the overloaded server to the underloaded one is smaller than the additional communication latency.

We present two algorithms, a centralized one and a decentralized one, for optimal load balancing. We prove bounds on the algorithms' convergence. To the best of our knowledge, these bounds were not known even for the special cases studied previously in queuing theory. Both algorithms are any-time algorithms. In the decentralized algorithm, each server balances the load with a randomly chosen peer. Such an algorithm is very robust to failures. We prove that the decentralized algorithm performs locally optimal steps.

Our work extends the currently known results by considering a broad class of load functions and by establishing theoretical bounds on the algorithms' convergence.

These results are especially applicable for servers whose characteristics under load cannot be described by standard mathematical models.

9.1 Introduction

We are impatient. An “immediate” reaction must take less than 100 ms [43]; a Google user is less willing to continue searching if the result page is slowed down by just 100-400 ms [36]; a web page loading faster by just 250 ms attracts more users than the competitor’s [185]. Few of us are thus willing to accept the 100-200ms Europe-US round-trip time; even fewer, the 300-400ms Europe-Asia round-trip time. Internet companies targeting global audiences must thus serve their contents locally. Google builds data centers all over the world; a company that doesn’t have Google scale uses a generic content delivery network (CDN) [103,233], such as Akamai [194,226,284] or spreads its content on multiple Amazon’s Web Service regions.

A geographically-distributed system is an abstract model of world-spanning networks. It is a network of interconnected servers processing requests. The system considers both communication (request routing) and computation (request handling). For example, apart from the communication latencies, a CDN handling complex content can no longer ignore the load imposed by requests on the servers. As another example, consider computational clouds, which are often distributed across multiple physical locations and, thus, must consider the network latency in addition to servers’ processing times.

Normally, each server handles only the requests issued by local users. For instance, a CDN node responds to queries incoming from the sub-network it is directly connected to (e.g., DNS redirections in Akamai [173,194,284]). However, load varies considerably. Typically, a service is more popular during the day than during the night (the daily usage cycle). Load also spikes during historic events, ranging from football finals to natural disasters. If a local server is overloaded, some requests might be handled faster on a remote, non-overloaded, server. The users will not notice the redirection if the remote server is “close” (the communication latency is small); but if the remote server is on another continent, the round-trip time may dominate the response time.

In this chapter we address the problem of balancing servers’ load taking into account the communication latency. We model the response time of a single server by a *load function*, i.e., a function that for a given load on a server (the number of requests handled by a server) returns the average processing time of requests. In particular, we continue the work of Liu et al. [181] and Tantawi and Towsley [287]. Liu et al. [181] showed the convergence of the algorithms for the particular load function that describes requests’ handling time in the queuing model [117]. Liu et al. [181] considered only the particular load function that describes requests’ handling time in the queuing model [117]. We use a broad class of functions that are continuous, convex and twice-differentiable (Section 9.2.1), which allows us to model not only queuing theory-based systems, but also a particular application with the response time measured empirically in a stress-test. Tantawi and Towsley [287] (who originally proposed the model), on the other hand, showed the algorithm for the case when the communication delay between each pair of nodes is the same.

We assume that the servers are connected by links with high bandwidth. Although some models (e.g., routing games [225]) consider limited bandwidth, our aim is to model servers connected by a dense network (such as the Internet), in which there are multiple cost-comparable routing paths between the servers. The communication time is thus dominated by the latency: a request is sent over a long distance with a finite speed. We assume that the latencies are known, as monitoring pairwise latencies is a well-studied problem [48,285]; if the latencies change due to, e.g., network problems, our optimization algorithms can be run again. On each link, the latency is constant, i.e., it does not vary with the number of sent requests. This assumption is consistent with the previous works on geographically distributed load balancing [13,44,118,122, 181,241] and validated by our experiments in Section 9.7.1.

Individual requests are small. Rather than an hour-long batch job, a request models, e.g., a single web page hit. Such assumption is often used [19,88,105,122, 181,241,287,294]. In particular, the continuous allocation of requests to servers in our model is analogous to the divisible load model with constant-cost communication (a special case of the affine cost model [19]) and multiple sources (multiple loads to be handled, [88,294]).

The problem of load balancing in geographically distributed systems has been already addressed, however it received limited attention. Liu et al. [181] shows the convergence of two algorithms for a particular load function from the queuing theory. Cardellini et al. [44] analyzes only simple redirection policies, like round robin, or redirection to least loaded server. Colajanni et al. [64] presents experimental evaluation of a round-robin-based algorithm for a similar problem. Minimizing the cost of energy due to the load balancing in geographically distributed systems is a similar problem considered in the literature [179,182,183].

Some papers analyze game-theoretic aspects of load balancing in geographically distributed systems [2,13,118,122,241]. These works use a similar model, but focus on capturing the economic relations between the participating entities.

The majority of the works in the literature on distributed load balancing ignore the communication costs [55,72,81,121,127,131,153,176,178] (some literature considers that during the communication delay the states of the servers may change, but do not consider communication delay as a cost that user perceive [81,131]). Our distributed algorithm is the extension of the diffusive load balancing [1,3,20]; it incorporates communication latencies into the classical diffusive load balancing algorithms.

Additionally to the problem of effective load balancing, we can optimize the choice of the locations for the servers [71,149,247]. The generic formulation of the placement problem, facility location problem [57] and k-median problem [145], have been extensively studied in the literature (in fact, our models from Part I are applicable for a variant of this problem as well).

The contributions of this chapter are the following.

1. We construct a centralized load-balancing algorithm that optimizes the response time up to a given (arbitrary small) distance to the optimal solution (Section 9.4). The algorithm has polynomial running time with respect to the total load of the system and the upper bounds of the derivatives of the load function.

2. We show a decentralized load-balancing algorithm (Section 9.5) in which pairs of servers balance their loads. We prove that the algorithm is optimal (there is no better algorithm that uses only a single pair of servers at each step). We also bound the number of pairwise exchanges required for convergence.

3. We do not use a particular load function; instead, we only require the load function to be continuous and twice-differentiable (Section 9.2.1). Thus we are able to model empirical response times of a particular application on a particular machine, but also to generalize (Section 9.2.2) Liu et al.'s [181] results on the queuing model.

4. We describe a new class of load functions that we obtain when the jobs arrive in batches. For this model we show that our load balancing problem is convex and, in particular, solvable in polynomial time. This result also indicates that the local optimization techniques can be applied to the problem.

5. We evaluate our distributed algorithm with the new proposed load function by simulation.

The algorithms we propose for the general model are suitable for real applications. These are any-time algorithms which means that we can stop them at any moment and get a complete, yet suboptimal, solution. Furthermore, the distributed algorithm is particularly suitable for distributed systems. It performs only pairwise optimizations (only two servers need to be available to perform a single optimization phase), which means that it is highly resilient to failures. It is also very simple and does not require additional complex protocols.

In this chapter we present the theoretical bounds, but we believe that the algorithms will have even better convergence in practice. Our experiments confirm this intuition for the case of distributed algorithm used for our batch model. The experimental evaluation for other load functions is the subject of our future work.

9.2 Preliminaries

In this section we first describe our mathematical model, and next we argue that our model is highly applicable. In particular, it generalizes two problems previously considered in the literature.

9.2.1 The Model

Servers, requests, relay fractions, current loads. The system consists of a set of m *servers* (processors) connected to the Internet. The i-th server has its *local (own) load* of size n_i consisting of small *requests*. The local load can be the current number of requests, the average number of requests, or the rate of incoming requests in the queuing model.

Each server can relay a part of its load to the other servers. We use a fractional model in which a *relay fraction* ρ_{ij} denotes the fraction of the i-th server's load that is sent (relayed) to the j-th server ($\forall_{i,j}\ \rho_{ij} \geq 0$ and $\forall_i\ \sum_{j=1}^{j=m} \rho_{ij} = 1$). Consequently, ρ_{ii} is the part of the i-th load that is kept on the i-th server. We consider two models. In the *single-hop model* the request can be sent over the network only once. In the *multiple-hop* model the requests can be routed between servers multiple times.[1] Let r_{ij} denote the size of the load that is sent from server i to server j. In the single-hop model, the requests transferred from i to j come only from the local load of the server i, thus:

$$r_{ij} = \rho_{ij} n_i. \tag{9.1}$$

In the multiple-hop model the requests come both from the local load of server i and from the loads of other servers that relay their requests to i. Thus r_{ij} is a solution of:

$$r_{ij} = \rho_{ij} \left(n_i + \sum_{k \neq i} r_{ki} \right). \tag{9.2}$$

The *(current) load* of the server i is the size of the load sent to i by all other servers, including i itself: $l_i = \sum_{j=1}^{m} r_{ji}$.

Load functions. Let f_i be a *load function* describing the average request's processing time on a server i as a function of i's load l_i (e.g.: if there are $l_i = 10$ requests and $f_i(10) = 7$, then on average it takes 7 time units to process each request). We assume f_i is known from a model or experimental evaluation; each server can have a different characteristic f_i (heterogeneous servers). The total processing time of the requests on a server i is equal to $h_i(l_i) = l_i f_i(l_i)$ (e.g., in the previous example it takes 70 time units to process all requests). In most of our results we use f_i instead of h_i to be consistent with [181].

[1]We point the analogy between the multiple-hop model and the Markov chain with the servers corresponding to states and relay fractions ρ_{ij} corresponding to the probabilities of changing states.

Instead of using a certain load function, we derive all our results for a broad class of load functions (see Section 9.2.2 on how to map existing results to our model). Let $l_{max,i}$ be the load that can be effectively handled on server i (beyond $l_{max,i}$ the server fails due to, e.g., trashing). Let $l_{max} = \max_i l_{max,i}$. Let $l_{tot} = \sum_i n_i$ be the total load in the system. We assume that the total load can be effectively handled, $\sum_i l_{max,i} \geq l_{tot}$ (otherwise, the system is clearly overloaded). We assume that the values $l_{max,i}$ are chosen so that $f_i(l_{max,i})$ are equal to each other (equal to the maximal allowed processing time of the request).

We assume that the load function f_i is bounded on the interval $[0, l_{max,i}]$. (If $l > l_{max,i}$ then we follow the convention that $f_i(l) = \infty$.) We assume f_i is non-decreasing as when the load increases, requests are not processed faster. We also assume that f_i is convex and twice-differentiable on the interval $[0; l_{max,i}]$ (functions that are not twice-differentiable can be well approximated by twice-differentiable functions). We assume that the first derivatives f_i' of all f_i are upper bounded by U_1 ($U_1 = \max_{i,l} f_i'(l)$), and that the second derivatives f_i'' are upper bounded by U_2 ($U_2 = \max_{i,l} f_i''(l)$). These assumptions are technical—every function that is defined on a closed interval can be upper-bounded by a constant (however the complexity of our algorithms depends on these constants).

Communication delays. If the request is sent over the network, the observed handling time is increased by the communication latency on the link. We denote the communication latency between the i-th and the j-th server as c_{ij} (with $c_{ii} = 0$). We assume that the requests are small, and so the communication delay of a single request does not depend on the amount of exchanged load (the same assumption was made in the previous works [19,88,105,122,181,241,294] and it is confirmed by the experiments we conducted on PlanetLab—Section 9.7.1). Thus, c_{ij} is a constant and not a function of the network load.

We assume *efficient ϵ-load processing*: for sufficiently small load $\epsilon \to 0$ the processing time is lower than the communication latency, so it is not profitable to send the requests over the network. Thus, for any two servers i and j we have:

$$h_i(\epsilon) < \epsilon c_{ij} + h_j(\epsilon). \tag{9.3}$$

We use an equivalent formulation of the above assumption (as $h_i(0) = h_j(0) = 0$):

$$\frac{h_i(\epsilon) - h_i(0)}{\epsilon} < c_{ij} + \frac{h_j(\epsilon) - h_j(0)}{\epsilon}. \tag{9.4}$$

Since the above must hold for every sufficiently small $\epsilon \to 0$, we get:

$$h_i'(0) < c_{ij} + h_j'(0) \Leftrightarrow f_i(0) < c_{ij} + f_j(0). \tag{9.5}$$

Problem formulation: The total processing time. We consider a system in which all requests have the same importance. Thus, the optimization goal is to minimize the total processing time of all requests $\sum C_i$, considering both the communication latencies and the requests' handling times on all servers, that is:

$$\sum C_i = \sum_{i=1}^{m} l_i f_i(l_i) + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} r_{ij}. \tag{9.6}$$

We formalize our problem in the following definition:

Definition 9.1 (Load balancing)**.** *Given m servers with initial loads $\{n_i, 0 \leq i \leq m\}$, load functions $\{f_i\}$ and communication delays $\{c_{ij} : 0 \leq i, j \leq m\}$ find ρ, a vector of fractions, that minimizes the total processing time of the requests, $\sum C_i$.*

We denote the optimal relay fractions by ρ^* and the load of the server i in the optimal solution as l_i^*.

9.2.2 Motivation

Since the assumptions about the load functions are moderate, our analysis is applicable to many systems. In order to apply our solutions one only needs to find the load functions f_i. In particular, our model generalizes the following models.

The Queuing Model

Our results generalize the results of Liu et al. [181] for the queuing model. In the queuing model, the initial load n_i corresponds to the rate of local requests at the i-th server. Every server i has a processing rate μ_i. According to the queuing theory, the dependency between load l (which is the effective rate of incoming requests) and the service time of the requests is described by $f_i(l) = \frac{1}{\mu_i - l}$ [117]. Its derivative, $f_i'(l) = \frac{1}{(\mu_i - l)^2}$ is upper bounded by $U_1 = \max_i \frac{1}{(\mu_i - l_{max,i})^2}$, and its second derivative $f_i''(l) = \frac{2}{(l-\mu_i)^3}$ is upper bounded by $U_2 = \max_i \frac{2}{(l_{max,i} - \mu_i)^3}$.

The Batch Model

Let us consider the model in which requests arrive in batches. Thus, in a given time moment all requests are available for execution. We consider the model in which requests are processed in the uniformly random order. The motivation for considering such model is the following. Since the number of requests is large, considering any particular order on the servers would increase the computational complexity. Also, since we are interested in the average processing time, the order of serving requests does not affect the optimization goal. We assume that the servers are uniform; each server i has a constant processing speed s_i.

In such a case, for each of the l_j request that are actually processed on j-th server, the expected processing time of each request is equal to $1/l_j \sum_1^{l_j} i/s_j \approx l_j/2s_j$. Thus, in this model the function f_i linearly depends on load $f_i(l) = \frac{l}{2s_i}$. Its derivative is constant, and thus upper bounded by $\frac{1}{2s_i}$. The second derivative is equal to 0.

Additionally to the results for general load functions, in Section 9.6 we present a load balancing algorithm for this specific batch model.

9.3 Characterization of the problem

In this section, we show various results that characterize the solutions in both the single-hop and the multiple-hop models. We will use these results in performance proofs in the next sections.

The relation between the single-hop model and the multiple-hop model is given by the two following lemmas.

Lemma 9.1. *If communication delays satisfy the triangle inequality (i.e., for every i, j, and k we have $c_{ij} < c_{ik} + c_{kj}$), then in the optimal solution there is no server i that both sends and receives the load, i.e., there is no server i such that $\exists_{j \neq i, k \neq i} ((\rho_{ij} > 0) \wedge (\rho_{ki} > 0))$*

Proof. For the sake of contradiction let us assume that there exist servers i, j and k, such that $\rho_{ij} > 0$ and $\rho_{ki} > 0$. Then, if we modify the relay fractions: $\rho_{ij} := \rho_{ij} - \min(\rho_{ij}, \rho_{ki})$, $\rho_{jk} := \rho_{jk} - \min(\rho_{ij}, \rho_{ki})$, and $\rho_{kj} := \rho_{kj} + \min(\rho_{ij}, \rho_{ki})$, then the loads l_i, l_j and l_k remain unchanged, but the communication delay is changed by:

$$\min(\rho_{ij}, \rho_{ki})(c_{kj} - c_{ki} - c_{ij}),$$

which is, by the triangle inequality, a negative value. This completes the proof. □

Proposition 9.2. *If communication delays satisfy the triangle inequality then the single-hop model and the multiple-hop model are equivalent.*

Proof. From Lemma 9.1 we get that in the optimal solution if $\rho_{ij} > 0$, then for every k we have $r_{ki} = 0$. Thus, $n_i + \sum_k r_{ki} = n_i$. □

We will also use the following simple observation.

Corollary 9.3. *The total processing time in the multiple-hop model is not higher than in the single-hop model.*

In the next two statements we recall two results given by Liu et al. [181] (these results were formulated for the general load functions). First, there exists an optimal solution in which only $(2m - 1)$ relay fractions ρ_{ij} are positive. This theorem makes our analysis more practical: the optimal load balancing can be achieved with sparse routing tables. However, we note that most of our results are also applicable to the case where every server is allowed to relay its requests only to a (small) subset of the servers; in such case we need to set the communication delays between the disallowed pairs of servers to infinity.

Theorem 9.4 (Liu et al. [181]). *In a single-hop model there exists an optimal solution in which at most $(2m-1)$ relay fractions ρ_{ij} have non-zero values.*

Second, all optimal solutions are equivalent.

Theorem 9.5 (Liu et al. [181]). *In all optimal solutions, every server i has the same load l_i^*.*

Finally, in the next series of lemmas we characterize the optimal solutions by linear equations. We will use this characterization in the analysis of the central algorithm.

Lemma 9.6. *In the multiple hop model, the optimal solution $\langle \rho_{ij}^* \rangle$ satisfies the following constraints:*

$$\forall_i \quad l_i^* \le l_{max,i} \tag{9.7}$$

$$\forall_{i,j} \quad \rho_{ij}^* \ge 0 \tag{9.8}$$

$$\forall_i \quad \sum_{j=1}^{m} \rho_{ij}^* = 1. \tag{9.9}$$

Proof. Inequality 9.7 ensures that the completion time of the requests is finite. Inequalities 9.8 and 9.9 state that the values of ρ_{ij}^* are valid relay fractions. □

Lemma 9.7. *In the multiple hop model, the optimal solution $\langle \rho_{ij}^* \rangle$ satisfies the following constraint:*

$$\forall_{i,j} \;\; f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} \ge f_i(l_i^*) + l_i^* f_i'(l_i^*) \tag{9.10}$$

Proof. For the sake of contradiction let us assume that $f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} < f_i(l_i^*) + l_i^* f_i'(l_i^*)$. Since we assumed that $f_j(0) + c_{ij} > f_i(0)$ (see Section 9.2.1), we infer that $l_i^* > 0$.

Next, we show that if $f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} < f_i(l_i^*) + l_i^* f_i'(l_i^*)$ and $l_i^* > 0$, then the server i can improve the total processing time of the requests $\sum C_i$ by relaying some more load to the j-th server (which will lead to a contradiction). Let us consider a function $F(\Delta r)$ that quantifies i's and j's contribution to $\sum C_i$ if Δr requests are additionally send from i to j (and also takes into account the additional communication latency $\Delta r c_{ij}$):

$$F(\Delta r) = (l_i^* - \Delta r) f_i(l_i^* - \Delta r) + (l_j^* + \Delta r) f_j(l_j^* + \Delta r) + \Delta r c_{ij}.$$

If $F(\Delta r) < F(0)$, then transferring extra Δr requests from i to j decreases $\sum C_i$ (thus leading to a better solution). We compute the derivative of F:

$$F'(\Delta r) = -f_i(l_i^* - \Delta r) - (l_i^* - \Delta r) f_i'(l_i^* - \Delta r) + f_j(l_j^* + \Delta r) + (l_j^* + \Delta r) f_j'(l_j^* + \Delta r) + c_{ij}.$$

Since we assumed that $f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} < f_i(l_i^*) + l_i^* f_i'(l_i^*)$, we get that:

$$F'(0) = -f_i(l_i^*) - l_i^* f_i'(l_i^*) + f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} < 0.$$

Since F' is differentiable, it is continuous; so there exists $\Delta r_0 > 0$ such that F' is negative on $[0; \Delta r_0]$, and thus F is decreasing on $[0; \Delta r_0]$. Consequently, $F(\Delta r_0) < F(0)$, which contradicts the optimality of $\langle \rho^*_{ij} \rangle$. □

Lemma 9.8. *In the multiple hop model, the optimal solution $\langle \rho^*_{ij} \rangle$ satisfies the following constraint:*

$$\forall_{i,j} \quad \textit{if } \rho^*_{ij} > 0 \textit{ then } f_j(l^*_j) + l^*_j f'_j(l^*_j) + c_{ij} \leq f_i(l^*_i) + l^*_i f'_i(l^*_i) \tag{9.11}$$

Proof. If $\rho^*_{ij} > 0$, then in the optimal solution i sends some requests to j. There are two possibilities. Either some of the transferred requests of i are processed on j, or j sends all of them further to another server j_2. Similarly, j_2 may process some of these requests or send them all further to j_3. Let $j, j_2, j_3, \dots, j_\ell$ be the sequence of servers such that every server from $j, j_2, j_3, \dots, j_{\ell-1}$ transfers all received requests of i to the next server in the sequence and j_ℓ processes some of them on its own.

First, we note that every server from $j, j_2, j_3, \dots, j_{\ell-1}$ has non-zero load. Indeed if this is not the case then let j_0 be the last server from the sequence which has load equal to 0. However we assumed that for sufficiently small load, it is faster to process it locally than to send it over the network to the next server j_k ($f_{j_0}(\epsilon) < f_k(\epsilon) + c_{j_0 k}$). This contradicts the optimality of the solution and shows that our observation is true.

Then, we take some requests processed on $j_{\ell-1}$ and swap them with the same number of requests owned by i, processed on j_ℓ. After this swap $j_{\ell-1}$ processes some requests of i; such a swap does not change $\sum C_i$. Next, we repeat the same procedure for $j_{\ell-1}$ and $j_{\ell-2}$; then $j_{\ell-2}$ and $j_{\ell-3}$; and so on. As a result, j processes some requests of i.

The next part of the proof is similar to the proof of Lemma 9.7. Let us consider the function $G(\Delta r)$ that quantifies i's and j's contribution to $\sum C_i$ if Δr requests are *moved back* from j to i (i.e., not sent from i to j):

$$G(\Delta r) = (l^*_i + \Delta r) f_i(l^*_i + \Delta r) + (l^*_j - \Delta r) f_j(l^*_j - \Delta r) - \Delta r c_{ij}.$$

If $G(\Delta r) < G(0)$, executing Δr requests on i (and not on j) reduces $\sum C_i$.

$G(\Delta r) = F(-\Delta r)$ (see the proof of Lemma 9.7). Thus, $G'(\Delta r) = -F'(\Delta r)$, and

$$G'(0) = -F'(0) = f_i(l^*_i) + l^*_i f'_i(l^*_i) - f_j(l^*_j) - l^*_j f'_j(l^*_j) - c_{ij}. \tag{9.12}$$

As l^*_i is optimal, $G'(0) \geq 0$, thus $f_i(l^*_i) + l^*_i f'_i(l^*_i) - f_j(l^*_j) - l^*_j f'_j(l^*_j) - c_{ij} \geq 0$, which proves the thesis. □

Lemma 9.9. *If some solution $\langle \rho_{ij} \rangle$ satisfies Inequalities 9.7, 9.8, 9.9, 9.10, and 9.11 then every server i under $\langle \rho_{ij} \rangle$ has the same load as in the optimal solution $\langle \rho^*_{ij} \rangle$.*

Proof. Let S_+ denote the set of servers that in ρ^* have greater or equal load than in ρ ($l^*_i \geq l_i$). For the sake of contradiction let us assume that S_+ is non-empty and that it contains at least one server i that in ρ^* has strictly greater load than in ρ ($l^*_i > l_i$).

Let $j \in S_+$; we will show that j in ρ^* can receive requests only from the servers from S_+. By definition of S_+, $l_j^* \geq l_j$. Consider a server i that in ρ^* relays some of its requests to j; we will show that $l_i^* \geq l_i$. Indeed, since $\rho_{ij}^* > 0$, from Inequality 9.11 we get that:

$$f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} \leq f_i(l_i^*) + l_i^* f_i'(l_i^*). \tag{9.13}$$

Since we assumed that $\langle \rho_{ij} \rangle$ satisfies Inequality 9.10, we get

$$f_j(l_j) + l_j f_j'(l_j) + c_{ij} \geq f_i(l_i) + l_i f_i'(l_i). \tag{9.14}$$

By combining these relations we get:

$$\begin{aligned} f_i(l_i^*) + l_i^* f_i'(l_i^*) &\geq f_j(l_j^*) + l_j^* f_j'(l_j^*) + c_{ij} && \text{from Eq. 9.13} \\ &\geq f_j(l_j) + l_j f_j'(l_j) + c_{ij} && \text{as } l_j^* \geq l_j \text{ and } f_j \text{ is convex} \\ &\geq f_i(l_i) + l_i f_i'(l_i) && \text{from Eq. 9.14.} \end{aligned}$$

Since f_i is convex, the function $f_i(l) + l f_i'(l)$ is non-decreasing (as the sum of two non-decreasing functions); thus $l_i^* \geq l_i$.

Similarly, we show that any $i \in S_+$ in ρ can send requests only to other S_+ servers. Consider a server j that in ρ receives requests from i.

$$\begin{aligned} f_j(l_j^*) + l_j^* f_j'(l_j^*) &\geq f_i(l_i^*) + l_i^* f_i'(l_i^*) - c_{ij} && \text{Eq. 9.10} \\ &\geq f_i(l_i) + l_i f_i'(l_i) - c_{ij} && \text{as } l_i^* \geq l_i \text{ and } f_i \text{ is convex} \\ &\geq f_j(l_j) + l_j f_j'(l_j) && \text{as } \rho_{ij} > 0, \text{ from Eq. 9.11.} \end{aligned}$$

Thus, $l_j^* \geq l_j$.

Let l_{in} be the total load sent in ρ to the servers from S_+ by the servers outside of S_+. Let l_{out} be the total load sent by the servers from S_+ in ρ to the servers outside of S_+. Analogously we define l_{in}^* and l_{out}^* for the state ρ^*. In the two previous paragraphs we showed that $l_{in}^* = 0$ and that $l_{out} = 0$. However, since the total load of the servers from S_+ is in ρ^* greater than in ρ, we get that:

$$l_{in}^* - l_{out}^* > l_{in} - l_{out}.$$

From which we get that: $-l_{out}^* > l_{in}$, i.e. $l_{in} + l_{out}^* < 0$, which leads to a contradiction as transfers l_{in} and l_{out}^* cannot be negative. □

9.4 An Approximate Centralized Algorithm

In this section we show a centralized algorithm for the multiple-hop model. As a consequence of Proposition 9.2, the results presented in this section also apply to the single-hop model with the communication delays satisfying the triangle inequality.

For the further analysis we introduce the notion of optimal network flow.

Definition 9.2 (Optimal network flow)**.** *The vector of relay fractions $\rho = \langle \rho_{ij} \rangle$ has an optimal network flow if and only if there is no $\rho' = \langle \rho'_{ij} \rangle$ such that every server in ρ' has the same load as in ρ and such that the total communication delay of the requests $\sum_{i,j} c_{ij} r'_{ij}$ in ρ' is lower than the total communication delay $\sum_{i,j} c_{ij} r_{ij}$ in ρ.*

The problem of finding the optimal network flow reduces to finding a minimum cost flow in an uncapacitated network. Indeed, in the problem of finding a minimum cost flow in an uncapacitated network we are given a graph with the cost of the arcs and demands (supplies) of the vertices. For each vertex i, b_i denotes the demand (if positive) or supply (if negative) of i. We look for the flow that satisfies demands and supplies and minimizes the total cost. To transform our problem of finding the optimal network flow to the above form, it suffices to set $b_i = l_i - n_i$. Thus our problem can be solved in time $O(m^3 \log m)$ [230]. Other distributed algorithms include the one of Goldberg et al. [113], and the asynchronous auction-based algorithms [24], with, e.g., the complexity of $O(m^3 \log(m) \log(\max_{i,j} c_{ij}))$.

The following theorem estimates how far is the current solution from the optimum based on the degree to which Inequality 9.10 is not satisfied. We use this theorem to prove the approximation ratio of our load balancing algorithm.

Theorem 9.10. *Let ρ be the vector of relay fractions satisfying Inequalities 9.7, 9.8, 9.9 and 9.11, and having an optimal network flow. Let Δ_{ij} quantify the extent to which Inequality 9.10 is not satisfied:*

$$\Delta_{ij} = \max(0, f_i(l_i) + l_i f'_i(l_i) - f_j(l_j) - l_j f'_j(l_j) - c_{ij}).$$

Let $\Delta = \max_{i,j} \Delta_{ij}$. Let e be the absolute error—the difference between $\sum C_i$ for solution ρ and for ρ^*, $e = \sum C_i(\rho) - \sum C_i(\rho^*)$. For the multiple-hop model and for the single-hop model satisfying the triangle inequality we get the following estimation:

$$e \leq l_{tot} m \Delta.$$

Proof. Let I be the problem instance. Let $\tilde{I}$ be a following instance: initial loads n_i in $\tilde{I}$ are the same as in I; communication delays c_{ij} are increased by Δ_{ij} ($\tilde{c_{ij}} := c_{ij} + \Delta_{ij}$). Let $\tilde{\rho^*}$ be the optimal solutions for $\tilde{I}$ in the multiple-hop model.

By Lemma 9.9, loads of servers in ρ are the same as in $\tilde{\rho^*}$, as ρ satisfies all inequalities for $\tilde{I}$. Let c^* and c denote the total communication delay of $\tilde{\rho^*}$ in $\tilde{I}$ and ρ in I, respectively. First, we show that $c^* \geq c$.

For the sake of contradiction, assume that $c^* < c$. We take the solution $\tilde{\rho^*}$ in $\tilde{I}$ and modify $\tilde{I}$ by decreasing each latency $\tilde{c_{ij}}$ by Δ_{ij}. We obtain instance I. During the process, we decreased (or did not change) communication delay over every link, and so we decreased (or did not change) the total communication delay. Thus, in I, $\tilde{\rho^*}$ has smaller communication delay than ρ. This contradicts the thesis assumption that ρ had in I the optimal network flow.

As $\tilde{I}$ has the same initial loads and not greater communication delay,

$$\sum C_i(\rho, I) \leq \sum C_i(\tilde{\rho^*}, \tilde{I}).$$

Based on Proposition 9.2, the same result holds if ρ is the solution in the single-hop model satisfying the triangle inequality.

We use a similar analysis to bound the processing time. In the multiple-hop model, if the network flow is optimal, then every request can be relayed at most m times. Thus, any solution transfers at most $l_{tot}m$ load. Thus, by increasing latencies from I to $\tilde{I}$ we increase the total communication delay of a solution by at most $l_{tot}m\Delta$. Taking the optimal solution ρ^*, we get:

$$\sum C_i(\rho^*, \tilde{I}) \leq l_{tot}m\Delta + \sum C_i(\rho^*, I).$$

As $\sum C_i(\tilde{\rho^*}, \tilde{I}) \leq \sum C_i(\rho^*, \tilde{I})$, by combining the two inequalities we get:

$$\sum C_i(\rho, I) \leq \sum C_i(\tilde{\rho^*}, \tilde{I}) \leq \sum C_i(\rho^*, \tilde{I}) \leq l_{tot}m\Delta + \sum C_i(\rho^*, I).$$

$\square$

The above estimations allow us to construct an approximation algorithm (the pseudo-code of the algorithm is presented in Figure 9.1). Lines 14 to 18 initialize the variables. In line 19 we build a certain arbitrary finite solution (i.e., a solution for which the load l_i on the i-th server does not exceed $l_{max,i}$). Next, in the while loop in line 22, we iteratively improve the solution. In each iteration we find a pair (i, j) with the maximal value of Δ_{ij}. Next we balance the servers i and j in line 7. Afterwards, it might be possible that the current solution does not satisfy Inequality 9.11. In lines 8 to 12 we fix the solution so that Inequality 9.11 holds.

The following Theorem shows that algorithm from Figure 9.1 achieves an arbitrary small absolute error e.

Theorem 9.11. *Let e_d be the desired absolute error for the centralized algorithm from Figure 9.1, and let e_i be the initial error. In the multiple-hop model the algorithm decreases the absolute error from e_i to e_d in time $O(\frac{{l_{tot}}^2 m^4 e_i(U_1 + l_{max}U_2)}{e_d^2})$.*

Proof. Let l_i and l_j be the loads of the servers i and j before the invocation of the `Adjust` function in line 7 of the algorithm from Figure 9.1. Let Δ_{ij} quantify how much Inequality 9.10 is not satisfied, $\Delta_{ij} = f_i(l_i) + l_i f_i'(l_i) - f_j(l_j) - l_j f_j'(l_j) - c_{ij}$. As in proof of Lemma 9.7, consider a function $F(\Delta r)$ that quantifies i's and j's contribution to $\sum C_i$ if Δr requests of are additionally send from i to j:

$$F(\Delta r) = (l_i - \Delta r)f_i(l_i - \Delta r) + (l_j + \Delta r)f_j(l_j + \Delta r) + \Delta r c_{ij}.$$

As previously, the derivative of F is:

$$F'(\Delta r) = -f_i(l_i - \Delta r) - (l_i - \Delta r)f_i'(l_i - \Delta r) + f_j(l_j + \Delta r) + (l_j + \Delta r)f_j'(l_j + \Delta r) + c_{ij}.$$

Notation:
e — the required absolute error of the algorithm.
$\boldsymbol{c_{ij}}$ — the communication delay between i-th and j-th server.
$\boldsymbol{l}[i]$ — the load of the i-th server in a current solution.
$\boldsymbol{r}[i,j]$ — the number of requests relayed between i-th and j-th server in a current solution.
`OptimizeNetworkFlow`(ρ, $\langle c_{ij}\rangle$) — builds an optimal network flow using algorithm of Orlin [230].

```
Adjust($i, j$):
    $\Delta r \leftarrow \mathrm{argmin}_{\Delta r}\left((l_i - \Delta r) f_i(l_i - \Delta r) + (l_j + \Delta r) f_j(l_j + \Delta r) + \Delta r c_{ij}\right)$;
    $l[i] \leftarrow l[i] - \Delta r$;
    $l[j] \leftarrow l[j] + \Delta r$;
    $r[i,j] \leftarrow r[i,j] + \Delta r$;
Improve($i, j$):
    Adjust ($i, j$);
    $servers$ ← sort servers topologically according to the order $\prec$: $i \prec j \iff \rho_{ij} > 0$;
    for $\ell$ in $servers$ do
        for $k \leftarrow 1$ to $m$ do
            if $r[k,\ell] > 0$ and $f_\ell(l^*_\ell) + l^*_\ell f'_\ell(l^*_\ell) + c_{k\ell} > f_k(l^*_k) + l^*_k f'_k(l^*_k)$ then
                AdjustBack ($\ell, k$);
Main($\langle c_{ij}\rangle$, $\langle n_i\rangle$, $\langle s_i\rangle$):
    for $i \leftarrow 1$ to $m$ do
        $l[i] \leftarrow n_i$;
        for $j \leftarrow 1$ to $m$ do
            $r[i,j] \leftarrow 0$;
        $r[i,i] \leftarrow n_i$;
    BuildAnyFiniteSolution() ;
    OptimizeNetworkFlow($r$, $\langle c_{ij}\rangle$);
    $(i,j) \leftarrow \mathrm{argmax}_{(i,j)} \Delta_{ij}$;
    while $\Delta_{ij} > \frac{e}{l_{tot} m}$ do
        $(i,j) \leftarrow \mathrm{argmax}_{(i,j)} \Delta_{ij}$;
        Improve($i$, $j$);
    OptimizeNetworkFlow($r$, $\langle c_{ij}\rangle$);
```

Figure 9.1: The approximation algorithm for multiple-hop model.

Thus, $F'(0) = -\Delta_{ij}$. The second derivative of F is equal to:

$$F''(\Delta r) = 2f_i'(l_i - \Delta r) + (l_i - \Delta r)f_i''(l_i - \Delta r) + 2f_j'(l_j + \Delta r) + (l_j + \Delta r)f_j''(l_j + \Delta r).$$

The second derivative is bounded by:

$$|F''(\Delta r)| \leq 4U_1 + 2l_{max}U_2. \tag{9.15}$$

For any function f with a derivative f' bounded on range $[x_0, x]$ by a constant $f'_{\max}$, the value $f(x)$ is upper-bounded by:

$$f(x) \leq f(x_0) + (x - x_0)f'_{\max}. \tag{9.16}$$

Using this fact, we upper-bound the first derivative by:

$$F'(\Delta r) \leq F'(0) + \Delta r(4U_1 + 2l_{max}U_2).$$

We use a particular value of the load difference $\Delta r_0 = \frac{\Delta_{ij}}{8U_1 + 4l_{max}U_2}$, getting that for $\Delta r \leq \Delta r_0$, we have:

$$\begin{aligned} F'(\Delta r) &\leq F'(0) + \Delta r(4U_1 + 2l_{max}U_2) \\ &\leq F'(0) + \Delta r_0(4U_1 + 2l_{max}U_2) \\ &\leq -\Delta_{ij} + \frac{\Delta_{ij}}{8U_1 + 4l_{max}U_2} \cdot (4U_1 + 2l_{max}U_2) \leq -\frac{1}{2}\Delta_{ij}. \end{aligned}$$

We can use Inequality 9.16 for a function F to lower-bound the reduction in $\sum C_i$ for Δr_0 as $F(0) - F(\Delta r_0)$:

$$F(0) - F(\Delta r_0) \geq \frac{1}{2}\Delta_{ij}|r_0 - 0| = \frac{\Delta_{ij}}{8U_1 + 4l_{max}U_2} \cdot \frac{1}{2}\Delta_{ij} = \frac{\Delta_{ij}^2}{16U_1 + 8l_{max}U_2}.$$

To conclude that `Adjust` function invoked in line 7 reduces the total processing time by at least $\frac{\Delta_{ij}^2}{16U_1 + 8l_{max}U_2}$, we still need ensure that the server i has enough (at least $\Delta r_0 = \frac{\Delta_{ij}}{8U_1 + 4l_{max}U_2}$) load to be transferred to j. However we recall that the value of F' in Δr_0 is negative, $F'(\Delta r_0) < -\frac{1}{2}\Delta_{ij} < 0$. This means that after transferring Δr_0 requests, sending more requests from i to j further reduces $\sum C_i$. Thus, if i's load would be lower than Δr_0, this would contradict the efficient ϵ-load processing assumption.

Also, every invocation of `AdjustBack` decreases the total completion time $\sum C_i$. Thus, after invocation of `Improve` the total completion time $\sum C_i$ is decreased by at least $\frac{\Delta_{ij}^2}{16U_1 + 8l_{max}U_2}$.

Each invocation of `Improve` preserves the following invariant: in the current solution Inequalities 9.8, 9.9 and 9.11 are satisfied. It is easy to see that Inequalities 9.8, 9.9 are satisfied. We will show that Inequality 9.11 holds too. Indeed, this is accomplished by a series of invocations of `AdjustBack` in line 12. Indeed, from the proof of Lemma 9.8, after invocation of the `Adjust` function for the servers i and j, these servers satisfy Inequality 9.11.

We also need to prove that the servers can be topologically sorted in line 8, that is that there is no such sequence of servers $i_1, \ldots, i_k$ that $r_{i_j i_{j+1}} > 0$ and $r_{i_k i_1} > 0$. For the sake of contradiction let us assume that there exists such a sequence. Let us consider the first invocation of `Adjust` in line 7 that creates such a sequence. Without loss of generality let us assume that such `Adjust` was invoked for the servers i_k and i_1. This means that before this invocation $\Delta_{i_k i_1} > 0$, and so $f_{i_k}(l_{i_k}) + l_{i_k} f'_{i_k}(l_{i_k}) > f_{i_1}(l_{i_1}) + l_{i_1} f'_{i_1}(l_{i_1})$. Since the invariant was satisfied before entering `Adjust` and since $r_{i_j i_{j+1}} > 0$, from Inequality 9.11 we infer that $f_{i_{j+1}}(l_{i_{j+1}}) + l_{i_{j+1}} f'_{i_{j+1}}(l_{i_{j+1}}) + c_{i_j, i_{j+1}} \leq f_{i_j}(l_{i_j}) + l_{i_j} f'_{i_j}(l_{i_j})$, and so that $f_{i_{j+1}}(l_{i_{j+1}}) + l_{i_{j+1}} f'_{i_{j+1}}(l_{i_{j+1}}) \leq f_{i_j}(l_{i_j}) + l_{i_j} f'_{i_j}(l_{i_j})$. Thus, we get contradiction:

$$f_{i_1}(l_{i_1}) + l_{i_1} f'_{i_1}(l_{i_1}) \geq f_{i_2}(l_{i_2}) + l_{i_2} f'_{i_2}(l_{i_2}) \geq \cdots \geq f_{i_k}(l_{i_k}) + l_{i_k} f'_{i_k}(l_{i_k}) \geq f_{i_1}(l_{i_1}) + l_{i_1} f'_{i_1}(l_{i_1}).$$

Which proves that the invariant is true.

If the algorithm finishes, then $\Delta < \frac{e_d}{l_{tot} m}$. After performing the last step of the algorithm the network flow is optimized and we can use Theorem 9.10 to infer that the error is at most e_d.

We estimate the number of iterations to decrease the absolute error from e_i to e_d. To this end, we estimated the decrease of the error after a single iteration of the while loop in line 22. The algorithm continues the last loop only when $\Delta \geq \frac{e_d}{l_{tot} m}$. Thus, after a single iteration of the loop the error decreases by at least $\frac{\Delta_{ij}^2}{16U_1 + 8l_{max}U_2} \geq \frac{e_d^2}{{l_{tot}}^2 m^2 (16U_1 + 8l_{max}U_2)}$. Thus, after $O(\frac{{l_{tot}}^2 m^2 e_i (U_1 + l_{max}U_2)}{e_d^2})$ iterations the error decreases to 0. Since every iteration of the loop has complexity $O(m^2)$, we get the thesis. □

Using a bound from Theorem 9.10 corresponding to the single-hop model we get the following analogous results.

Corollary 9.12. *If the communication delays satisfy the triangle inequality then the centralized algorithm from Figure 9.1 for the single-hop model decreases the absolute error from e_i to e_d in time $O(\frac{{l_{tot}}^2 m^4 e_i (U_1 + l_{max}U_2)}{e_d^2})$.*

For the relative (to the total load) errors $e_{i,r} = \frac{e_i}{l_{tot}}$, and $e_{d,r} = \frac{e_d}{l_{tot}}$, the centralized algorithm decreases $e_{i,r}$ to $e_{d,r}$ in time $O(\frac{l_{tot}(U_1 + l_{max}U_2) e_{i,r}}{e_{d,r}^2} m^4)$. Thus, we get the shortest runtime if l_{tot} is large and $e_{i,r}$ is small. If the initial error $e_{i,r}$ is large we can use a modified algorithm that performs `OptimizeNetworkFlow` in every iteration of the last "while" loop (line 22). Using a similar analysis as before we get the following bound.

Theorem 9.13. *The modified algorithm from Figure 9.1 that performs `OptimizeNetworkFlow` in every iteration of the last "while" loop (line 22) decreases the relative error $e_{i,r}$ by a multiplicative constant factor in time $O(\frac{l_{tot} m^5 \log m (U_1 + l_{max}U_2)}{e_{i,r}})$.*

Proof. The analysis is similar as in the proof of Theorem 9.11. Here however at the beginning of each loop the network flow is optimized. If the absolute error before the loop is equal to e, then from Theorem 9.10 we infer that $\Delta \geq \frac{e}{l_{tot}m}$. Thus, after a single iteration of the loop the error decreases by $\frac{\Delta^2}{16U_1+8l_{max}U_2} \geq \frac{e^2}{{l_{tot}}^2m^2(16U_1+8l_{max}U_2)}$, and so by the factor of:

$$\left(e - \frac{e^2}{{l_{tot}}^2m^2(16U_1+8l_{max}U_2)}\right)/e = \left(1 - \frac{e}{{l_{tot}}^2m^2(16U_1+8l_{max}U_2)}\right).$$

Taking the relative error $e_{i,r}$ as $\frac{e}{l_{tot}}$ we get that every iteration decreases the relative error by a constant factor $\left(1 - \frac{e_{i,r}}{l_{tot}m^2(16U_1+8l_{max}U_2)}\right)$. Thus, after $O(\frac{l_{tot}m^2(U_1+l_{max}U_2)}{e_{i,r}})$ iterations the error decreases by a constant factor. Since the complexity of every iteration of the loop is dominated by the algorithm optimizing the network flow (which has complexity $O(m^3 \log m)$), we get the thesis. □

Our centralized algorithm is an any-time algorithm. We can stop it at any time and get a so-far optimized solution.

9.5 Distributed algorithm

```
input: $(i, j)$ – the identifiers of the two servers
Data: $\forall_k\ r_{ki}$ – initialized to the number of requests owned by $k$ and relayed to
      $i$ ($\forall_k\ r_{kj}$ is defined analogously)
Result: The new values of $r_{ki}$ and $r_{kj}$
1  foreach $k$ do
2      $r_{ki} \leftarrow r_{ki} + r_{kj}$; $r_{kj} \leftarrow 0$;
3  $l_i \leftarrow \sum_k r_{ki}$ ; $l_j \leftarrow 0$ ;
4  $servers \leftarrow$ sort $[k]$ so that $c_{kj} - c_{ki} < c_{k'j} - c_{k'i} \implies k$ is before $k'$;
5  foreach $k \in servers$ do
6      $\Delta_{opt}r_{ikj} \leftarrow \mathrm{argmin}_{\Delta r}\left(h_i(l_i - \Delta r) + h_j(l_j + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj}\right)$ ;
7      $\Delta r_{ikj} \leftarrow \min\left(\Delta_{opt}r_{ikj}, r_{ki}\right)$ ;
8      if $\Delta r_{ikj} > 0$ then
9          $r_{ki} \leftarrow r_{ki} - \Delta r_{ikj}$; $r_{kj} \leftarrow r_{kj} + \Delta r_{ikj}$ ;
10         $l_i \leftarrow l_i - \Delta r_{ikj}$; $l_j \leftarrow l_j + \Delta r_{ikj}$ ;
11 return for each k: $r_{ki}$ and $r_{kj}$
```

Figure 9.2: CALCBESTTRANSFER(i, j)

The centralized algorithm requires the information about the whole network. The size of the input data is $O(m^2)$. A centralized algorithm has thus the following

```
partner ← random($m$);
relay (id, partner, calcBestTransfer($id$, $partner$));
```

Figure 9.3: Min-Error (MinE) algorithm performed by server id.

drawbacks: (i) collecting information about the whole network is time-consuming; moreover, loads and latencies may frequently change; (ii) the central algorithm is more vulnerable to failures. Motivated by these limitations we introduce a distributed algorithm for optimizing the query processing time.

Each server, i, keeps for each server, k, information about the number of requests that were relayed to i by k. The algorithm iteratively improves the solution—the i-th server in each step communicates with a random partner server, server j (Figure 9.3). The pair (i, j) locally optimizes the current solution by adjusting, for each k, r_{ki} and r_{kj} (the pseudo-code of the algorithm is presented in Figure 9.2). In the first loop of the algorithm from Figure 9.2, server i takes all the requests that were previously assigned to i and to j. Next, all the servers $[k]$ are sorted according to the ascending order of $(c_{kj} - c_{ki})$. The lower the value of $(c_{kj} - c_{ki})$, the less communication delay we need to pay for running requests of k on j rather than on i. Then, for each k, the loads are balanced between servers i and j. Theorem 9.14 shows that this decentralized algorithm optimally balances the loads on the servers i and j.

The idea of the algorithm is similar to the diffusive load balancing [1,3,20]; however there are substantial differences related to the fact that the machines are geographically distributed: (i) In each step no real requests are transferred between the servers; this process can be viewed as a simulation run to calculate the relay fractions ρ_{ij}. Once the fractions are calculated the requests are transferred and executed at the appropriate server. (ii) Each pair (i, j) of servers exchanges not only its own requests but the requests of all servers that relayed their requests either to i or to j. Since different servers may have different communication delays to i and to j, local balancing requires more care (algorithms from Figures 9.2 and 9.3).

The decentralized algorithm has the following properties: (i) The size of the input data is $O(m)$ for each server—communication latencies from a server to all other servers (and not for all pairs of servers). It is easy to measure these pairwise latencies (Section 9.1). The algorithm is also applicable to the case where we allow the server to relay its requests only to the certain subset of servers (we set the latencies to the servers outside of this subset to infinity). (ii) A single optimization step requires only two servers to be available (thus, it is very robust to failures). (iii) Any algorithm that in a single step involves only two servers cannot perform better (Theorem 9.14). (iv) The algorithm does not require any requests to be unnecessarily delegated—once the relay fractions are calculated the requests are sent over the network. (v) In each step of the algorithm we are able to estimate the distance between the current solution and the optimal one (Proposition 9.15).

9.5.1 Optimality

The following theorem shows the optimality of the decentralized algorithm.

Theorem 9.14. *After execution of balancing algorithm from Figure 9.2 for the pair of servers i and j, $\sum C_i$ cannot be further improved by sending the load of any servers between i and j (by adjusting r_{ki} and r_{kj} for any k).*

Proof. For the sake of simplicity of the presentation we prove that after performing algorithm from Figure 9.2, for any single server k we cannot improve the processing time $\sum C_i$ by moving any requests of k from i to j or from j to i. Similarly it can be proven that we cannot improve $\sum C_i$ by moving the requests of any *set* of the servers from i to j or from j to i.

Let us consider the total processing time function $h_i(l) = lf_i(l)$. Since f_i is non-decreasing and convex, h_i is convex. Indeed if $l > 0$, then:

$$h_i''(l) = (f_i(l) + lf_i'(l))' = 2f_i'(l) + lf_i''(l) > 0.$$

Now, let l be the total load on the servers i and j, $l = l_i + l_j$. Let us consider the function $P(\Delta r)$ describing the contribution in $\sum C_i$ of servers i and j as a function of load Δr processed on the server j (excluding communication):

$$\begin{aligned} P(\Delta r) &= (l - \Delta r)f_i(l - \Delta r) + \Delta r f_j(\Delta r) \\ &= h_i(l - \Delta r) + h_j(\Delta r). \end{aligned}$$

The function P is convex as well. Indeed:

$$P''(\Delta r) = h_i''(l - \Delta r) + h_j''(\Delta r) > 0.$$

Now, we show that after the second loop (Figure 9.2, lines 5-10) transferring any load from i to j, would not further decrease the total completion time $\sum C_i$. For the sake of contradiction let us assume that for some server k after the second loop some additional requests of k should be transferred from i to j. The second loop considers the servers in some particular order and in each iteration moves some load (possibly of size equal to 0) from i to j. Let I_k be the iteration of the second loop in which the algorithm considers the requests owned by k and tries to move some of them from i to j. Let $l - \Delta r_1$ and $l - \Delta r_2$ be the loads on the server i immediately after I_k and after the last iteration of the second loop, respectively. As no request is moved back from j to i, $\Delta r_2 \geq \Delta r_1$. We will use a function P_k:

$$P_k(\Delta r) = P(\Delta r_1 + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj}.$$

The function $P_k(\Delta r)$ returns the total processing time of i and j assuming the server i after iteration I_k sent additional Δr more requests of k to j (including the communication delay of these extra Δr requests).

Immediately after iteration I_k the algorithm could not improve the processing time of the requests by moving some requests owned by k from i to j. This is the consequence of one of two facts. Either all the requests of k are already on j, and so there are no requests of k to be moved (but in such case we know that when the whole loop is finished there are still no such requests, and thus we get a contradiction). Alternatively, the function P_k is increasing for some interval $[0, \epsilon]$ ($\epsilon > 0$). But then we infer that the function:

$$Q_k(\Delta r) = P(\Delta r_2 + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj},$$

is also increasing on $[0, \epsilon]$. Indeed:

$$Q'_k(\Delta r) = P'(\Delta r_2 + \Delta r) - c_{ki} + c_{kj} \geq P'(\Delta r_1 + \Delta r) - c_{ki} + c_{kj} = P'_k(\Delta r),$$

Since Q_k is convex (because P is convex) we get that Q_k is increasing not only on $[0, \epsilon]$, but also for any positive Δr. Thus, it is not possible to improve the total completion time by sending the requests of k from i to j after the whole loop is finished. This gives a contradiction.

Second, we will show that when the algorithm finishes no requests should be transferred back from j to i either. Again, for the sake of contradiction let us assume that for some server k after the second loop (Figure 9.2, lines 5-10) some requests of k should be transferred back from j to i. Let I_k be the iteration of the second in which the algorithm considers the requests owned by k. Let us take the last iteration $I_{s_{last}}$ of the second loop in which the requests of some server s_{last} were transferred from i to j. Let $l - \Delta r_3$ be the load on i after $I_{s_{last}}$. After $I_{s_{last}}$ no requests of s_{last} should be transferred back from j to i (argmin in line 6). Thus, for some $\epsilon > 0$ the function R_k:

$$R_k(\Delta r) = P(\Delta r_3 - \Delta r) + \Delta r c_{s_{last}i} - \Delta r c_{s_{last}j}$$

is increasing on $[0, \epsilon]$. Since the servers are ordered by decreasing latency differences $(c_{ki} - c_{kj})$ (increasing latency differences $(c_{kj} - c_{ki})$), we get $c_{s_{last}i} - c_{s_{last}j} \leq c_{ki} - c_{kj}$, and so that the function:

$$S_k(\Delta r) = P(\Delta r_3 - \Delta r) + \Delta r c_{ki} - \Delta r c_{kj}$$

is also increasing on $[0, \epsilon]$. Since S_k is convex we see that it is increasing or any positive Δr, and thus we get the contradiction. This completes the proof. $\square$

9.5.2 Convergence

The following analysis bounds the error of the distributed algorithm as a function of the servers' loads. When running the algorithm, this result can be used to assess whether it is still profitable to continue. As the corollary of our analysis, we will show the convergence of the distributed algorithm.

In proofs, we will use an *error graph* that quantifies the difference of loads between the current and the optimal solution.

Definition 9.3 (Error graph). *Let ρ be the snapshot (the current solution) at some moment of execution of the distributed algorithm. Let ρ^* be the optimal solution (if there are multiple optimal solutions with the same $\sum C_i$, ρ^* is the closest solution to ρ in the Manhattan metric). $(P, \Delta\rho)$ is a weighted, directed* error graph *with multiple edges. The vertices in the error graph correspond to the servers; $\Delta\rho[i][j][k]$ is a weight of the edge $i \to j$ with a label k. The weight indicates the number of requests owned by k that should be executed on j instead of i in order to reach ρ^* from ρ.*

The error graphs are not unique. For instance, to move x requests owned by k from i to j we can move them directly, or through some other server ℓ. In our analysis, we will assume that the total weight of the edges in the error graph $\sum_{i,j,k} \Delta\rho[i][j][k]$ is minimal, that is that there is no i, j, k, and ℓ, such that $\Delta\rho[i][\ell][k] > 0$ and $\Delta\rho[\ell][j][k] > 0$.

Let $succ(i) = \{j : \exists_k \Delta\rho[i][j][k] > 0\}$ denote the set of (immediate) successors of server i in the error graph; $prec(i) = \{j : \exists_k \Delta\rho[j][i][k] > 0\}$ denotes the set of (immediate) predecessors of i.

We will also use a notion of a *negative cycle*: a sequence of servers in the error graph that essentially redirect some of their requests to one another.

Definition 9.4 (Negative cycle). *In the error graph, a* negative cycle *is a sequence of servers $i_1, i_2, \ldots, i_n$ and labels $k_1, k_2, \ldots, k_n$ such that:*

1. *$i_1 = i_n$; (the sequence is a cycle)*
2. *$\forall_{j \in \{1, \ldots n-1\}}$ $\Delta\rho[i_j][i_{j+1}][k_j] > 0$; (for each pair there is an edge in the error graph)*
3. *$\sum_{j=1}^{n-1} c_{k_j i_{j+1}} < \sum_{j=1}^{n-1} c_{k_j i_j}$ (the transfer in the circle $i_j \xrightarrow{k_j} i_{j+1}$ decreases communication delay).*

A current solution that results in an error graph without negative cycles has smaller processing time: After dismantling a negative cycle, loads on servers remain the same, but the communication time is reduced. Thus, if the current solution has an optimal network flow, then there are no negative cycles in the error graph.

Analogously, we define *positive cycles*. The only difference is that instead in the third inequality we require $\sum_{j=1}^{n-1} c_{k_j i_{j+1}} \geq \sum_{j=1}^{n-1} c_{k_j i_j}$. Thus, when an error graph has a positive cycle, the current solution is better than if the cycle would be dismantled.

We start by bounding the load imbalance when there are no negative cycles.

Lemma 9.15. *Let $impr_{pq}$ be the improvement of the total processing time $\sum C_i$ after balancing servers p and q by algorithm from Figure 9.2. Let l_i be the load of server i in the current state; and l_i^* be the optimal load. If the error graph $\Delta\rho$ has no negative cycles, then for every positive ϵ the following estimation holds:*

$$f_i(l_i) - f_i(l_i^*) \leq \frac{6U_1 + 3l_{max}U_2}{\epsilon} \max_{pq} impr_{pq} + m\epsilon.$$

Proof. First we show that there is no cycle (positive nor negative) in the error graph. By contradiction let us assume that there is a cycle: $i_1, \dots, i_{n-1}, i_n$ (with $i_1 = i_n$) with labels $k_1, k_2, \dots, k_n$. Because, we assumed the error graph has no negative cycle, we have: $\sum_{j=1}^{n-1}(c_{k_j i_{j+1}} - c_{k_j i_j}) \geq 0$. Now, let $\Delta\rho_{min} = \min_{j \in \{1,\dots,n-1\}}(\rho[i_j][i_{j+1}][k_j])$ be the minimal load on the cycle. If we reduce the number of requests sent on each edge of the cycle:

$$\Delta\rho[i_j][i_{j+1}][k_j] := \Delta\rho[i_j][i_{j+1}][k_j] - \Delta\rho_{min}$$

then the load of the servers $i_j, j \in \{1, \dots, n-1\}$ will not change. Additionally, the latencies decrease by $\rho_{min}\left(\sum_{j=1}^{n-1} c_{k_j i_{j+1}} - c_{k_j i_j})\right)$ which is at least equal to 0. Thus, we get a new optimal solution which is closer to ρ in Manhattan metric, which contradicts that ρ^* is optimal.

In the remaining part of the proof, we show how to bound the difference $|f_i(l_i) - f_i(l_i^*)|$. Consider a server i for which $l_i > l_i^*$, and a server $j \in succ(i)$. We define as Δr_{ij}^{ϵ} the load that in the current state ρ should be transferred between i and j so that after this transfer, moving any Δr more load owned by any k between i to j would be either impossible or would not improve $\sum C_i$ by more than $\epsilon\Delta r$. Intuitively, after moving Δr_{ij}^{ϵ}, we won't be able to further "significantly" reduce $\sum C_i$: further reductions depend on the moved load (Δr), but the rate of the improvement is lower than ϵ. This move resembles algorithm from Figure 9.2: i.e., algorithm from Figure 9.2 moves Δr_{ij}^{ϵ} for $\epsilon = 0$.

Let $\tilde{\rho}$ denote a state obtained from ρ when i moves to j exactly Δr_{ij}^{ϵ} requests. Let $\tilde{l_i}$ and $\tilde{l_j}$ denote the loads of the servers i and j in $\tilde{\rho}$, respectively. We define $H_k(\Delta r)$ as the change of $\sum C_i$ resulting from moving additional Δr requests produced by k from i to j:

$$H_k(\Delta r) = h_i(\tilde{l_i} - \Delta r) + h_j(\tilde{l_j} + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj}.$$

State $\tilde{\rho}$ satisfies one of the following conditions for each k (consider Δr as a small, positive number):

1. The servers i and j are ϵ-balanced, thus moving Δr requests from i to j would not reduce $\sum C_i$ by more than $\epsilon\Delta r$. In such case we can bound the derivative of H_k:

$$|H_k'(0)| \leq \epsilon, \tag{9.17}$$

2. Or, moving Δr requests from i to j would decrease $\sum C_i$ by more than $\epsilon\Delta r$, but there are no requests of k on i:

$$H_k'(0) < -\epsilon \quad \text{and} \quad \tilde{r_{ki}} = 0, \tag{9.18}$$

3. Or, moving Δr requests back from j to i would decrease $\sum C_i$ by more than $\epsilon \Delta r$, but there are no requests of k to be moved back:

$$H'_k(0) > \epsilon \quad \text{and} \quad \tilde{r_{kj}} = 0. \tag{9.19}$$

In the optimal solution, for any k, no k's requests should be moved between i and j. We define $G_k(\Delta r)$ similarly to H_k, but for the optimal loads:

$$G_k(\Delta r) = h_i(l^*_i - \Delta r) + h_j(l^*_j + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj}. \tag{9.20}$$

By the same reasoning, at least one of the three following inequalities holds:

$$G'_k(0) = 0, \qquad \text{or} \tag{9.21}$$
$$G'_k(0) < 0 \quad \text{and} \quad r^*_{ki} = 0, \quad \text{or} \tag{9.22}$$
$$G'_k(0) > 0 \quad \text{and} \quad r^*_{kj} = 0. \tag{9.23}$$

We consider two cases on the sum of weights between i and j in the error graph. Either (1) in the error graph, i sends to j at most Δr^ϵ_{ij} requests ($\sum_k \Delta\rho[i][j][k] \leq \Delta r^\epsilon_{ij}$); or (2) $\sum_k \Delta\rho[i][j][k] > \Delta r^\epsilon_{ij}$. We further analyze (2). Since Δr^ϵ_{ij} is the total load transferred from i to j in ρ to get $\tilde{\rho}$, there must exist a k such that $\rho[i][j][k] > \Delta r^\epsilon_{ij}(k)$ (from i to j more k's requests are moved in the error graph than in ρ to get $\tilde{\rho}$). We show that $\tilde{r_{ki}} > 0$ by contradiction. If $\tilde{r_{ki}} = 0$ (in $\tilde{\rho}$, no k's requests are processed on i), then $r_{ki} = \Delta r^\epsilon_{ij}(k)$ (all k's requests were moved to j in ρ to get $\tilde{\rho}$). As $\rho[i][j][k] \leq r_{ki}$ (the error graph does not transfer more requests than available), $\rho[i][j][k] \leq \Delta r^\epsilon_{ij}(k)$, which contradicts $\rho[i][j][k] > \Delta r^\epsilon_{ij}(k)$. As $\tilde{r_{ki}} > 0$, Ineq. 9.17 or 9.19 holds (Ineq. 9.18 does not hold), thus $H'_k(0) \geq -\epsilon$. As $\rho[i][j][k] > \Delta r^\epsilon_{ij}(k)$, $\rho[i][j][k] > 0$, thus, $r^*_{kj} > 0$, so Ineq. 9.21 or 9.22 holds (Ineq. 9.23 does not hold), so $G'_k(0) \leq 0$.

$$G'_k(0) \leq 0 \Leftrightarrow -h'_i(l^*_i) + h'_j(l^*_j) - c_{ki} + c_{kj} \leq 0$$
$$H'_k(0) \geq -\epsilon \Leftrightarrow -h'_i(\tilde{l_i}) + h'_j(\tilde{l_j}) - c_{ki} + c_{kj} \geq -\epsilon$$

Combining the above inequalities,

$$-h'_i(\tilde{l_i}) + h'_j(\tilde{l_j}) + \epsilon \geq -h'_i(l^*_i) + h'_j(l^*_j)$$

Or equivalently:

$$h'_i(\tilde{l_i}) - h'_i(l^*_i) \leq h'_j(\tilde{l_j}) - h'_j(l^*_j) + \epsilon \tag{9.24}$$

We can further expand the above inequality for j and its successors (and each expansion is applied on state ρ), and so on towards the end of the error graph (we proved there are no cycles), until we get that for some p and its successor q the condition of the case (2) does not hold, so (1) must hold ($\sum_k \Delta\rho[p][q][k] \leq \Delta r^{\epsilon}_{pq}$, or equivalently $|l_p - l^*_p| \leq \Delta r^{\epsilon}_{pq}$). Analogously to $\tilde{\rho}$ we define $\tilde{\tilde{\rho}}$ as the state in which p moves to q load Δr^{ϵ}_{pq}. Thus, we have:

$$h'_i(\tilde{l}_i) - h'_i(l^*_i) \leq h'_p(\tilde{\tilde{l}}_p) - h'_p(l^*_p) + m\epsilon \text{ (Ineq. 9.24 expanded for at most } m \text{ successors)} \tag{9.25}$$

$$|l_p - l^*_p| \leq \Delta r^{\epsilon}_{pq} \text{ (condition (1))} \tag{9.26}$$

From the definition of $\tilde{\tilde{\rho}}$ we have:

$$|\tilde{\tilde{l}}_p - l_p| \leq \Delta r^{\epsilon}_{pq} \tag{9.27}$$

Combining Inequalities 9.26 and 9.27 we get:

$$|\tilde{\tilde{l}}_p - l^*_p| \leq 2\Delta r^{\epsilon}_{pq} \tag{9.28}$$

We bound the second derivative of h''_p:

$$h''_p(l) = (f_p(l) + lf'_p(l))' = 2f'_p(l) + lf''_p(l) \leq 2U_1 + l_{max}U_2 = U_3. \tag{9.29}$$

With the above observations, and using the bound from Inequality 9.16 we get:

$$\begin{aligned}
h'_i(l_i) - h'_i(l^*_i) &\leq h'_i(\tilde{l}_i) - h'_i(l^*_i) + U_3|\tilde{l}_i - l_i| && \text{Ineq. 9.16} \\
&\leq h'_i(\tilde{l}_i) - h'_i(l^*_i) + U_3\Delta r^{\epsilon}_{ij} && \text{condition (1)} \\
&\leq h'_p(\tilde{\tilde{l}}_p) - h'_p(l^*_p) + U_3\Delta r^{\epsilon}_{ij} + m\epsilon && \text{Ineq. 9.25} \\
&\leq U_3|\tilde{\tilde{l}}_p - l^*_p| + U_3\Delta r^{\epsilon}_{ij} + m\epsilon && \text{Ineq. 9.16} \\
&\leq 2U_3\Delta r^{\epsilon}_{pq} + U_3\Delta r^{\epsilon}_{ij} + m\epsilon && \text{Ineq. 9.28} \\
&\leq (6U_1 + 3l_{max}U_2)\max_{pq}\Delta r^{\epsilon}_{pq} + m\epsilon && \text{Ineq. 9.29.}
\end{aligned}$$

As $h'_i(l_i) = f_i(l_i) + l_i f'_i(l_i)$, from $l_i \geq l^*_i$, we get that:

$$\begin{aligned}
f_i(l_i) - f_i(l^*_i) &\leq h'_i(l_i) - h'_i(l^*_i) \\
&\leq (6U_1 + 3l_{max}U_2)\max_{pq}\Delta r^{\epsilon}_{pq} + m\epsilon.
\end{aligned}$$

We can get the same results for the server i such that $l_i \leq l^*_i$, by expanding the inequalities towards the predecessors of i.

We now relate Δr^{ϵ}_{pq} with $impr_{pq}$, the result of balancing algorithm from Figure 9.2. The balancing algorithm stops when no further improvement is possible, thus the load moved by the algorithm is at least Δr^{ϵ}_{pq} for any ϵ. By definition of Δr^{ϵ}_{pq}, moving Δr^{ϵ}_{pq} load improves $\sum C_i$ by at least $\epsilon\Delta r^{\epsilon}_{pq}$. Thus, $impr_{ij} \geq \epsilon\Delta r^{\epsilon}_{ij}$. This completes the proof. □

As a result, we get the following corollary.

Corollary 9.16. *If the network flow in the current solution ρ is optimal, then the absolute error e is bounded:*

$$e \leq l_{tot} \frac{6U_1 + 3l_{max}U_2}{\epsilon} \max_{pq} impr_{pq} + ml_{tot}\epsilon$$

Proof. The value $f_i(l_i)$ denotes the average processing time of a request on the i-th server. For every server i the average processing time of every request on i in ρ is by at most $\frac{6U_1+3l_{max}U_2}{\epsilon} \max_{pq} impr_{pq} + m\epsilon$ greater than in ρ^*. Thus, since there are l_{tot} requests in total, we get the thesis. □

We can use Lemma 9.15 directly to estimate the error during the optimization if we run a distributed negative cycle removal algorithm (e.g. [24,113]). However, this result is even more powerful when applied together with the lemmas below, as it will allow to bound the speed of convergence of the algorithm (even without additional protocols optimizing the network flow). Now, we show how to bound the impact of negative cycles.

Lemma 9.17. *For every $\epsilon > 0$, removing the negative cycles improves the total processing time $\sum C_i$ of the solution by at most:*

$$\epsilon l_{tot} + 2m \sum_{ij} impr_{ij} + \frac{16U_1 + 8l_{max}U_2}{\epsilon} \max_{ij} impr_{ij} l_{tot}.$$

Proof. In the analysis we will use a function $F_{i,j,k}$ defined as H_k in the proof of Lemma 9.15 (here, we will use indices i, j)

$$\begin{aligned} F_{i,j,k}(\Delta r) &= h_i(l_i - \Delta r) + h_j(l_j + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj} \\ &= (l_i - \Delta r) f_i(l_i - \Delta r) + (l_j + \Delta r) f_j(l_j + \Delta r) - \Delta r c_{ki} + \Delta r c_{kj}. \end{aligned}$$

We will analyze a procedure that removes negative cycles one by one.

First, we prove that we can remove all the negative cycles by only considering the cycles that satisfy the *one-way transfers* property: if in a cycle there is a transfer $i \to j$, then in no cycle there is a transfer $j \to i$. Indeed, consider the set of all cycles. We do the following procedure:

1. If there are two negative cycles C and $\tilde{C}$ with a common edge (i, j), such that in the first cycle load l is transferred from i to j and in the second load $\tilde{l} \leq l$ is transferred back from j to i then we split the first cycle C into two cycles C_1 and C_2 such that C_1 transfers $\tilde{l}$ and C_2 transfers $l - \tilde{l}$. Next, we merge C_1 and $\tilde{C}$ into one negative cycle that does not contain edge (i, j).

2. If a single cycle transfers load first from i to j, and then from j to i then we break the cycle at the edge $i \leftrightarrow j$ into two cycles (without edges between i and j).

Let us note that each of the two above steps does not increase the total load transferred on the cycles. For a given error graph, there are many possible ways (sets of cycles) to express a non-optimal flow as a sum of negative cycles. We will consider a set of cycles with the smallest total load (sum of loads transferred over all edges of all cycles). In this set, we will remove individual cycles sequentially in an arbitrary order.

In this sequence of cycles with the smallest load, every request is transferred at most once. Indeed, if this is not the case, a request was transferred through adjacent edges e_1 and e_2. Thus, among the cycles we consider there are two negative cycles, such that the first one contains the edge $e_1 = (i, j)$ and the second one contains $e_2 = (j, \ell)$. (a single request cannot be transferred $i \to j \to \ell$ in a single cycle, because by sending $i \to \ell$ we would get a cycle with a smaller load). Also, between i and j and between j and ℓ requests of the same server k are transferred. Let $e_3 = (\ell, p)$ be the edge adjacent to e_2 in the second cycle. If in the first cycle we send requests from i to ℓ and in the second from j to p, we would obtain two cycles that transfer an equivalent load (each server has the same requests) and have smaller total transfer, a contradiction.

Let us consider a state ρ with a negative cycle c, that is the sequence of servers $i_1, i_2, \ldots, i_n$ and labels $k_1, k_2, \ldots, k_n$. Let us assume that in a negative cycle c the load Δr is carried on. After removing the cycle c, $\sum C_i$ is improved by I_c:

$$\begin{aligned}
0 < I_c &= \sum_j \Delta r \left(c_{k_j i_j} - c_{k_j i_{j+1}}\right) \\
&= \Delta r \sum_j \left(h'_{i_j}(l_{i_j}) - h'_{i_{j+1}}(l_{i_{j+1}}) + c_{k_j i_j} - c_{k_j i_{j+1}}\right) \quad \textit{the sequence is a cycle} \\
&= \Delta r \sum_j -F'_{i_j, i_{j+1}, k_j}(0).
\end{aligned}$$

Let us distribute among the edges the cost of all negative cycles (the cost, i.e., the increase in $\sum C_i$ resulting from the cycles). For removing a single cycle with load Δr, from the above inequality we assign the cost $-F'_{i,j,k}(0)\Delta r$ to the labeled edge $i \xrightarrow{k} j$. As every request is sent over a single edge at most once, the total cost assigned to the labeled edge $i \xrightarrow{k} j$ will be at most $-F'_{i,j,k}(0) r_{ki}$.

In the further part of the proof we will estimate $-F'_{i,j,k}(0)$. First, we bound the second derivative of F as in Eq. 9.15 in the proof of Theorem 9.11:

$$|F''_{i,j,k}(\Delta r)| \leq 4U_1 + 2l_{max}U_2.$$

Next, we consider two cases: (1) $F'_{i,j,k}(0) \geq -\epsilon$, and (2) $F'_{i,j,k}(0) < -\epsilon$. If (1) is the case then the total cost associated with $i \xrightarrow{k} j$ is at most ϵr_{ki}. We further analyze (2). Let us take $\Delta r_0 = \min(r_{ki}, \frac{\epsilon}{8U_1 + 4l_{max}U_2})$. From Inequality 9.16 we get that:

$$F'_{i,j,k}(\Delta r_0) \leq F'_{i,j,k}(0) + \Delta r_0 (4U_1 + 2l_{max}U_2)$$

$$\leq F'_{i,j,k}(0) + \frac{\epsilon}{8U_1 + 4l_{max}U_2}(4U_1 + 2l_{max}U_2)$$
$$\leq F'_{i,j,k}(0) - \frac{F'_{i,j,k}(0)}{2} \leq \frac{F'_{i,j,k}(0)}{2}.$$

Thus (again from Ineq. 9.16 but applied for F) we get that:

$$F_{i,j,k}(0) - F_{i,j,k}(\Delta r_0) \geq \Delta r_0 \frac{-F'_{i,j,k}(0)}{2}.$$

Since $\Delta r_0 \leq r_{ki}$ (there are at least Δr_0 requests of k on i), we infer that the balancing algorithm from Figure 9.2 would achieve improvement $impr_{ij}$ lower-bounded by:

$$impr_{ij} \geq F_{i,j,k}(0) - F_{i,j,k}(\Delta r_0) \geq -F'_{i,j,k}(0)\frac{\Delta r_0}{2}.$$

Further, we consider two sub-cases. If (2a) $\Delta r_0 = r_{ki}$ then the total cost associated with $i \xrightarrow{k} j$ is at most $2impr_{ij}$. (2b) Otherwise ($\Delta r_0 = \frac{\epsilon}{8U_1+4l_{max}U_2}$), we have $impr_{ij}\frac{16U_1+8l_{max}U_2}{\epsilon} \geq -F'_{i,j,k}(0)$. Since $\sum_{i,k} r_{ki} = l_{tot}$, we get that the total cost associated with all edges is at most:

$$\begin{aligned} &\epsilon l_{tot} + && \textit{condition (1)} \\ &2m\sum_{ij} impr_{ij} + && \textit{condition (2a)} \\ &\frac{16U_1 + 8l_{max}U_2}{\epsilon}\max_{ij} impr_{ij} l_{tot}. && \textit{condition (2b)} \end{aligned}$$

Thus, we get the thesis. □

Finally we get the following estimation.

Theorem 9.18. *Let $impr_{ij}$ be the improvement of the total processing time $\sum C_i$ after balancing servers i and j through algorithm from Figure 9.2. Let e be the absolute error in $\sum C_i$ (the difference between $\sum C_i$ in the current and the optimal state). For every $\epsilon > 0$, we have:*

$$e \leq 2m\sum_{ij} impr_{ij} + \max_{ij} impr_{ij}\frac{22U_1 + 11l_{max}U_2}{\epsilon}l_{tot} + (m+1)l_{tot}\epsilon.$$

Proof. The error coming from the negative cycles is bounded by Lemma 9.17 by:

$$\epsilon l_{tot} + 2m\sum_{ij} impr_{ij} + \frac{16U_1 + 8l_{max}U_2}{\epsilon}\max_{ij} impr_{ij} l_{tot}. \tag{9.30}$$

The error coming from the processing times is, according to Lemma 9.15 bounded by:

$$l_{tot}\frac{6U_1 + 3l_{max}U_2}{\epsilon}\max_{ij} impr_{ij} + ml_{tot}\epsilon$$

The sum of the above errors leads to the thesis. □

Finally, we obtain the following theorem.

Theorem 9.19. *Let e_i and e_d be the initial and the desired absolute errors. The distributed algorithm reaches the e_d in expected time complexity:*

$$O\left(\frac{{l_{tot}}^2(U_1 + l_{max}U_2)e_i m^3}{e_d^2}\right).$$

Proof. In the estimation from Theorem 9.18 we set $\epsilon = \frac{e_d}{2(m+1)l_{tot}}$ and relax the upper bound by replacing $\max_{i,j} impr_{ij}$ with $\sum_{i,j} impr_{ij}$:

$$\begin{aligned} e &\leq (2m+2)\sum_{i,j} impr_{ij}\left(1 + \frac{22U_1 + 11l_{max}U_2}{e_d}{l_{tot}}^2\right) + \frac{e_d}{2} \\ &\approx 2m\sum_{i,j} impr_{ij}\frac{22U_1 + 11l_{max}U_2}{e_d}{l_{tot}}^2 + \frac{e_d}{2}. \end{aligned}$$

Thus, either

$$2m\sum_{ij} impr_{ij}\frac{22U_1 + 11l_{max}U_2}{e_d}{l_{tot}}^2 \leq \frac{e_d}{2},$$

and the algorithm has already reached the error e_d; or in every execution step we have:

$$\sum_{i,j} impr_{ij} \geq \frac{e_d^2}{4m(22U_1 + 11l_{max}U_2){l_{tot}}^2}.$$

The expected improvement of the distributed algorithm during every pairwise communication is $\frac{1}{m^2}\sum_{i,j} impr_{ij}$, and thus it is lower bounded by:

$$\frac{e_d^2}{4m^3(22U_1 + 11l_{max}U_2){l_{tot}}^2}.$$

Thus, after, in expectation, $O(\frac{{l_{tot}}^2(U_1+l_{max}U_2)e_i m^3}{e_d^2})$ steps the initial error drops to 0. This completes the proof. □

For the relative errors $e_{i,r} = \frac{e_i}{l_{tot}}$ and $e_{d,r} = \frac{e_d}{l_{tot}}$, the complexity of the algorithm is equal to $O(\frac{l_{tot}(U_1+l_{max}U_2)e_{i,r}m^3}{e_{d,r}^2})$.

9.6 Batch Model: Optimal Solution

In this section we consider the batch model, i.e., the case where the load functions are linear: $f_i(l) = \frac{l}{2s_i}$. We assume that there is a central processing unit that has complete knowledge about the whole system. Given the communication latencies c_{ij} and the servers' initial loads n_i, our goal is to find an algorithm setting relay fractions ρ_{ij} so that the total processing time of all the requests $\sum C_i$ is minimized. Below we show that the problem can be stated as a quadratic programming problem with a positive-definite matrix Q. This means that the optimization problem is convex and, in particular, it is solvable in polynomial time. We consider this result powerful, as it indicates that any local optimization techniques can be applied to the problem.

Theorem 9.20. *The problem of minimizing $\sum C_i$ for the batch model can be expressed as a quadratic programming problem: $\sum C_i = \rho^T Q \rho + b^T \rho$, with a positive-definite matrix Q. In particular, this means that the function we minimize is convex.*

Proof. We express the total processing time $\sum C_i$ in a matrix form as $\sum C_i = \rho^T Q \rho + b^T \rho$, where:

- ρ is a vector of *relay* fractions with $m \cdot m$ elements. $\rho_{(i,j)}$, the element at $(i \cdot m + j)$-th position, denotes the fraction of local requests of i-th server that are relayed to j-th server ρ_{ij}, thus:
  $\rho = [\rho_{(1,1)}, \rho_{(1,2)}, \ldots, \rho_{(1,m)}, \rho_{(2,1)}, \ldots, \rho_{(m,m)}]^T$;
- Q is m^2-by-m^2 matrix in which $q_{(i,j),(k,l)}$ denotes the element in $(i \cdot m + j)$-th row and in $(k \cdot m + l)$-th column:

$$q_{(i,j),(k,l)} = \begin{cases} n_i n_k / s_j & \text{if } j = l \text{ and } i < k; \\ n_i n_k / 2s_j & \text{if } j = l \text{ and } i = k; \\ 0 & \text{otherwise;} \end{cases} \tag{9.31}$$

  Figure 9.4 presents the structure of matrix Q.

- b is a vector with m^2 elements with b_{ij} denoting an element at $(i \cdot m + j)$-th position: $b_{(i,j)} = c_{ij} n_i$.

The following derivation shows how the matrix Q is constructed:

$$\rho^T Q \rho = \sum_{i,j} \rho_{(i,j)} \sum_{k \geq i} q_{(i,j),(k,j)} \rho_{(k,j)} \tag{9.32}$$

$$= \sum_{i,j} \rho_{(i,j)} \left(\sum_{k>i} \frac{n_i n_k \rho_{(k,j)}}{s_j} + \frac{n_i^2 \rho_{(i,j)}}{2s_j} \right) \tag{9.33}$$

$$= \sum_i \sum_j \sum_k \frac{n_i n_k \rho_{(i,j)} \rho_{(k,j)}}{2s_j} = \sum_i \sum_j \frac{r_{ij} l_j}{2s_j}. \tag{9.34}$$

(9.32) follows from the construction of the matrix Q (only elements $k \geq i$ are non-zero). (9.33) substitutes $q_{(i,j),(k,l)}$ with the values defined in (9.31). (9.34) uses commutativity of multiplication and substitutes $l_j = \sum_k n_k \rho_{(k,j)}$ and $r_{ij} = n_i \rho_{(i,j)}$.

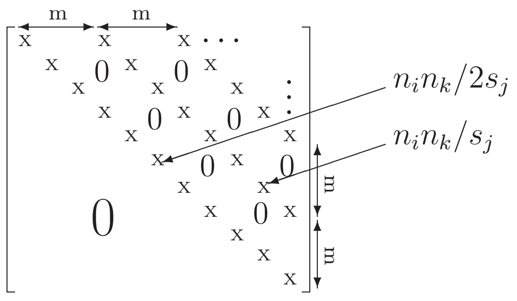

Figure 9.4: Matrix Q: X denotes non-zero values

The constraints that ρ_{ij} are the fractions ($\forall_{i,j}\ \rho_{ij} \geq 0$ and $\forall_i\ \sum_{j=1}^{j=m} \rho_{ij} = 1$) can also be expressed in the matrix form. First, $\rho \geq 0_{m^2}$, where 0_{m^2} is a vector of length m^2 consisting of zeros. Second, $A\rho = 1_m$, where 1_m is a vector of length m and consisting of ones, and A is a m-by-m^2 matrix defined by the following equation:

$$a_{ij} = \begin{cases} 1 & \text{if } im \leq j < (i+1)m \\ 0 & \text{otherwise.} \end{cases} \tag{9.35}$$

Minimization of $\sum C_i(\rho) = \rho^T Q \rho + b^T \rho$ with constraints $\rho \geq 0_{m^2}$ and $A\rho = 1_m$ is an instance of quadratic programing problem. As an upper triangular matrix, matrix Q has m^2 eigenvalues equal to the values at the diagonal: $n_i^2/2s_j$ $(1 \leq i, j \leq m)$. All eigenvalues are positive so Q is positive-definite. □

Corollary 9.21. *The problem of minimizing $\sum C_i$ for the batch model is solvable in polynomial time.*

According to [137], the best running time reported for solving quadratic programing problems with linear constraints is $O(n^3 L)$ [114], where L represents the total length of the input coefficients, and n is the number of variables (here $n = m^2$), so the complexity of the best solution is $O(Lm^6)$. The part of this complexity that depends on the number of machines is higher than in case of the algorithms for the general load functions. Here, however, the complexity does not depend on the derivative of load functions, initial error, nor the total load in the system.

Theorem 9.20 additionally encourages one to apply local optimization techniques, such as the distributed algorithm presented in the previous section.

9.7 Experiments

In this section we show the results from two groups of experiments. First, we ran experiments on PlanetLab to validate the assumption that the latencies between servers do not depend on the amount of load sent over the network. Second, for

t_b	$e(\cdot,\cdot,t_b)$	
	μ	σ
10 KB/s	0.0	0.0
20 KB/s	-0.05	0.21
50 KB/s	-0.05	0.27
0.1 MB/s	-0.08	0.33

t_b	$e(\cdot,\cdot,t_b)$	
	μ	σ
0.2 MB/s	0.0	0.37
0.5 MB/s	0.28	0.8
2 MB/s	0.45	1.31
5 MB/s	0.18	0.8

Table 9.1: The relative deviation of the average throughput caused by the increase of the background load (after removal of 5% largest deviations). In the figure, μ denotes the mean and σ denotes the standard deviation.

the batch model we investigate convergence time of the distributed algorithm, by simulations.

9.7.1 Validation of the Constant Latency Assumption

We experimentally verified how the amount of load sent over the network influences communication delays between the servers. We randomly selected 60 PlanetLab servers, scattered around Europe, and simulated different intensity of the background load in the following way. Each server chooses its 5 neighbors randomly, but in a way that each server has exactly 5 neighbors. Then each server starts sending data with a constant throughput to its 5 neighbors. In different experiments, we used 8 values of the throughputs: 10KB/s, 20KB/s, 50KB/s, 100KB/s, 200KB/s, 500KB/s, 1MB/s, 2MB/s. If a particular throughput was not achievable, the server was just sending data with the maximal achievable throughput. By the nature of experiments on PlanetLab, we were not granted a dedicated access to the machines; thus other experiments running on the same servers added further, unknown network transfers. Additionally, almost none of PlanetLab servers specify the bandwidth of their Internet connection or the historical bandwidth usage. For each value of the background load, we calculated the average round trip time (RTT) between the server and each of its 5 neighbors (we used the average from 300 RTT samples).

Let $rtt(s_i, s_j, t_b)$ denote the average rtt between servers s_i and s_j with the background load generated with throughput t_b. For each pair of servers s_i and s_j for which we measured the RTT, and for each value of the background throughput t_b, we calculated the relative deviation of the average throughput caused by the increase of the background load compared to the minimal throughput 10KB/s: $e(s_i, s_j, b_t) = \frac{rtt(s_i,s_j,t_b) - rtt(s_i,s_j,10KB/s)}{rtt(s_i,s_j,10KB/s)}$. For each value of the background throughput, we removed 5% of the largest deviations and then calculated the mean from deviations $e(s_i, s_j, b_t)$, averaged over all pairs of servers (μ). For each value of the background throughput we additionally calculated the standard deviations (σ). These results are presented in Table 9.1.

From the data, we see that up to $b_t = 0.2$MB/s, which corresponds to the case where each server accepts $5 \cdot 0.2 \cdot 8 = 8$Mb/s of incoming data, the average RTT was not influenced by the background throughput. This is also confirmed by the statistical analysis of the data run for the RTTs (instead of for deviations). For $b_t \leq 0.2$MB/s the ANOVA test (which we run for the whole population—without removing 5% of the highest RTTs) confirmed the null hypothesis (that the background throughput does not influence the RTTs) for over 56% of the pairs of servers. For $b_t \leq 0.1$MB/s (corresponding to 4Mb/s of incoming throughput) the ANOVA test confirmed null hypothesis for over 70% of the pairs of servers and for $b_t \leq 50$KB/s (corresponding to 2Mb/s of incoming throughput) for over 90% of the pairs. We consider that these results strongly justify the assumption of a constant latency in our model.

9.7.2 Convergence of Distributed Algorithm: Simulations

Settings

We experimented on two kinds of networks: homogeneous, with equal communication latencies ($c_{ij} = 20$), and heterogeneous, where latencies were based on measurements between PlanetLab nodes[2] expressed in milliseconds.[3]

In the initial experiments, we analyzed networks composed of 20, 30, 50, 100, 200 and 300 serves. We also performed some experiments on larger networks (500, 1000, 2000, 3000 servers). We used the batch model; the processing speeds of the servers s_i were uniformly distributed on the interval $\langle 1, 5 \rangle$.

We conducted the experiments for exponential and uniform distribution of the initial load over the servers. For each distribution we analyzed five cases with the average load equal to 10, 20, 50, 200 and 1000 requests per second (assuming that processing a single request on a single server takes 1ms). We also analyzed the case of peak distribution—with 100.000 requests owned by a single server.

We evaluated the results based on the distance to the optimal solution, which because of the $O(m^6)$ complexity of standard solvers (see Section 9.6) was approximated by our distributed algorithm.

Convergence Time of the Distributed Algorithm

In the first series of experiments, we evaluated the efficiency of the distributed algorithm measured as the number of iterations the algorithm must perform in order to decrease the difference between the total processing times in the current and the optimal request distributions to less than 2% of the average load. In a single iteration of the distributed algorithm, each server executes distributed algorithm from

[2]`http://iplane.cs.washington.edu/data/data.html`

[3]The dataset does not contain latencies for all pairs of nodes, so we had to complement the data by calculating minimal distances.

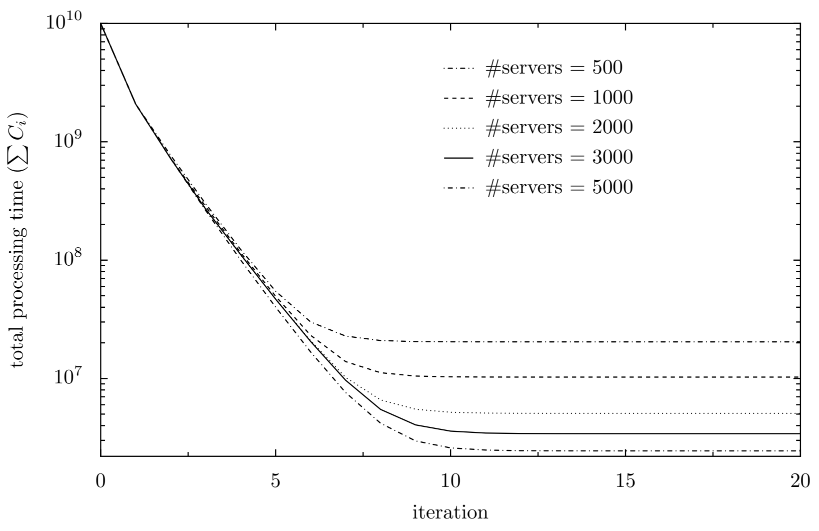

Figure 9.5: The convergence of the distributed algorithm for peak distribution of initial loads.

Figure 9.3; if there were many pairs of the servers to be optimized we run optimization in the random order. Table 9.2 summarizes the results.

The results indicate that the number of iterations mostly depends on the size of the network and on the distribution of the initial load. The type of the network (planet-lab vs. homogeneous) does not influence the convergence time. Larger networks and peak distribution result in higher convergence times. In all considered networks, the algorithm converged in at most 8 iterations.

Next, we increased the required precision error from 2% to 0.1%, and ran the same experiments. The results are given in Table 9.3. In this case, similarly, the required number of iterations was the highest for peak distribution of the initial load. In each case the algorithm converged in at most 11 iterations. Even for 300 servers the average number of iterations is below 8. Also, the standard deviations are low, which indicates that the algorithm is stable with respect to its fast convergence.

Finally, we analyzed the convergence of the distributed algorithm on larger networks (Figure 9.5). The previous experiments have shown that the algorithm convergence is the slowest for peak distribution of the initial load, therefore we chose this case for the analysis. The experiments used heterogeneous network. The results indicate that even for larger networks the total processing time decreases exponentially.

		# iterations		
		average	max	st. dev.
$m \leq 50$	uniform	1.65	3	0.49
	exp.	2.35	3	0.47
	peak	4.87	6	0.71
$m = 100$	uniform	2.0	2.0	0.0
	exp.	2.62	3	0.48
	peak	6.88	7	0.32
$m = 200$	uniform	2.1	3	0.33
	exp.	3.1	4	0.33
	peak	7.84	8	0.37
$m = 300$	uniform	2.0	2	0.0
	exp.	3.25	4	0.43
	peak	8.0	8	0.0

Table 9.2: The number of iterations of the distributed algorithm required to obtain at most 2% relative error in the total processing time ΣC_i.

		# iterations		
		average	max	st. dev.
$m \leq 50$	uniform	5.1	7	1.0
	exp	5.5	7	0.9
	peak	6.4	7	0.5
$m = 100$	uniform	5.8	9	1.6
	exp.	6.3	9	1.5
	peak	8.0	9	0.2
$m = 200$	uniform	6.1	9	2.2
	exp.	7.1	10	2.0
	peak	9.9	10	0.3
$m = 300$	uniform	6.2	10	2.4
	exp.	7.7	11	2.0
	peak	10.0	10	0.0

Table 9.3: The number of iterations of the distributed algorithm required to obtain at most 0.1% relative error in the total processing time ΣC_i.

9.8 Conclusions

In this chapter we considered the problem of balancing the load between geographically distributed servers. In this problem the completion time of a request is a sum of the communication latency needed to send the request to a server and the server's processing time. The processing time on a server is described by a load function and depends on the total load on the server. Through the most part of this chapter, we considered a broad class of load functions with the mild assumptions that they are convex and twice differentiable.

We presented two algorithms—the centralized one and the distributed one. Both algorithms are any-time algorithms that continuously optimize the current solution. We have shown that both algorithms converge for almost arbitrary load functions. We also presented bounds on speed of their convergence that depend (apart from the standard parameters) on the bounds on the first and second derivatives of the load functions. The centralized algorithm decreases an initial relative error $e_{i,r}$ to a desired value $e_{d,r}$ in time $O(\frac{l_{tot}(U_1+l_{max}U_2)e_{i,r}}{e_{d,r}^2}m^4)$. The distributed algorithm decreases $e_{i,r}$ to $e_{d,r}$ in time $O(\frac{l_{tot}(U_1+l_{max}U_2)e_{i,r}}{e_{d,r}^2}m^3)$. Also, for the large values of initial error $e_{i,r}$, the centralized algorithm decreases the error by half in time $O(\frac{l_{tot}(U_1+l_{max}U_2)}{e_{i,r}}m^5 \log m)$.

The distributed algorithm is based on the idea of gossiping. To perform a single optimization step, the algorithm requires just two servers to be available. Thus, the algorithm is robust to transient failures. It also does not require additional protocols. In some sense it is also optimal: We proved that the local optimization step performed by this algorithm cannot be improved. Finally, at any time moment, during the execution of the distributed algorithm, we are able to assess the current error.

Experimental results show that for the batch model the distributed algorithm quickly converges to the optimal solution.

Part III

Resource allocation in real systems

In the two following chapters we consider resource management mechanisms in two real distributed systems. The approach presented in this part of the dissertation is different from the previous ones in several aspects. First, the considered systems are too complex to obtain an accurate mathematical model. In most cases, it is even very hard to obtain a well-specified problem. Second, effective resource management requires elaborate mechanisms that, apart from algorithms, consist of communication protocols, monitoring services, or other interdependent resource management services. Consequently, design, analysis, and evaluation of resource management mechanisms in real complex systems requires a different methodology. Since the formal analysis of the whole resource management mechanism is virtually impossible, our analysis mainly focuses on extensive experimental evaluation. Additionally, for such systems we argue that evaluation through simulation is not sufficient, the experiments involving the whole real system are necessary.

In this dissertation we consider two perspectives coming from the analysis of two distributed systems. First, we study the already-tuned and strictly-structured, distributed storage system HYDRAstor [89,218], aimed at the enterprise market. Second, we consider a prototype of a peer-to-peer (P2P) backup system that is built as an unstructured network of low-performance workstations. On these two systems we present two different views on resource management in real distributed systems.

In the first view we are given an existing complex system with already implemented data management mechanisms. In such a case we require a resource allocation mechanism that is decoupled from other parts of the system. In particular, such a mechanism should not influence the architectural specifics of the system, such as the way in which data is distributed between the physical nodes (as this is the responsibility of the data management mechanisms). In this view we consider the following resource allocation problem. In HYDRAstor there are several categories of tasks: user tasks, i.e., data writes and reads, and a number of different types of background tasks (maintenance tasks). The goal is to define a way in which these different categories of tasks will be allocated common resources.

HYDRAstor is a content-addressable storage system—the location of each data block is fully determined by the hash of its content; thus, the location of data is fully structured. This is a very complex system developed for many years. In contrast, in the second view we are given a system that is simpler, and in which there is no structured overlay. Thus, in this view we can interfere with the other parts of the system such as mechanisms of locating data. For instance, resource management solutions may include mechanisms that optimize data distribution in order to provide, e.g., a certain degree of resilience, little overhead for the network, or faster data access.

In other words, in the first view, our goal is to design a single subsystem that provides proper resource distribution between existing services. In the second view, on the other hand, we aim at providing a whole set of new services that result in a smart allocation of resources. These two views, however, have common properties. In both views we consider real distributed systems, which provide efficient and reliable storage service. In both cases we use a similar methodology of analyzing the resource management mechanisms: (i) We provide limited formal analysis of selected parts of the mechanisms. (ii) We provide extensive experimental evaluation of the proposed solutions. (iii) We argue that experimental evaluation through simulations is not enough, and so we run our experiments on real systems.

In the first view, studied in Chapter 10, our main contribution is a novel mechanism for sharing resources among various types of highly-variable loads. The new approach uses throughput as the tasks' progress indicator—it controls resource allocation to obtain certain proportions of throughputs of different categories of tasks. For instance, in a system running out of storage space, the system administrator may want to ensure that the proportion of the throughputs of the garbage collection tasks and user writes is 1:1. In a typical case, she may want to use configuration in which the background tasks proceed with, e.g., 5% the speed of the user tasks (unless there are idle resources that could be used). Using throughput as the tasks' progress indicator allows to avoid assumptions about task specifics such as their resource consumption or handling process. This makes our new technique particularly well suited for complex systems for which defining an accurate model is difficult.

We provide and evaluate a fuzzy adaptive control by using it to allocate resources to user load and to background tasks in HYDRAstor. We compare our mechanism with several variants of fair queuing and we show that it is more stable in the case of irregular workloads.

In Chapter 11 we describe an architecture of a P2P backup system that allows one to implement different strategies of replica placement. These strategies may include: geographical distribution of the replicas (to increase data resiliency), minimization of the load sent over the network, or minimization of the backup/restore time. Thus, here, the resource is the storage of heterogeneous physical machines—we want to allocate this resource to different data replicas. Next, we propose an optimization protocol that continuously improves replica placement according to a given strategy. Our optimization protocol works for a general class of optimization strategies.

We experimentally evaluated our prototype implementation on 150 workstations in our university's computer laboratories and, separately, on 50 PlanetLab nodes. We found out that the main factor affecting the quality of the performance-based optimization goals is the availability of the machines. Yet, our main conclusion is that it is possible to build an efficient resource management mechanism in a system built from highly unavailable machines.

Chapter 10

Fuzzy Adaptive Control for Heterogeneous Tasks in High-Performance Storage Systems

Beyond handling user reads and writes, storage systems execute multiple background tasks of various types, such as reconstruction of missing parity data and defragmentation. The resources of the system must be divided between the user load and the internal tasks using a specific policy.

In this chapter we describe a fuzzy adaptive control, which is a novel mechanism for sharing resources among various types of highly-variable loads. In this new approach we use throughput as the task progress indicator—our mechanism ensures that the throughputs of different types of tasks are in certain proportions. As a result, we present a general mechanism that abstracts from the details regarding the tasks in our system (such as their resource consumption or their handling process). Consequently, our new technique is particularly well suited to complex systems for which defining an accurate model is difficult.

We evaluate the fuzzy adaptive control mechanism by using it to divide resources between user load and background tasks in HYDRAstor—a commercial high-performance distributed secondary storage system. We compare our mechanism with some variants of fair queuing and we show that our solution is more stable in the case of irregular workload. We experimentally prove that our approach is responsive to changing load conditions and that it ensures high resource utilization.

10.1 Introduction

In parallel to read/write operations, storage systems often execute multiple types of background tasks, such as reconstruction of parity data, data defragmentation, or garbage collection [186,207,208]. The priority of tasks usually depends on the state of the system. In a standard case, user load has the highest priority in order to achieve the desired quality of service. However, when a failure occurs, missing parity data must be reconstructed with high priority (RAID rebuilding is an example of such reconstruction) even though reconstruction tasks may decrease the speed of processing user writes. The priority of such reconstruction may depend on the current resiliency level. Also, in a system running out of space substantial resources must be assigned to garbage collection. Even in a healthy system, maintenance tasks, such as garbage collection and defragmentation, must not be starved regardless of their lower priority.

User load and background tasks in modern storage systems are significantly different from simple IO operations. In this chapter we consider HYDRAstor which is a distributed storage system (see [89,218] for the detailed description of the architecture of HYDRAstor). In HYDRAstor user writes use multiple resources and are handled by multiple software components. This differs from the systems based on storage arrays in which writes burden hard disks only. Indeed, writes operations include CPU-intensive processes of ereasure-coding, duplicate elimination, etc. Similarly, some background tasks in HYDRAstor perform heavy CPU-intensive computations on data and meta-data.

The tasks in HYDRAstor are diversified in terms of their size. Background tasks are long and heavy—a single task can take up to 15 seconds (which is about 10 times longer than the duration of a user write in a fully loaded system, and about 100 times longer than a user read), and its execution can significantly slow down other tasks that are run in parallel. These tasks cannot be shortened easily for the following reasons. Data in HYDRAstor is stored in the units called data containers [89]. The size of a single data container must be large, because otherwise its size would be dominated by the size of meta-data. Some background tasks process entire data containers and cannot be split into smaller pieces (e.g., sorting); such tasks are in essence large and time-consuming. Other tasks can process only some fragments of the data containers, but splitting them, while theoretically possible, is undesired as it would increase implementation complexity. User requests are smaller—the execution of 2500 user writes in parallel takes about 1.5 second; and for duplicate writes and for reads this time can be reduced even to 150 ms.

Finally, while a storage system has full control over the process of background tasks creation, user load can be highly irregular; requests may be unavailable for a moment and then appear in large number. In each state the system must ensure that (i) user requests' latency is kept below a predefined level, and (ii) many user requests are handled concurrently. Indeed, the filesystem prefetching algorithms [292] require bounded latency to ensure high throughput of read/write operations. On the other hand, to ensure high throughput of the backend system, many requests should be processed in parallel. Unfortunately, these two goals stay, somehow, in contradiction—too high concurrency level may increase the latency of user requests too much. Consequently, the concurrency level must be adjusted carefully in run time,

	user writes	user reads	background tasks
characteristic	highly irregular pattern		unbreakable, non-preemptive
parallelism	2500	2500	10
duration	1.5s	150ms	up to 15s
size	$\approx$ 64KB	$\approx$ 64KB	up to 100MB

Table 10.1: The characteristic of the tasks in HYDRAstor. The value in the row "parallelism" is the number of requests that should be executed in parallel to obtain optimal performance. The value in the row "duration" indicates the required (and optimal) latency of the appropriate categories of tasks.

in response to changes in workloads and in external conditions. The characteristic of the tasks in HYDRAstor is summarized in Table 10.1.

In this chapter we address the problem of scheduling heterogeneous tasks (background tasks and user requests) in storage systems. The tasks are heterogeneous in the following sense: (i) they use multiple different resources, (ii) they have diversified size and (iii) their arrival patterns differ (and are highly irregular). To the best of our knowledge, this problem has not been addressed before. Most literature on proportional resource allocation addresses the cases when all tasks categories have similar characteristic and when they use a single resource [51,124,125,157,158,300, 301]; we must stress that considering task heterogeneity significantly complicates the problem and induces new challenges (c.f. Section 10.3).

The heterogeneity and variability of the loads are the main reasons why standard queuing techniques [35,115,150] cannot be applied. In the queuing-based mechanisms the tasks of different categories are first put into appropriate queues (thus there are as many queues as different categories of tasks). Whenever the system is ready to admit a task, the scheduler selects a queue from which the first waiting task is sent to the system. The scheduler selects an appropriate queue according to a predefined strategy. For instance, if we want to divide throughput between three types of loads according to proportions 2:1:1, then the scheduler, in a loop, selects two tasks of the first type, one task of the second type and one task of the third type. Such queuing-based mechanisms are work-conserving, i.e., the system admits a task whenever it has idle resources. These mechanisms would admit heavy background tasks always when there are no user requests. These tasks would burden the system significantly leaving no resources for incoming user requests. Keeping a system ready for admitting highly volatile user load requires delaying background tasks leading to underutilization of the system resources. However, we are willing to pay this price to guarantee quality of service to the user. Our new mechanism, unlike fair queuing, controls the speed of admitting various tasks and smoothly changes these speeds. This way we avoid burdening the system with heavy background tasks whenever the user load is temporarily unavailable.

	user load	deletion	maintenance
Critical reconstruction	20%	10%	70%
Critical garbage collection	30%	35%	35%
Normal reconstruction	50%	30%	20%
Garbage collection	50%	30%	20%
Regular tasks	70%	30%	0%

Table 10.2: Examples of profiles. The throughput shares depend on the state. For example, critical garbage collection takes effect when free space in the system is low.

The fact that HYDRAstor is a distributed system additionally complicates the problem of tasks scheduling. First, we present a mechanism for controlling local load. Next, we show that with some constraints on the configuration parameters of our control mechanism, the local instances of the mechanisms working independently guarantee that the system works correctly as the whole.

In our approach, user requests and background tasks are scheduled to achieve desired proportional division of observed throughputs (thus, we assume that we can compute the throughput of each category of tasks—this is *the amount of processed data per second*). Our motivation was that proportions of the throughputs reflect the proportions of the observable progresses made by the loads which is of more interest to users than low level details, such as resource consumption (cf. [238]). Throughput-based scheduling is also preferable from the system's point of view, as precise resource allocation and accounting is problematic for complex systems, in particular for distributed ones. Similar assumption was made before [125,158,191].

A throughput division policy is defined with the so-called *throughput shares.* These shares determine the proportions in which the total system throughput should be divided among existing loads. Defining the shares is easy for engineers but not for administrators, thus our system provides a set of profiles, each of them defining the policy for each state of the system. The profiles have intuitive names (such as ”resiliency”, ”performance”) and can be easily chosen by the administrators. The way the shares were calculated in each profile is out of scope of this chapter, because it strongly depends on the types of tasks and internal assumptions in HYDRAstor. The example profiles are presented in Table 10.2.

This chapter has the following contributions: (i) We introduce a novel fuzzy adaptive algorithm enforcing proportional division of the load throughputs while maintaining high resource utilization. (ii) We present how the algorithm can be used in a distributed environment. (iii) We implement the new mechanism in a commercial high-performance distributed storage system—HYDRAstor [89,218]. The described control mechanism has been used for several years by real-world customers. (iv) We evaluate the new mechanism in a 60-server configuration which achieves 10GB/s write performance and has raw capacity of 480TB (that is 10PB with 95% duplicate elimination ratio). This is the world's largest configuration of this system[1]. This configuration is used only for the internal evaluation, and the systems used by the customers have significantly smaller numbers of nodes.

10.2 Related Work

The problem of scheduling tasks in storage systems is common and frequently described. Most available literature addresses the problem of sharing resources among tasks of the same type [124,125,157,158,300,301] or scheduling packets of a known size [51]. This chapter addresses the issue of dividing resources among user load and background tasks.

Scheduling methods for heterogeneous tasks are fairly more complicated, which is, for example, pointed out by Popescu and Ghanbari [243]. Most of the approaches to scheduling homogeneous tasks use the mechanism of queuing: YFQ [35], SFQ and FSFQ [115,150], or their modifications [51,124,300,301,315]. Each load source puts its tasks in a queue. The tasks are taken from the queue and sent to the system in a way that achieves the desired division of the throughputs. The limitation of these basic solutions is that they use a fixed level of concurrency (the scheduler ensures that a fixed number of tasks is processed concurrently) and thus, in case of a server performance drop or a change in a workload (e.g. heavier tasks), they are unable to respect latency requirements.

Some solutions combine the application of the queuing theory with a feedback loop controlling the latency [139,311]. Standard queuing is enriched with a feedback controller that determines the number of concurrently processed tasks so that their latency requirements are met.

The queuing approach in most regular cases was proven to give required throughput division with almost negligible settling time. We address these solutions (as the most important ones) in the context of HYDRAstor system. In Section 10.3 we show that solutions based on queuing do not give correct resource division in case of systems that must accommodate strong irregularity of user requests; this fact motivated us to introduce our new fuzzy adaptive mechanism.

New trends in the application of the control theory encourage to use model-based solutions [83,133,162,314,315]. Feedback controllers are an established method to control one parameter (the input signal) to obtain a certain value of the other parameter (the output signal). Feedback controllers can be used for ensuring proportional resource division as well. One controller may regulate the speed of work of the load sources (input signal) in order to obtain the desired proportions of their throughputs (output signal). Another controller may regulate the concurrency level (the number of user tasks handled in parallel). Finding a proper model, however, requires identifying how presence of each task influences the performance of the

[1]This information is from 2013.

others. Having three kinds of user requests (writes, reads, duplicate writes[2]), several maintenance tasks and a few data deletion phases[3], the model would have many dimensions. Moreover, feedback controllers using linear (or even dynamic) models are less adequate for optimization problems, such as optimizing resource utilization. Control theory gives us well studied methodologies for ensuring stability and the best settling time but with the complicated architecture of our system and, in consequence, with the difficulty of modeling it accurately, we decided to consider different approaches to ensuring proportional resource division. However, the idea of our solution is also based on controlling the parameters of the system that influence the throughput of the different categories of the tasks (these parameters are called *limits*), and we use standard controllers as the elements of our resource management system.

Resource division of tasks of different nature can be also accomplished through virtualization [152,191,232,238,308]. Each virtual machine hosts a single application performing an appropriate class of tasks. Virtual machines allow only for accurate division of CPU cycles among applications; our tasks, on the other hand, use multiple resources, and, consequently, a certain proportion of the CPU cycles does not have to induce the same proportion of the tasks throughputs. In addition, using virtual machines results in architectural and implementational limitations (the tasks of a particular category should be executed by a single software component since they need to be run on a separate virtual machine).

Other existing solutions try to execute background tasks in idle periods [94,207, 208] but such methods are inadequate when servers constantly handle user requests and when background tasks can effectively be executed in parallel to user activity.

10.3 Resource Allocation under Irregular Workload: New Challenges

The problem of proportional throughput allocation is often addressed in the literature [51,125,150,157,158,300]; however always in the context of the tasks with similar characteristics. A great majority of the current approaches suggests using one of the mechanisms based on fair queuing [51,124,125,150,157,158,301,311,315]. In the queuing mechanism the requests of different categories are put into the appropriate queues. In each step the scheduler selects one queue and starts processing the first request from the selected queue. In consecutive steps the queues are selected in a way to keep the desired throughput proportions (the specific mechanisms differ in the policies of selecting the queue).

[2]HYDRAstor uses data duplicate elimination (see [33,205]); Writing duplicates may be an order of magnitude faster than writing non-duplicates.

[3]Data deletion is a process of marking blocks of data for removal. In our system data deletion can be called on demand and we treat it as a special kind of background task.

Since the queuing mechanisms treat the underlying system as a black box it seems natural to adopt this approach for the heterogeneous tasks. Apparently, the heterogeneous tasks exhibit new challenges which make standard approaches behave surprisingly wrong. In this section we present the nature of the new problems on the example of a queuing mechanism. For our analysis we use a simplified model that hardly captures the complexity of HYDRAstor. However, such a simplified model precisely exposes the root of the problems of scheduling irregular tasks using fair queuing. Although our model characteristics were influenced by the observations of tasks in HYDRAstor, we believe that the presented problems might be relevant also for the non-storage computer systems.

10.3.1 The Simplified Model

As we argued before, our system is too complex to obtain its accurate model. In this section we present a very simplified, non-accurate model that hardly captures the complexity of HYDRAstor. This model is not used to verify our solution, but rather to expose the problems with application of the standard queuing mechanisms to our setting. This simplified model allows one to understand the key difficulties when dealing with the highly variable workload. In other words, proving that our solution gives expected results in simulations based on this model is not sufficient; on the other hand showing that queuing-based solutions do not work correctly even for the simplified scenarios is sufficient to expect that such solutions would not work correctly in the real complex system.

Our simplified model reflects the basic idea that the duration of a single task mostly depends on the concurrency level (the number of tasks currently processed by the system). This number of tasks processed at the same time moment is the current load of the system. Figure 10.1 presents the dependency between the system throughput and the level of concurrency (i.e., the load of the system) for user writes in HYDRAstor. Although different categories of background tasks affect the overall performance in a very different way, and, as indicated before, modeling the tasks' interdependencies accurately is almost impossible, some of the tasks exhibit similar behavior to user writes, but are heavier and longer. In our simulator we assumed that there are user writes and background tasks of two types (type I and type II). We took the following simplifying assumptions about the background tasks. In our model handling a single background task of type I burdens the system in the same way as handling 5 user writes. Also, processing a single background task of type I takes on average 3 times longer than handling a single user write under the same load of the system. Similarly, the load that a single background task of type II puts on the system is, on average, 25 times larger than the load of a single user write. The execution of a single background task of type II is, on the average, 10 times longer than the execution of a single write in the same conditions.

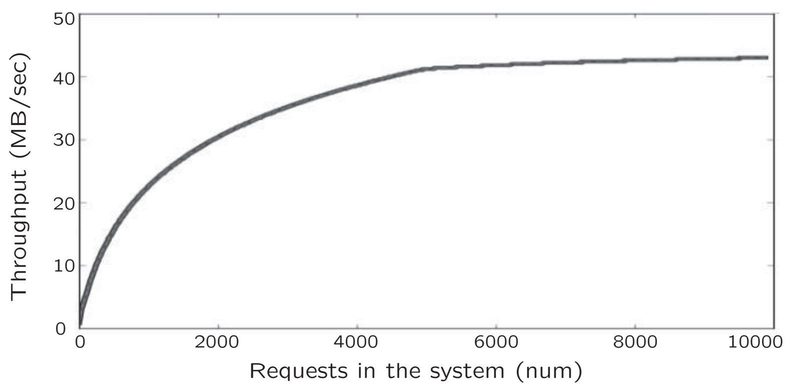

Figure 10.1: The relation between the number of requests in the system and the system throughput used in our simulator. This is the dependency that we observed for the user writes is HYDRAstor.

For example, handling 100 background tasks of type II puts the same load on the system as handling 2500 user writes, or as handling 80 background tasks of type II, 50 background tasks of type I, and 250 user writes. In each of these cases the execution of a certain task of a certain category will take the same time.

Based on these assumptions, we constructed a simple simulator that works as a black box—it admits requests, calculates their duration, and, eventually, when the duration elapses, marks them as completed. Each request does not carry any real data—it is only a header indicating its size and category. A request is held in the simulator for a time period which is sampled with the normal distribution (the time a request is held in the system models the processing time of such request)—the mean of this distribution depends on the load of the system, i.e., on the number and the types of tasks that the simulator currently processes. As we can see from Figure 10.1, increasing the concurrency level to some point increases system throughput, but beyond this point submitting more requests does not result in further performance improvement.

Additionally, the simulator contains three sources generating the requests. The three load types correspond to the user writes and the two categories of background tasks—small and large. The patterns of availability for the user requests are strongly diversified. Thus, again for the sake of the clarity of the presentation, we turned to a simplified model in which user requests come in regular batches, and within each batch the release times of the requests are sampled with the Poisson distribution. In our further presentation we will use the following notation: (x ms / y ms) means that the writes are generated in a loop—x ms of writing and y ms of sleeping.[4] The characteristics of the tasks used in the simulator are summarized in Table 10.3.

[4]This scenario corresponds to copying files between two storage systems.

	writes	background tasks I	background tasks II
weight	1	5	25
avg. duration	$1d$	$3d$	$10d$
pattern	batches	continuous	
batch pattern	Poisson pattern	—	

Table 10.3: The characteristic of the tasks in simulator. In the table d denotes the average duration of a single user write. Thus, d depends solely on the total weight of the tasks currently processed by the system. This dependency is presented in Figure 10.1.

Even in such a simplified environment, the aforementioned problems become hard to solve with the standard queuing techniques. We describe our arguments for this hardness below.

10.3.2 Evaluation of Queuing-Based Approaches

In the simulator we implemented a queuing mechanism with controlled slots [311]—the standard queuing mechanism is enriched with feedback controller which regulates the number of *slots* (number of concurrently executed requests) in order to keep user requests latency at the predefined level. We set this upper bound on the latency of user writes in our simulator to 1500 milliseconds. Such latency can be obtained for 2500 requests (e.g. 2500 user writes or 1000 user writes and 300 background tasks of the first category).

If there are no free slots then the waiting requests are not admitted to the system and the corresponding load sources are paused. Once the slots are available the load sources are resumed and continue producing requests with the same pattern. To amortize the effects of irregularities in the workload, we introduced additional buffers between load sources and the system simulator. Here, similarly, when the buffer is full, the corresponding load source is paused. Large buffers require significant memory consumption and, what is even more important, increase the average request's latency (this is often unacceptable in real systems). Therefore we tried several small sizes of the buffer: 50, 100, 300 (a buffer of the size 1000 would increase the writes' latency by about 600ms which is too much). Using each of the aforementioned sizes gave similar effect, so the results are presented only for the size 100.

If the user requests do not come in batches, but instead are produced continuously, then the queuing mechanism is very accurate and stable. We tested two variants of fair queuing. In the first one, every request occupies one slot, while in the second background tasks occupy respectively 5 and 25 slots. Both variants gave similar results so we present results from the experiments testing the second variant only. Figure 10.2 shows the throughputs achieved by the sources for a policy that divides the throughputs of the three load types in proportions 1:1:1.

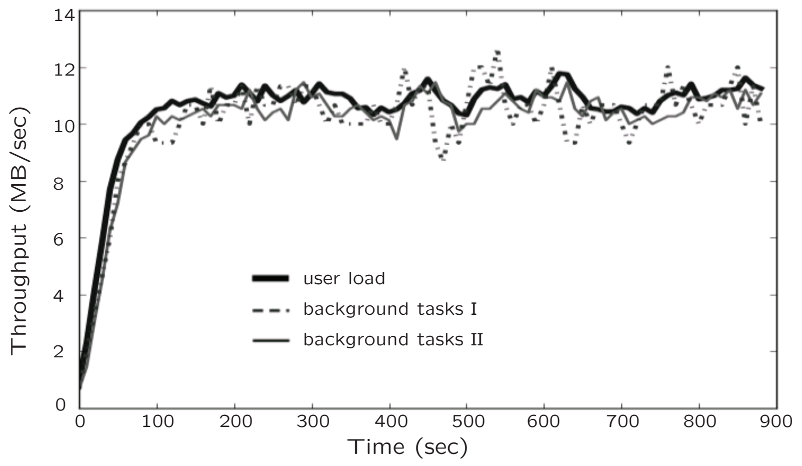

Figure 10.2: The throughputs of user writes and two categories of background tasks on system simulator with queuing using controlled slots mechanism. All three load sources are working at full speed. The fluctuations are the result of short period of averaging.

The case of more irregular writes, which is relevant for real filesystems [116], is more problematic. Let us consider four cases: (i) the user writes are produced in a loop: 300ms of writing and 50ms of sleeping (300ms/50ms); (ii) in a loop (300ms/150ms); (iii) in a loop (1s/300ms); and (iv) in a loop (500ms/500ms). As the user is not writing continuously, the fair queuing mechanism cannot ensure that the average throughput of user load is exactly equal to the average throughput of each type of background tasks. To have equal proportions of the throughputs in long term (as indicated by the throughput shares profile), we modified the queuing mechanism to remember history, so that when user writes appear again, they are preferred till they compensate the previous period of sleeping. Apparently, even with remembering the history,[5] the throughput of user writes may be over 4 times lower than expected, as illustrated in Figure 10.3.

The problem illustrated in Figure 10.3, e.g. for the case (500ms/500ms), is caused by background tasks which are carelessly let into the system. When there are no user writes waiting in the buffer for execution, background tasks are let into the system and spend significant amount of time there (in our case, up to 15 seconds). When user writes are again available in the buffer, the system is still busy and is not able to admit available user writes (within the latency restriction). Even though from now on the mechanism with history will always prioritize user writes over the background tasks, this is not enough to compensate the difference in the throughput. This is because, due to the latency restrictions and limited buffers, the system can accept only limited

[5]The variant without remembering history gave similar results.

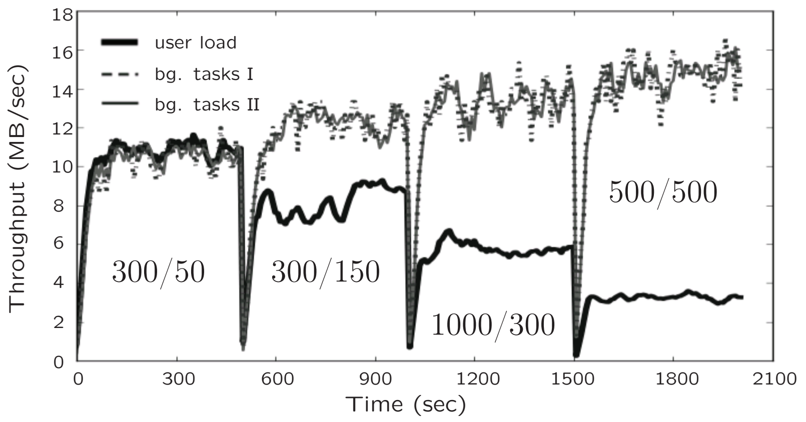

Figure 10.3: The throughputs of user writes and two categories of background tasks on system simulator with queuing using controlled slots mechanism. Four cases: (i) 300ms of writing and 50ms of sleeping (300ms/50ms), (ii) 300ms/150ms, (iii) 1s/300ms, and (iv) 500ms/500ms, are presented sequentially on one plot. User writes (thick solid line) are irregular. The expected throughput proportions are: 1:1:1.

number of write requests. The writes are still produced with the same pattern, so in a short time the same situation happens again: when finished processing lots of requests in the same time, the system takes all the writes from the buffer; now, as the buffer is empty, the background tasks are carelessly admitted by the work conserving scheduler. What is more, such scenario will happen independently of the predefined throughput shares.

The only case where queuing mechanism works as desired is (i) (300ms/50ms). This is because during the 50 milliseconds of sleeping some of write tasks are still in the buffer. As system is not lacking user writes, the background tasks are not started and the behavior is similar to the regular case[6] (see Figure 10.2). In all other cases, the queuing mechanism gives unsatisfactory results.

The results from the same scenarios but using our new fuzzy adaptive framework are presented in Section 10.6.

10.4 Fuzzy Adaptive Control

The evaluation presented in the previous section shows that unless we use large buffers, no work-conserving mechanism can ensure proper resource division in case of volatile user workload (instead of expected 1:1:1 proportions in some cases we obtained

[6]Increasing the capacity of the buffer may improve accuracy of the queuing mechanism in some of the cases but too large buffers are unacceptable.

5:5:1). Thus, we modified the queuing mechanism by introducing the possibility of controlling the speeds of producing/admitting tasks. Considering the aforementioned problems we wanted the new mechanism to have the following properties:

1. Smooth adaptiveness. When in a given time moment there are no user writes, the system does not admit background tasks eagerly (such a mechanism is no longer work-conserving), but, instead, smoothly and slowly starts increasing the speed of admitting background tasks.

2. High utilization. The requirement of the smooth adaptiveness may result in underutilization of the system. We would like to reduce this effect as much as possible, and utilize resources effectively. For instance, we would like to admit short and light tasks more eagerly than long and heavy background tasks.

The adaptive control mechanism, however, requires introducing a new algorithm of controlling the throughputs. In our mechanism, each load in the system is controlled by its *limit*, a variable. The *limit* of each load is adjusted by a controller which takes into account availability of resources and system utilization.

We cannot use a standard multiple-output controller because of many conditions that govern its decisions. Basically, standard feedback controllers increase the *limit* for the loads which are under their throughput share and decrease for the ones which are above their share (see the work of Hellerstein et al. [133] for standard techniques of designing feedback controllers). Such controlling algorithms would not work correctly for the following reasons.

1. If some load does not achieve its throughput fraction, then increasing its *limit* does not always make sense. Such load may be starved by other types of loads and increasing its limit would not increase its throughput, as already there are no free resources in the system. For instance, if we set the limit for a load to 50MB/s and the system is able to process tasks with the throughput at most equal to 30MB/s, then increasing the limit from 50MB/s to 60MB/s will not affect the throughput of that load. *In such a case, we say that the load does not meet its limit.* In such cases we would rather want to decrease the *limit* for other types of load, instead.

2. On the other hand, if some load does not achieve its throughput fraction, decreasing the *limits* for other types of load is not always optimal. Naturally, sometimes it is sufficient to increase the limit for the load that is under its required throughput fraction.

In addition to the above arguments, designing a feedback controller to work correctly in a distributed environment seems hard. Thus, we decided to design the fuzzy adaptive resource allocation mechanism. This mechanism takes as an input descriptive information about loads. For example, the loads indicate whether they are able to speed up when given more resources, or whether they meet their limits. Due to this descriptive information, we are able to distinguish the situations where some type of load is working too slowly because it is given too low a limit (in such a case, such load would meet its limit) or because it is starved by the other loads whose limit is set too high (i.e., if our underperforming load can speed up when given more resources, and if, additionally, it does not meet its limit).

10.4.1 Architecture

The high-level structure of the fuzzy adaptive resource division mechanism is presented in Figure 10.4. Every load source has its own admission control mechanism (AC) that controls the speed of its work based on the value of an exposed limit variable. The algorithm periodically[7] collects information about the loads (in the diagram denoted by info), and adjusts the current values of their limits (setLimit) in order to achieve proportions of the throughputs as in the policy. The admission control sends to the system only as many tasks as the value of its limit allows for. The limit might denote the upper bound for the throughput (for background tasks and for deletion tasks) or the upper bound on the slots, i.e., the number of concurrently executed tasks (for user writes). We discuss the idea behind the choice of the implementations of the limits in Section 10.4.2, below.

The tasks might be not sent to the system either because the value of the limit does not allow for it, or because of some additional constraints. These constraints are described in detail in Section 10.4.3. In the diagram we show an example of such a constraint for user writes—the value of the slots is affected by Latency Controller, that tries to keep requests' latencies at the appropriate level.

The idea of the adaptive high-level controller comes from the area of fuzzy control. In each cycle, every AC prepares *info* which consists of performance metrics and descriptive information on the state of each load (the fields sent as info are described in Section 10.4.4). The process of extracting such descriptive information is called *fuzzyfication* [133]. The algorithm collects *infos* from all the ACs, *filters* the performance metrics (e.g. , by averaging), makes *decisions*, and forwards them to the proper controllers. The decision is also in a descriptive form and has one of the following values: *increase*, *decrease*, or *no-change*. The controllers are responsible for defuzzyfication; they calculate new values of the *limit*s and pass them to the ACs. These controllers are described in Section 10.4.7.

In every cycle, the limits are corrected to make the actual division of the throughputs closer to the desired one (as indicated by the policy). There is a trade-off between the speed of the convergence of the throughput division to the division from the policy and the stability of the system.

[7]The interval is chosen to satisfy the constraints presented in Section 10.5 – for HYDRAstor it is 500 ms.

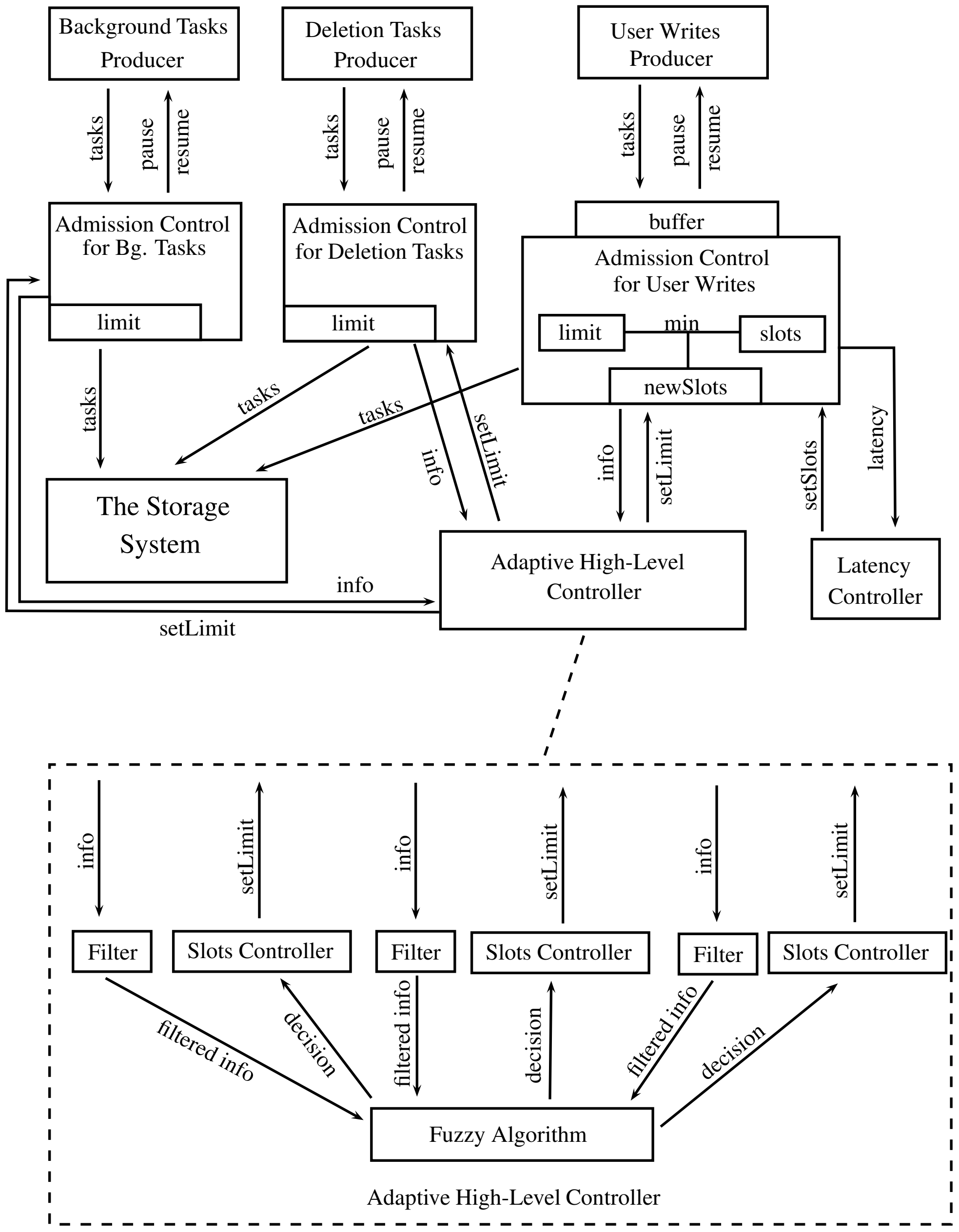

Figure 10.4: The high-level mechanism of adaptive resource management.

10.4.2 Implementation of the Limit Variables

We believe that in many systems there might be a couple of reasonable choices for the limit variables. In HYDRAstor we introduced two different implementations. In the case of background tasks, the limit is the upper bound on the throughput. For user load, the limit is the upper bound on the number of concurrently executed tasks (slots). This is motivated by the necessity for efficient utilization of system resources. For example, consider the case where the algorithm increases the limit for the user load but the system has not converged to the steady state yet (or the case where there is temporarily no background tasks). If we used the bound for the throughput, the system would be underutilized until the algorithm increased the limit to a sufficiently high value. In case of using the bound on the slots, even if the limit is too low (because the algorithm has not achieved a steady state yet) the requests will have lower latency, which results in higher throughput. This is illustrated in the example below.

Example 10.1. Let us assume that the maximum performance of the system is 50MB/s, which is achievable with 2500 concurrently processed user writes. Further, let us assume that the user wants to write to the system with its full speed. If we set the bound on the throughput to 25MB/s, we will obtain exactly such throughput (even if the system is underutilized). On the other hand, limiting the slots to 1250 would result in having the throughput equal to 25MB/s if the system is loaded or to 40MB/s if the system is underutilized. This is because in the underutilized system the requests are processed faster.

10.4.3 Additional Constraints

Since the user requires a bounded latency, there is a separate controller which regulates the number of slots in order to achieve the desired referral latency L_{ref} (cf. [51,125,156–158,191]). We recall that in HYDRAstor this referral latency L_{ref} is equal to $1.5s$. If the user requests exceed the expected latency, the latency controller decreases the number of slots even if the AC limit allows for a higher number.

In case of background tasks, there is an additional constraint for the number of concurrently processed tasks. This constraint is used to eliminate thrashing and to decrease memory consumption and is chosen in a way that does not affect the performance.

Given aforementioned constraints we see that if the system is lacking resources, the load source may not be able to achieve the given AC limit (we say that the AC *did not achieve its limit*). Such situation may happen if the system is loaded and the required latency enforces the number of slots which is lower than given by the limit, or when the background tasks are not able to achieve the desired throughput. In spite of lacking resources, a load may not achieve its limit also when it has not enough work to do (in such a case this load simply does not need resources).

Introducing the aforementioned constraints seems justified and reasonable even as an independent mechanism, however we would like to stress that the constraints are also required by our algorithm in order to determine the values of the appropriate variables (see the implementation of *needsResources* described in Section 10.4.5).

```
: recountDesiredShares
    targetShares = {};
    spareShare = 0;
    total = 0;
    foreach loadInfo in infos do
        if loadInfo.needsResources = false and
        loadInfo.currentShare < loadInfo.policyShare then
            targetShares[loadInfo.load] ← loadInfo.currentShare ;
            spareShare ← spareShare + (loadInfo.policyShare
                            - loadInfo.currentShare);
        else
            targetShares[loadInfo.load] ← loadInfo.policyShare;
            total ← total + loadInfo.policyShare;
    while spareShare > 0 do
        newTotal ← total;
        foreach loadInfo in infos do
            if loadInfo.needsResources = true or
            targetShares[loadInfo.load] < loadInfo.currentShare then
                newTS ← targetShares[loadInfo.load]
                            + $\frac{loadInfo.policyShare}{total}$ spareShare;
                if loadInfo.needsResources = false and
                newTS ≥ loadInfo.currentShare then
                    newTS ← loadInfo.currentShare ;
                    newTotal ← newTotal - loadInfo.policyShare;
                spareShare ← spareShare
                            - (newTS - targetShares[loadInfo.load]);
                targetShares[loadInfo.load] ← newTS ;
            total ← newTotal;
    return targetShares
```

Figure 10.5: The function `recountDesiredShares` that is used by the fuzzy-adaptive algorithm to adjust the policy shares of the loads.

10.4.4 Admission Control (AC) Information

The information (called info) sent from each AC to the algorithm consists of the following values:

NeedsResources. *NeedsResources* indicates if the tasks can work faster given more resources (see Section 10.4.5 below).

```
Input: policyShares – map of desired shares per load type,
       infos – information collected from ACs. Info is the vector of loadInfos.
       LoadInfo is a structure containing fields like: throughput,
       needsResources, and limitAchieved—see Section 10.4.4
Output: decisions – a map of decision per load type. Decision can be
        INCREASE, DECREASE or NO_CHANGE
1  decisions ← NO_CHANGE;
2  someLoadWasIncreased ← false;
3  someLoadNeedsResources ← false;
4  foreach loadInfo in infos do
5      currentShares[loadInfo.load] ← $\frac{loadInfo.throughput}{totalThroughput}$;
6  foreach loadInfo in infos do
7      if loadInfo.needsResources = true then
8          someLoadNeedsResources ← true
9  targetShares ← recountDesiredShares( policyShares, currentShares, infos);
   // Try to increase limits
10 foreach loadInfo in infos do
11     if (loadInfo.needsResources = true) and (loadInfo.limitAchieved = true)
       and (currentShares[loadInfo.load] ≤ targetShares[loadInfo.load]) then
12         decisions[loadInfo.load] ← INCREASE;
13         someLoadWasIncreased ← true
   // Try to decrease limits - only if no limit was increased
14 if (not someloadWasIncreased) and someLoadNeedsResources then
15     foreach loadInfo in infos do
16         if (currentShares[loadInfo.load] ≥ targetShares[loadInfo.load]) then
17             decisions[loadInfo.load] ← DECREASE;
   // Pass decisions to the controllers
18 foreach load in infos do
19     controller[load].passDecision();
```

Figure 10.6: The fuzzy-adaptive algorithm that takes descriptive information for each load (*load infos*) and for each load finds a *decision*, which is one of three values: *increase*, *decrease*, or *no-change*. The algorithm uses the function `recountDesiredShares` that is presented in Figure 10.5.

Throughput. *Throughput* is the average throughput generated since the previous info was collected.

LimitAchieved. *LimitAchieved* indicates whether the load type has achieved its limit since the last measurement.

10.4.5 Resources' needs

Defining whether or not a load can work faster given more resources depends on the load properties. Background tasks are accumulated in a buffer and sent to the system when the AC throughput limit allows for it, and when there are no additional constraints for pending load (see Section 10.4.3). In such model, *needsResources* is true unless the buffer is empty.

In case of user load, which can be irregular (see Section 10.3.2), the size of a buffer fluctuates heavily, so indicating whether the source needs resources by examining its buffer fill level is inappropriate. Therefore, for user load we chose a different metric—*needsResources* tells if average user requests latency is higher than the threshold T_{nr}. The threshold is set to the 70% of the referral latency L_{ref}. T_{nr} is lower than L_{ref} to reduce the effect of latency fluctuations (also see Section 10.5.2).[8]

10.4.6 The Fuzzy-Adaptive Algorithm

Using the information collected from the ACs, the algorithm deducts whether the limit for each load source should be increased, decreased, or left unchanged. Based on this decision, the controllers calculate the new value for each limit.

The skeleton of the algorithm is presented in Figure 10.6. As input, it takes information collected from the ACs and the policy, which gives the share for each load.

The core algorithm from Figure 10.6 first, in lines 4–9, calculates the current and the target shares of the loads.

Current shares. The current share of a load is the proportion of its current throughput of the load to the total throughput of all loads.

Policy shares. The policy share of a load is a fraction of the total throughput of all loads that, according to the profile, should be allocated to the load (we recall that an example of a profile with policy shares is given in Table 10.2).

Target shares. The target share of a load is its recalculated policy share. This recalculation is performed by the function `recountDesiredShares` from Figure 10.5 and takes into account such aspects as the fact whether loads need resources or whether their spare resources can be allocated to other loads. The

[8]As explained in Section 10.4.6, if user load has not enough work to do, the background tasks will be let in, to utilize the resources. The limit of background tasks will be increased until user load starts reporting needsResources as true (so, until the user load latency reaches T_{nr}). Because T_{nr} is lower than L_{ref}, we can underutilize the system. The dependency between throughput and latency is not linear (see Figure 10.1)—we will lose about 15% of the system performance. This is acceptable as handling heavy fluctuations and supporting bounds on the user load latency are primary requirements. Also, we note that with the user load working full speed its latency is kept at the referral level, which means the system is fully utilized.

fuzzy adaptive algorithm effectively uses the calculated target shares and tries to keep the throughput proportions according to these target shares.

In line 9, the algorithm distributes the shares of loads that do not need resources among the loads that do. Loads that do not need resources are left only with the shares they utilize (their current shares), and the remaining part of their share is distributed among loads that need resources. The formal procedure of calculating these target shares is given in Figure 10.5. We note that if all loads have work to do and are waiting for resources, then the target share of each load will be equal to the share indicated by the policy. In the further part the algorithm will make changes to the limits to get closer to thus computed target shares. We say that a load type l is eligible to get more resources if $currentShares[l] \leq targetShares[l]$. At least one of the load types needing resources is eligible to get more resources, which is formally expressed by the following invariant.

Lemma 10.2. *If there are load types for which* needsResources = true*, then at least one of them has* currentShare ≤ targetShare*.*

Proof. We first show that all types of load which do not need resources (*needsResources = false*) have *targetShare ≤ currentShare*. Let us analyze the procedure of computing target shares from Figure 10.5. In this procedure, if for some load *needsResources = false* and *currentShare < policyShare*, we set its target share to its current share (line 9 in Figure 10.5), and so *targetShare ≤ currentShare*. If *needsResources = false* and *currentShare ≥ policyShare*, we set target share to the policy share (line 12 in Figure 10.5), and, again, we have that *targetShare ≤ currentShare*. This target share can be increased, but it never exceeds the current share (lines 22–23 in Figure 10.5).

This implicates that:

$$\sum_{\substack{loadType\ \boldsymbol{l}:\\ needsResorces=false}} targetShare[l] \leq \sum_{\substack{loadType\ \boldsymbol{l}:\\ needsResources=false}} currentShare[l]$$

Additionally, the sum of all target shares is equal to the sum of all current shares (which is 100%), so:

$$\sum_{\substack{loadType\ \boldsymbol{l}:\\ needsResources=true}} targetShare[l] \geq \sum_{\substack{loadType\ \boldsymbol{l}:\\ needsResources=true}} currentShare[l]$$

Now it is clear that *currentShare ≤ targetShare* for at least one load type with *needsResources = true*. □

If the load needs resources and if its throughput is lower than the expected one, the algorithm tries to increase the speed of this load, either by increasing the limit for this type of load (if the load source achieves the given limit; lines 10–13 in Figure 10.6) or by decreasing the limit for the others (if the limit is not achieved; lines 14–17 in Figure 10.6). In the latter case, there is no point in increasing the limit more as it already does not affect the speed of the load. In such a case, the limit for the other loads is decreased so that they leave some resources for the loads below their share.

If the load l has work to do and its limit allows for faster speed, but it cannot proceed faster, it must be lacking resources. We formalize this intuition in the definition below.

Definition 10.1. *We call the state when some load l has needsResources 'true' and limitAchieved 'false' as a* system saturation*.*

In order to increase the speed of work of such load l, the limit for other loads should be decreased. On the other hand, if the system is not saturated (each load either does not need resources or achieves the given limit), the algorithm will increase the limit for some load, which is specified by the following theorem.

The theorem below proves that our resource management mechanism makes action that lead to high resource utilization.

Theorem 10.3. *If there is work in the system (at least one load type needs resources), then either the system is already saturated or the* limit *for some load type will be increased.*

Proof. Let us assume there is some work in the system and consider load types needing resources. Either one of them has *limitAchieved* = *false*, which means the system is already saturated, or every load has *limitAchieved* = *true*. In the second case, we can use Lemma 10.2 and infer that there is at least one load type, l, which satisfies all of the following:

$$needsResources[l] = true,$$

$$limitAchieved[l] = true, \text{and}$$

$$currentShares[l] \leq targetShares[l],$$

so the limit for l will be increased by the algorithm. □

Finally, the decisions of the core algorithm are passed to the appropriate controllers (lines 18–19 in Figure 10.6).

10.4.7 Defuzzyfication of the Limit

The algorithm described in Section 10.4.6 makes, for each load, a linguistic decision: *increase/decrease/no-change*. The decision is then converted to the specific value of limit (defuzzyfication).

For each load, to find the proper value of the limit we use two independent proportional integral controllers (PI controllers) [133]: the share-based controller C_s, and the latency-based controller C_l. A PI controller takes an error signal and controls the value of the input signal to minimize the error. The share-based controller C_s uses the difference between the current and the desired shares as control error (err_s); the latency-based controller C_l uses the difference between T_{nr} (the threshold for *needsResources*; see Section 10.4.5) and the current latency as control error (err_l). Both controllers use the limit as the input signal. We take the maximum from two limits returned by the two controllers. The first controller has an intuitive interpretation—it tries to reduce the difference between the current and the desired share. To understand the need for the second controller consider the following cases.

1. If the decision of the algorithm is "increase", and the value returned by the second controller is greater than the one returned by the first controller, we infer that the user load has low latency. The limits of all the loads are increased more aggressively in order to quickly saturate the system.

2. If the decision of the algorithm is "decrease", we want to slow down the user load to the speed for which it would not influence the throughput of other loads. If the user load is processed with latency T_{nr}, the system is still able to admit and process other tasks efficiently.

If there is no user load, we use the difference between the upper limit for the pending background tasks and the number of pending tasks, err_p (see Section 10.4.2) instead of err_l; err_p also measures whether the system is loaded or not.

10.5 Resource Allocation for Multiple Servers

In distributed systems, the tasks might be delegated to multiple servers [89,300]. We are considering a sequence of tasks where the i-th task should be executed at the specific server s_i. The tasks are sent to the servers sequentially. This model captures the characteristics of user writes in HYDRAstor. Indeed, HYDRAstor is a content-addressable storage system—the server that handles a user write is fully determined by the content of the written data block. Consequently, each request needs to be sent to a certain server. In such a model, slow execution of tasks on one node may influence their speed of execution on the other servers. Although the server s_{i+1} may have free resources, it may not be given task for execution because s_i had not admitted the previous task yet. As the result, the speed of execution of the tasks is bounded by the speed of their execution at the slowest server.

The fuzzy adaptive algorithm can be used in such distributed environment with a separate instance of the control mechanism working on each server independently. Such distributed version of our mechanism utilizes resources effectively. Let us consider the group of servers handling a sequential stream of user writes. Let us consider that the server A is a bottleneck. Since the stream is processed sequentially, the speed of processing user writes from such a stream is equal to the speed of processing tasks on A. Naturally, other, faster than A, servers might be significantly underutilized. In such a case, our mechanism allocates spare resources on such faster servers to the background tasks. The algorithm can detect unused resources and allocate them to the background tasks because *infos* describe only the local load. For instance, the load coming from the stream that is processed with the speed of a slow remote server A, reports at the faster server that *needsResources* is equal to false, indicating that there is space for different loads at such faster server.

Recapturing unused resources in a distributed environment requires, however, extra care. The background tasks should be let into the system in such a way that they do not influence the speed of the user load. Ensuring such a guarantee is not straightforward as overshoots and fluctuations in load characteristic are both possible. In the following sections we present the consequences of careless recapturing of the unused resources. Additionally, we introduce the constraints on the algorithm wake-up time w_a that guarantee that the local load, which is let in to recapture unused resources, does not affect the performance of the user load.

10.5.1 Recapturing Unused Resources

In a distributed environment the decrease of the limit for a user load at any server, s, makes the user load slow down at each server. After such a decrease the unused resources may appear on some other server s_o. This server is not a bottleneck for the user load so it infers that it may safely increase the limit for local background tasks; because overshoots and fluctuations are possible, the increase of the limit for the background tasks may make this server become a new bottleneck for a user load, further decreasing global speed of user requests. It may happen that in consecutive steps the servers will alternately increase the throughput of the local tasks, eventually causing the starvation of the user load. The local tasks are not throttled because in each time step one of the servers sees that the user load does not need local resources (because it is not a bottleneck for a user load at the moment), causing even more decrease of its throughput in the next step.

To prevent this problem, we introduce an additional constraint on the algorithm wake up interval w_a. Intuitively, this constraint indicates that we should run iterations of our algorithm frequently enough to prevent system from desynchronization. With the given constraints, after increasing the limit for background tasks, server s_o from the example above will not become a bottleneck for any other load (here, for user load) at least until the next wakeup of the algorithm. Intuitively, if such a server increases the speed of background tasks, then we can guarantee that until the next wakeup time this server will yet have abilities to accommodate some additional incoming load. This is possible only because the wakeup interval is short and only because the servers are not fully utilized. This intuition is described in the following example.

Example 10.4. Consider a server s_o for which the user load in time t_0 reports *needsResources* equal to false. Such a server in time $t_1 = t_0 + w_a$ will increase the speed of the background tasks to take advantage of the spare resources. Naturally, these background tasks may affect the speed of processing of the user writes. Thus, in the worst case, since time t_1 the user load might start reporting *needsResources* equal to true. However, with the short value of the wakeup interval w_a, we can guarantee that this latency of the user requests will not exceed the referral latency until $t_2 = t_1 + w_a$. Consequently, the requests will not be rejected and the number of slots will not be throttled down. Because the user load indicates it needs resources, the limit for background tasks, which was previously increased, will be throttled down in time t_2 back to its previous value, causing no degradation of the user load speed.

10.5.2 Constraints on Wake Up Interval

Consider the implementation of *needsResources* that we use for user load (see Section 10.4.5). We recall that for user load *needsResources* is true whenever the requests latency is higher than T_{nr} (We recall that T_{nr} is a constant that is lower than referral latency L_{ref}. In our implementation $T_{nr} = 0.7 L_{ref}$.). We define the latency of a yet unfinished task as the duration from its admission time until the current moment. We recall that Latency Controller decreases the number of slots (number of concurrently executed requests) only if the average latency exceeds the refferal latency L_{ref}. Now, assume that w_a is lower than $(L_{ref} - T_{nr})$. As we recalled, if *needsResources* is *false* then the latency of requests is lower than T_{nr}. Consequently, after w_a (even if no requests are completed until this time), the latency of requests is lower than$T_{nr} + w_a < L_{ref}$. As a result, Latency Controller does not decrease the number of slots and no requests are rejected until that time.

In the buffer-based implementation, *needsResources* indicates whether there is a request waiting in a buffer. In this case, the server is able to admit requests unless the buffer is full. Let w_a be the algorithm wake up interval, $size_b$ be the size of the buffer, and T_{max} be the maximum throughput of the source. Assume $w_a < \frac{size_b}{T_{max}}$. With such assumptions, if *needsResources* was *false* then, after w_a, the buffer will not be full, which results in no slow down in admitting the requests.

In both cases the constraints on w_a force the desired property: if the tasks of a distributed load (here, user load) do not need resources, and so the local tasks are let in, the speed of the admission of the distributed tasks will not be affected until the next wakeup of the algorithm. If too much background tasks were admitted, the mechanism in the next wake-up time decreases their speed to the previous value.

With such constraints, even with no coordination between the servers the control mechanism will enforce correct proportions of the throughputs on at least one loaded server. If the servers are homogeneous (as in the case of HYDRAstor), the control mechanism will enforce correct proportions of the throughputs on all servers.

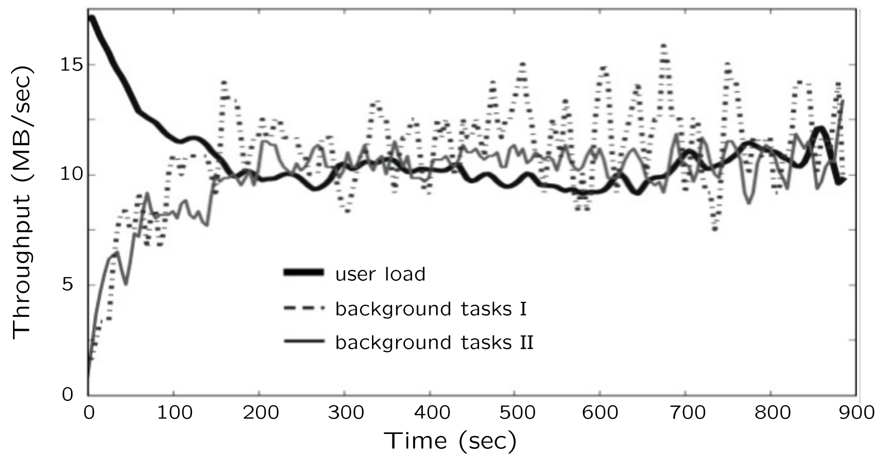

Figure 10.7: The throughputs of user writes and two categories of background tasks on system simulator using fuzzy adaptive control mechanism. User writes (thick solid line) are irregular (500ms/500ms).

10.6 Experimental Evaluation

In the first part of this section we present the results from the experiments using the simulator described in Section 10.3.2. The new fuzzy adaptive algorithm was also implemented in a commercial system, HYDRAstor, and the second part of this section describes experiments conducted on this system.

10.6.1 Artificial Workload

Figure 10.7 presents fuzzy adaptive framework used for irregular writes with the simulator (compare with Figure 10.3). Since the results for each of the 4 scenarios (300ms/50ms; 300ms/150ms; 1s/300ms; 500ms/500ms) were the same, we present the results for 500ms/500ms only. They indicate that fuzzy adaptive controller induces smooth changes and, as a result, forces correct throughput division. After 170 seconds (the time needed for convergence), the average throughput proportions were differing from the expected ones by no more than 3 % (this is when throughputs are averaged over 300s). Figure 10.7 presents the throughputs averaged over 30s, and so including the fluctuations which are the results of (i) the fluctuations in the load pattern (ii) the large size of the background tasks.

10.6.2 HYDRAstor

In HYDRAstor, which is a high-performance distributed storage system [89], apart from user requests (writes, duplicate writes, and reads) there are two main classes of background tasks: data deletion (marking blocks for removal) and maintenance tasks (reconstructions, defragmentation, garbage collection, etc.). A single server can execute concurrently all three classes of tasks, which use the same resources but are handled by different software components.

The shares for the task categories in the experiments were changing dynamically, depending on the state of the system. For example, after failures that significantly reduce resiliency level of some data, there was a critical reconstruction policy in effect, which gave significant share to background tasks. Example polices as results of the states of the system are defined in Table 10.2. The resource management algorithm always uses the policy that corresponds to the current state.

Experimental Setup

For the experiments, we have used HYDRAstor in two configurations: (1) a 60-server configuration and (2) a 6-server configuration. We used the two sizes of the system to verify the influence of irregularity in the background load on the stability of the fuzzy adaptive mechanism. The tasks in a 60-server system (especially data deletion) are more irregular (compare Figures 10.10 and 10.11).

Each storage server had two quad-core 64-bit 3.0 GHz Intel Xeon processors, twelve 7200 RPM Hitachi HUA72101AC3A SATA disks and 24GB of memory. Each server held two logical storage servers, with 6 disks each (for details about HYDRAstor architecture, see [89,218]). The servers were connected with 10Gb network.

High Resource Utilization

The first experiment shows that if writes do not consume all the resources in the system, then maintenance tasks are let in so that all the resources are highly utilized. It also shows that the system is kept loaded independently of which resource is a bottleneck.

In this experiment the characteristic of writes changed three times. For the first 30 minutes, external backup application worked at full speed, generating 60 MB/s per logical node (this is throughput after compression). Then, in the 30th minute, the configuration was changed so that the backup application generated constant throughput of 40 MB/s. In the 60th minute, the write speed was changed to 10 MB/s. Finally, in the 90th minute, we set the write speed to 45 MB/s but included duplicated data. In parallel to handling user writes, the system was doing defragmentation.

The test uses the “Regular tasks” policy from Table 10.2 and there is no data deletion, which means the whole 100% of the throughput should be given to user writes. Background tasks should be allowed to proceed only if there are unused resources in the system.

The results of this experiment are presented in Figure 10.8. The plot shows the throughput of user writes (which in the last period is divided into total write throughput and non-duplicated data throughput) and the throughput of maintenance tasks from one storage server. Figure 10.9 shows the utilization of the processor and hard disks, allowing to identify the bottleneck resource in each phase of writing.

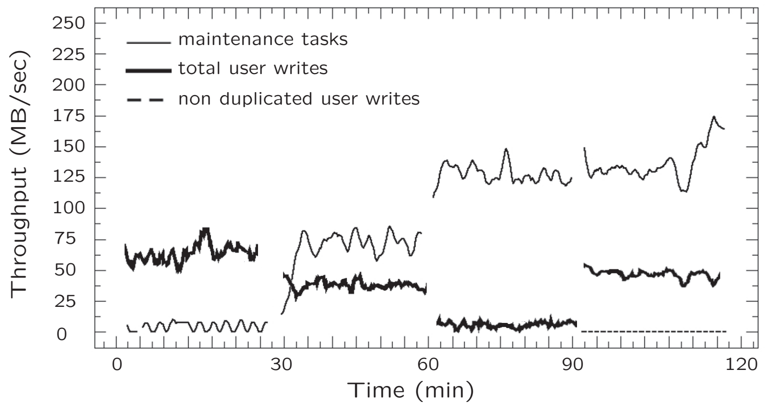

Figure 10.8: The throughputs of maintenance tasks and user writes. Duplicates were present only in the last phase. The experiments were run on a 6-server configuration. In minutes 0-30 the backup application is working at full speed. In minutes 30-60 we changed the configuration of the backup application so that it writes with the speed 40MB/s. In minutes 60-90 we configured the backup application to write with the speed 10MB/s. In minutes 90-120 the backup application is writing with the speed 45MB/s, but writes mostly duplicated data.

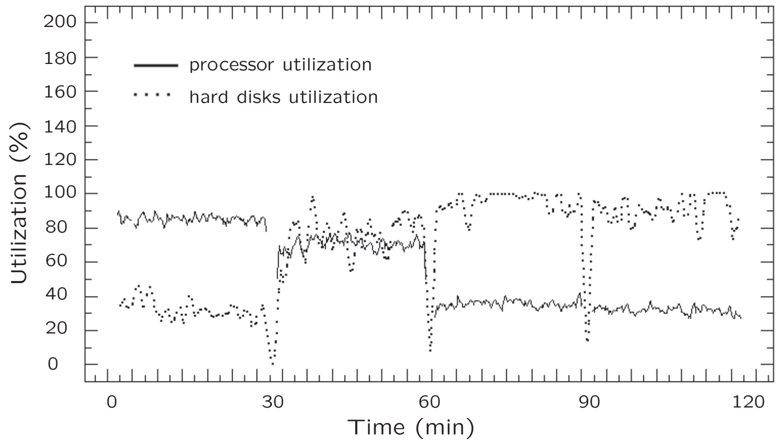

Figure 10.9: Utilization of the processor and hard disks for the experiment from Figure 10.8.

In the first period, when the application worked at full speed the processor was a bottleneck. Recall that user writes are not simple IO operations because HYDRAstor uses data compression, erasure coding, and data deduplication (see Section 10.1). Here the average latency was roughly equal L_{ref} so the system was kept loaded; any further increase of the speed of any load would result in violating the latency contract. In the second period, the unused resources were allocated to maintenance tasks. The maintenance tasks, according to our expectations, did not affect the throughput

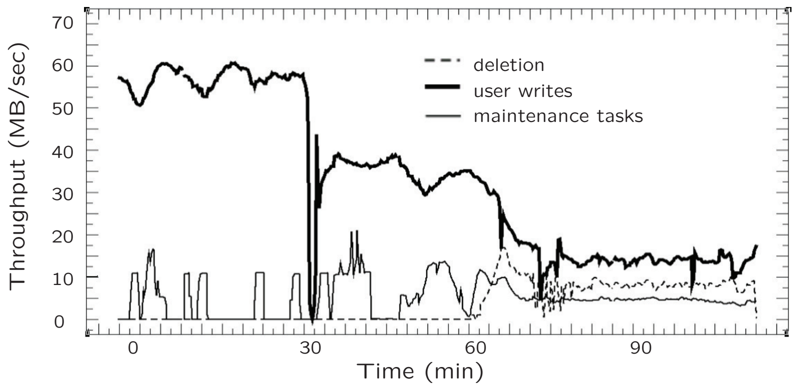

Figure 10.10: The throughputs of maintenance tasks, user writes, and data deletion—6-server configuration.

of the writes. In this phase both the processors and the hard disks were highly utilized. In the third period, when the backup application worked even more slowly, the maintenance tasks were allowed to achieve higher throughput. In this phase there were fewer write tasks executed concurrently so it was easier to maintain their latency bounded. As a result, we managed to obtain almost 100% utilization of hard disks. In the fourth period, writes, in spite of their high throughput, introduced small burden on the system (almost all writes were duplicates), so the maintenance tasks achieved high speed. Writing duplicates to the system does not burden hard disks much. On the other hand, the hard disks are usually the bottleneck for background task. Therefore the background tasks little affected duplicates latency which explains why, here, we obtained almost 100% disks utilization.

Policy Changes

The next two experiments analyze effects of policy changes. They were run with, respectively, 6-server and 60-server configuration. In both experiments the backup application writing to the system was configured to work at full speed for the whole duration of the experiment. In the first experiment, for the first 30 minutes, the system executed defragmentation tasks so it used the "Regular tasks" policy from Table 10.2. Next, in the 30th minute, we simulated a failure of one storage server so the system changed the policy to "Normal reconstruction" and started reconstructing data. After 30 minutes (60th minute), we started data deletion, which did not change the policy, but created tasks of a new load.

Figure 10.10 presents the throughput of user writes, maintenance tasks and data deletion on one of the storage servers. The expected throughput proportions (writes–maintenance–deletion) should be 100%–0%–0% in minutes 0-30, 71.5%–28.5%–0% in minutes 30-60, and 50%–20%–30% after the 60th minute. The

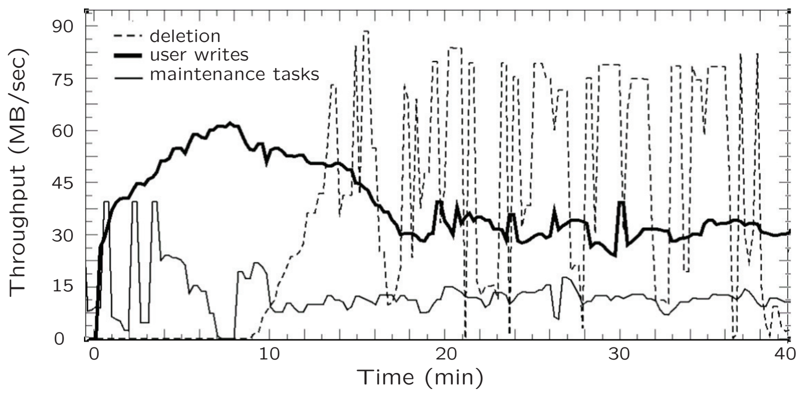

Figure 10.11: The throughputs of maintenance tasks, user writes, and data deletion—60-server configuration.

obtained proportions differ from the expected by less than 2% – this error comes from the load fluctuations.

Figure 10.11 presents the throughput achieved by user writes, maintenance tasks, and data deletion in case of the 60-server configuration. Such system generates more irregular data deletion load in comparison with the 6-server configuration. Here the expected throughput shares (writes–maintenance tasks–deletion) were 30%-10%-60%. In this experiment data deletion, which is an irregular load, is given a large allocation of the throughput because we want to verify the system behavior in case of irregularities in multiple workloads. The user load (at the plot is averaged) is also highly irregular. Additionally 4 servers were killed in order to obtain the 0 resiliency level. As we can see in Figure 10.11, the deletion tasks are highly irregular and their throughput fluctuates heavily. Nevertheless, the average throughput of these deletion tasks is consistent with the required throughput shares. Even if the loads fluctuate, our mechanism is able to stably keep the correct throughput proportions.

The settling time of the mechanism is around 3-4 minutes. This is the consequence of the duration of the longest unbreakable task, and in case of the system with shorter tasks it can be strongly reduced. For example, in systems where the duration of the tasks is of the order of 100ms, by executing the algorithm more frequently, and adjusting the defuzzyfication controllers (the standard techniques of choosing the controllers parameters are described in [133]), we could obtain the convergence time of the order of 10-15s. There is a trade-off between the speed of convergence of the throughput division and the system stability (see [133]); a too aggressive control would expose the same problem that fair queuing suffers from (see Section 10.3.2). Also, the 3-minutes settling time is not a problem for the storage systems because the target shares change rarely (once in several hours). In comparison with the duration of the maintenance tasks, the convergence time is negligible.

10.7 Conclusions and Future Work

We presented a novel mechanism for dividing resources among tasks of different load types. The new approach is based on an abstraction of the tasks and avoids assumptions about their characteristics. Therefore, it is suitable for distributed systems, where standard methods of defining models fail due to complex system architecture. The mechanism was implemented in the commercial system HYDRAstor, with the focus on achieving high performance of the controlled system. We theoretically and experimentally proved that the throughput proportions in the controlled system converge to the desired ones and confirmed that in a steady state the algorithm keeps the system saturated, which means high utilization of the resources. According to the experiments, the controlled system is stable (there are neither serious fluctuations of throughputs nor overshoots). Also, the reaction time is suitable for the storage systems' workloads. The evaluation was done on a 60-server system with the write performance of 10GB/s.

We showed how existing modifications of fair queuing can be adapted for resource division between user load and background tasks. We compared our new fuzzy adaptive mechanism with fair queuing algorithms. The evaluation was done on an artificial model which was a simplification of the HYDRAstor environment. The conclusion from the comparison was that although fair queuing mechanisms have better reaction time to changing target shares, they may behave unstably in case of irregular user load. Fuzzy adaptive control has, on the other hand, longer settling time, but it is more robust to irregularities in the load pattern. In the HYDRAstor, policies are changing quite rarely so settling time of fuzzy adaptive mechanism is acceptable. We concluded that the fuzzy adaptive mechanism is more suitable for our system.

Future work will concern changing the defuzzyfication controller during runtime. It would be changed depending on the kind of background tasks, in order to reduce the convergence time in case of shorter tasks. Our research will also focus on using automatic mechanism for choosing throughput shares at runtime.

Chapter 11

Optimizing Replica Placement for P2P Backup on Heterogeneous, Unreliable Machines

P2P architecture is a viable option for enterprise backup. In contrast to dedicated backup servers, nowadays a standard solution, making backups directly on organization's workstations should be cheaper (as existing hardware is used), more efficient (as there is no single bottleneck server), and more reliable (as the machines could be geographically dispersed).

We present an architecture of a P2P backup system that uses pairwise replication contracts between a data owner and a replicator. In contrast to a standard P2P storage systems using directly a distributed hash table (DHT) [74], the contracts allow our system to optimize replicas' placement depending on a specific optimization strategy, and to take advantage of the heterogeneity of the machines and the network. Such optimization is particularly appealing in the context of backup: replicas can be geographically dispersed, the load sent over the network can be minimized, or the optimization goal can be set to minimize the backup/restore time. However, managing the contracts, keeping them consistent, and adjusting them in response to dynamically changing environment is challenging.

We built a scientific prototype and ran experiments on 150 workstations in our university's computer laboratories and, separately, on 50 PlanetLab nodes. We found out that the main factor affecting the performance of the system is the availability of the machines. Yet, our main conclusion is that it is possible to build an efficient and reliable backup system on highly unavailable machines (our computers had just 13% average availability).

11.1 Introduction

Large corporations, medium and small enterprises, universities, research centers, and common computer users are all interested in protecting their data against hardware failures. The most common approach to protecting data is to keep the backup copies on tape drives, specially designated storage systems, or to buy cloud storage space. All such solutions are highly reliable, but also expensive. In 2013 the costs of renting 1TB of cloud storage per year from Amazon, Google, Rackspace, or Dropbox was approximately \$1000. Additionally, for some organizations, internal data handling policies require that data cannot be stored externally. The price of a single backup server with raw capacity of 14TB often exceeds \$12,000. A tape-based backup system for 14TB costs about \$7,000. These figures do not include additional costs of service, maintenance, and energy. With a large number of workstations that must be replicated at a single server, the server may become a bottleneck and may not be able to provide satisfactory throughput. Also, performance can be further degraded by network congestion. More scalable solutions exist, but are even more expensive. Yet, the market for the backup solutions is vast. DataDomain, a company providing modern backup systems, had in 2009 over 3.000 customers and over 8.000 systems deployed [261]. In the same year, the company was bought by EMC for \$2,4 billion.

There is still a need for cheaper alternatives for enterprise backup. On one hand, a significant research effort focuses on optimization techniques for dedicated backup servers, such as deduplication techniques [84,206] or erasure codes [228,236]. On the other hand, a P2P architecture can be explored in the context enterprise backup. Common PCs are cheaper than reliable servers. Also, in many cases, the unused disk space on desktop workstations can be used without additional costs (Adya et al. [4] discovered a tendency that the unused disk space on the desktop workstations is growing every year; the Moore's law for hard disks capacities, first formulated by Kryder [299] still holds). The bandwidth of the nodes connected in a distributed way scales much better than of a single server as the network load is more evenly distributed causing less bottlenecks. The system can take advantage of the geographical dispersion of the resources, thus offering better protection in case of theft or natural disasters (e.g., fire or flood). Finally, P2P solutions have already proved to work well in enterprise environments (GFS [109], MapReduce [78], Astrolabe [255], DHT [74] used in HYDRAstor [89] described also in the previous chapter, etc.).

Indeed, many P2P storage systems have been already built [10,34,37,49,62,171, 193,215,264,291,312]. The deduplication techniques get adapted for P2P storage systems [237,307]. There are new erasure codes more suitable for P2P systems [228]. Finally, there are many theoretical models for data placement optimizing data availability [22,28,59,86,234,256,262] and backup/restore performance [235,290] in P2P storage systems. However, real systems do not fully take advantage of the P2P architecture. There is a gap between theoretical models and real implementations. There are systems (e.g., OceanStore [171] and Cleversafe) that distribute data between geographically remote servers. These systems could be used for backup, but they both use dedicated servers, which stays in contradiction with our primary goal of creating a cheap backup system based on existing, unreliable machines.

There are P2P storage systems designed to work on unreliable machines; perhaps the most known such a system is Farsite [34]—a 6-years long Microsoft's project. However, Farsite offers much more than a simple backup. As a complete distributed file system, Farsite must deal with parallel accesses to data, must manage the file system namespace and ensure that frequently accessed data is highly available. Such requirements force additional complexity and many architectural limitations that do not exist in case of a backup system. On the other hand, since data backup is not a primary use-case, Farsite does not focus on implementing replica placement strategies (e.g., geographical dispersion of replicas or ensuring that data is backed up within a given time window, etc.). For more discussion on P2P storage systems we refer the reader to Section 11.2.

Bridging the gap between many theoretical models [22,28,59,86,234,235,256,262, 290] and prototype implementations, we asked the following question: Is it possible to implement various data placement strategies, focusing on the case where the machines are unreliable? Certainly, there are more challenges than in the case of centralized or highly-available systems. The machines' unreliability, and perhaps low availability, requires data locations to change dynamically. Is it difficult to continuously optimize the data placement with such assumptions? And, finally, is it difficult to take advantage of the machines' and network heterogeneity?

Our main contribution is the following: (i) We present an architecture of a prototype storage system that uses pairwise (bilateral) replication contracts for storing data and (ii) we show that we can efficiently manage the contracts and ensure efficient backup even under significant peers' unavailability. Our scientific prototype is evaluated in a real distributed environment.

We built a scientific prototype that replicates user data on different workstations of the organization. In our prototype, the machines that enter the system besides the standard activities also keep replicas of data of other peers. We assume that the workstations are heterogeneous and prone to failures. In particular, (i) the hardware might be heterogeneous and inefficient; (ii) the workstations may have variable amount of unused disk space (the space that is available for keeping replicas); (iii) the workstations are not always available—computers might stay powered on, or be powered off when not used by anyone (transient failures); (iv) the machines may experience permanent failures after which it is not possible to recover the data stored on a machine.

In contrast to fixed data placement (storing data in a DHT [10,37,49,215,312]), our replication is based on storage contracts between an owner of the data and its replicators. A contract for storing a data chunk of owner i on replicator j is a promise made by j to keep i's data chunk for a certain amount of time. Until the contract expires, it cannot be dropped by j (but it can be revoked by i). Since every data chunk is associated with a list of storage contracts, each chunk can be placed at any location (the location depends on the placement strategy). This contract-based architecture can be exploited in two ways. First, the contracts form an unstructured, decentralized architecture enables one to optimize replica placement, making the system both more robust and able to take advantage of the network and hardware configuration. Second, contracts also allow strategies for replica placement that are incentive-compatible, such as mutual storage contracts [69,262]. To the best of our knowledge, all previous literature on mutual contracts focused on theoretical analysis only. We complement these theoretical works by presenting an architecture (and a sample implementation) of a contract-based storage system. Yet, in this chapter, for the sake of concreteness, we focus on optimization of replica placement for P2P backup in a single organization, where incentives are not needed.

We have implemented a prototype. We tested our prototype on 150 computers in students' computer laboratories at the University of Warsaw; and on 50 machines in PlanetLab. The lab environment might be considered as a worst-case scenario for an enterprise network, as the computers have just 13% average availability and are frequently rebooted. Moreover, we assumed that all the local data is modified daily.

The results of our work show that: (i) In a P2P backup system we are able to efficiently transfer data chunks as the bandwidth of such a system scales linearly with the number of machines. (ii) Even on machines with very low availability we are able to efficiently optimize the placement of the replicas. We verified two different placement strategies (where the optimization goal was either to finish the backup of each data chunk within required backup window, or to enforce a certain geographical dispersion of the replicas) in two different settings. This leads to our main conclusion: (iii) It is possible to create an efficient P2P backup system and to take advantage of the peers' heterogeneity. (iv) Hardware unavailability significantly degrades backup performance; because of unavailability the time needed for direct communication of two peers can be long (in our experiments 20h, on average). We call this effect the *cost of unavailability*. Our measurements confirm the simulation results of Sharma et al. [266] and Tinedo et al. [288].

Since our results are supported not only by the simulations, but also by measurements of an implementation on a real system, we consider them as the proof of the concept that an efficient P2P backup systems can be created and that the heterogeneity of the machines in such a system can be exploited.

11.2 Related Work

In this section, we review related commercial projects and scientific approaches to data replication in distributed systems.

HYDRAstor [89] and Data Domain [313] are commercial distributed storage systems, which use data deduplication to increase amount of the virtual disk space (for the analysis of resource management in HYDRAstor we refer the reader to Chapter 10).

Many papers analyze various aspects of P2P storage by either simulation or mathematical modeling. Usually, the analysis focuses on probabilistic analysis of data availability in the presence of peers' failures (see, e.g., the work of Bernard and Le Fessant [22]). Douceur et al. [86], similarly to our system, optimize availability of a set of files over a pool of hosts with given availability: theoretical as well as simulation results are provided for file availability. Chun et al. [59] studied by simulation durability and availability in a large scale storage system. Bhagwan et al. [28] and Rodrigues and Liskov [256] show basic analytical models and simulation results for data availability under replication and erasure coding. Toka et al. [290] find a schedule of transfers which minimizes the restore time. They also analyze the impact of the size of the set of replicators on restore time. Pamies-Juarez et al. [235] studied the impact of the redundancy on the data retrieval time. This chapter complements these works by presenting a software architecture that allows to implement placement strategies, by considering other measures of efficiency, and by proving that various optimization strategies can be used in an unreliable environment.

As the focus of this work is on data backup in a single organization, we do not analyze incentives to participate in the system. However, to store the data, our system relies on agreements (contracts) between peers. In contrast, in DHT-based storage systems contracts are (implicitly) made between a peer and the system as a whole. Thus, our architecture naturally supports methods of organization that emphasize incentives for high availability, such as mutual storage contracts [69,262] (also these using asymmetric contracts [234]). It is worth mentioning that some papers explore the social interconnections while choosing the replica locations [291]; the tradeoffs between the redundancy, data availability and the ability to place data on the trusted nodes is analyzed by Sharma et al. [266] and Tinedo et al. [288]. These methods can also be adopted for our system.

Many P2P file systems [10,37,49,215,312] use storage and routing based on a DHT [74,260]. The address of the block, which is a hash of its content, fully determines the locations of its replicas. Thus, such architectures are less suitable for balancing the load on replicating workstations, or for optimizing the placement of the replicas. While these solutions focus on consistency of the data modified by multiple users, this chapter focuses on the issue of the best replication of the data.

To optimize the placement of replication contracts, we use a distributed optimization protocol relying on a distributed priority queue maintained by all the peers. An alternative is to use other load balancing or optimization techniques, such as Messor [211], T-Man [148] or pair-wise exchanges [280].

OceanStore [171] and Cleversafe [63] offer the idea of spreading replicas into geographically remote locations achieving the effect of deep archival storage. These systems combine software solutions with a specially designed infrastructure that consists of numerous, geographically-distributed, servers. The main contribution of these systems, from our perspective, is the resignation from a common DHT and the introduction of a new assumption that any piece of data can be located at any server. These systems, however, do not discuss the issue of replicating data on ordinary workstations (which are, in contrast to servers, frequently leaving and joining the network) and do not present any means allowing to handle such dynamism.

Wuala [193] moved one step further by proposing a distributed storage based not only on a specially dedicated infrastructure, but also including a cloud of workstations of users who install the Wuala application. However, since late 2011, Wuala no longer supports P2P storage. The idea of using a hybrid architecture of central servers and user machines, called peer-assisted backup, is also explored by Toka et al. [289]. Other P2P backup software include Backup P2P,[1] Zoogmo,[2] or ColonyFs.[3]

FreeNet [62] is a P2P application that exposes the interface of a file system. Its main design requirement is to ensure anonymity of both the authors and the readers. The underlying protocol relies on proximity-based caching. When a data item is no longer used, it can be removed from a caching location. Similarly, in Pangea [264] a replica is created whenever and wherever data is accessed. The idea of replicating data at locations near end users was successfully implemented by Akamai [194,226], the world largest content delivery network. Naturally, there are other works on proximity-based caching [143,180,192,267].

Farsite [34] was a Microsoft's 6-years long project aimed at creating distributed file system for sharing data between thousands of users. The retrospective view of the project gave us the feeling of following a good direction. Authors emphasize that: first, real scalability must face the problem of constant failures in the network; second, in a scalable system, manual administration must not increase with the size of the network. We followed both requirements when designing our system.

There are a few substantial differences between Farsite and our prototype implementation. Most importantly, Farsite's architecture does not rely on mutual contracts.

In Farsite updates of data are committed locally and the changes are appended to the log (similarly as in Coda [166]). The log is sent to a group of peers responsible for managing a subset of the global namespace (called the directory group). The group periodically broadcasts the log to all its members. As the directory group uses a byzantine fault tolerant protocol [85], no file can be modified if one third or more of the group is faulty. Since we consider a backup system in which data is modified only by the owner, we are able to gain in flexibility and robustness. In our asynchronous updates mechanism, every peer has an associated group of peers managing its asynchronous messages—we refer to such peers as *synchro-peers*. Synchro-peers are independent of replicas, which results in a desired property that

[1]sourceforge.net/projects/p2pbackupsmile/
[2]zoogmo.wordpress.com
[3]launchpad.net/colonyfs

any peer can keep replicas for any chunk of data. Thus, replicas can be chosen so that they constitute the most profitable replication group.

Farsite is a distributed file system so its complexity is higher compared to a backup system. However, Farsite is not a backup system, so it does not support backup-specific requirements like placing replicas in geographically distributed locations, or the optimization of the backup/restore time.

11.3 System Architecture

Our system uses a mixed architecture that stores control information, meta-data and data in three different ways. The control information that allows peers to locate and to connect to each other must be located efficiently—hence we use a DHT as a storage mechanism. In contrast, each peer is responsible for finding and managing peers who replicate its data (its *replicators*). The replication contracts enable us to optimize replica placement and thus to tune replication to a specific network configuration. The meta-data describing replication contracts are kept by both the data owner and the replicator. Chunks of data are kept in an unstructured overlay; concrete locations are described by the meta-data.

11.3.1 Control Information

The basic attributes of a peer are kept in a structure called *PeerDescriptor*. For each peer, its PeerDescriptor contains:

- identification information (the public key);
- information needed to connect to this peer (the current IP address and the port of an instance of our software running on the workstation) and the user account name in the operating system (account name is required by the current implementation of data transmission layer—see Section 11.3.3);
- identifiers of its *synchro-peers* (see Section 11.3.4).

PeerDescriptors of all peers are kept in a highly replicated DHT: peer's ID (a hash of its public key) is hashed to its PeerDescriptor. As the size of the control information is small, we are able to afford strong replication (compared to a generic DHT). Thus, instead of a single peer, a number of peers is responsible for keeping data hashed to a part of the key-space.

11.3.2 Replication Contracts

The main goal of our prototype is to support nontrivial replica placement strategies. We need to be able to store any replica at any peer. This architecture contrasts with content addressable storage systems that put each chunk of data under an address that is fully determined by the chunk's unique identifier (e.g., by the hash of its content). As a trade-off, for flexibility of placing replicas at any location, we need a mechanism for locating the data.

In our system each peer keeps information about replica placement of its data chunks in an index structure called *DataCatalog*. For each data chunk, the catalog stores information about: (i) identifiers of the peers that keep replicas of the data chunk (hereinafter *chunk replicators*); (ii) the size of the chunk; (iii) the version number of the chunk.

Additionally, each peer keeps information about data chunks it replicates. As each storage contract is kept in exactly two places (the owner and the replicator), contracts are consistent and it is easy to retrieve metadata (i.e., the DataCatalog) in case of a failure. Since peers are unreliable, the process of contract negotiation can break at any point, possibly leading to two types of inconsistency: an owner o believes j is its replicator, while j is not aware of such a contract; or a peer j believes to be o's replicator, while o is not aware of such a contract. Contract negotiation is however idempotent and because the contracts are kept both by the owner and by the replicator, such inconsistencies can be easily fixed. Each peer periodically sends messages to its replicators with the believed contracts (and versions of data chunks, which allows the replicators to update the chunks of data which are out of date). Each replicator periodically sends similar information to the appropriate owners. Detected inconsistencies can be resolved either by adopting the owner's state; or by always accepting a replication agreement.

The DataCatalog is stored in a file but it is not replicated between peers. In this way we avoid the additional overhead of updating the catalog at remote locations, when the contract for any chunk is changed. Since the machines are unreliable, in many cases we even would not be able to update the DataCatalog at a remote peer as the peer can simply be unavailable. On the other hand, both owners and replicators are aware of all their storage contracts. When the local data of any peer is lost, the peer (the owner) gossips the information about the failure. The replicators answer to the gossip message with the information about the contracts; the owner uses this information to rebuild the DataCatalog. Once the DataCatalog is reconstructed, the owner locates and rebuilds all missing data. Since the DataCatalog is persistent, its reconstruction is required only in the case of a non-transient failure; thus the reconstruction does not cause much overhead. The standard use cases of replicating a chunk of data and retrieving permanently lost data are depicted in Figures 11.1 and 11.2, respectively.

Alternatively, in an enterprise environment where a (replicated) server is an affordable option, the meta-data can be kept centrally (in primary memory for faster access). This solution is, however, less scalable.

We designed the mechanism that is responsible for relocating or additionally replicating data chunks that are weakly replicated according to a given abstract

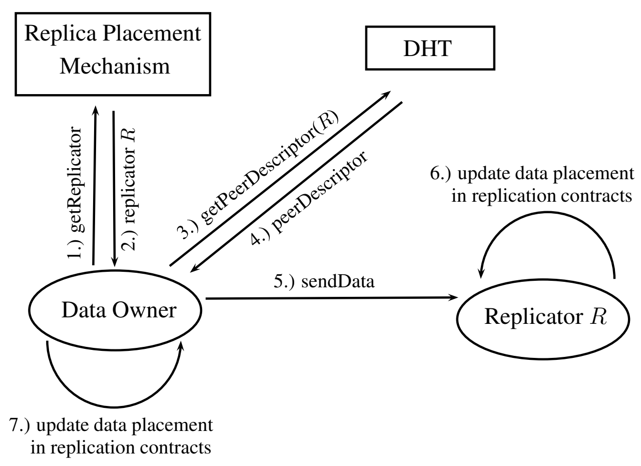
Replica Placement Mechanism
DHT
1.) getReplicator
2.) replicator R
3.) getPeerDescriptor(R)
4.) peerDescriptor
6.) update data placement in replication contracts
5.) sendData
Data Owner
Replicator R
7.) update data placement in replication contracts

Figure 11.1: The diagram showing consecutive steps performed when replicating a chunk of data. After asking the local replica placement mechanism for a replicator (steps 1 and 2), three different storage mechanisms are involved: (i) the peer descriptor is obtained from a DHT (steps 3 and 4 in the diagram), (ii) the data chunk is stored at the replicator (step 5), and (iii) meta-data (that contain storage contracts) is kept both by the data owner and the replicator (step 6 and 7).

metric. The specific metric used in our evaluation takes into account peers' availability, bandwidth, and geographic distribution. It tries to keep all but one replicas as close as possible (not to overload the network) and to keep one replica in a remote location for additional safety (for location-dependent failures, such as fire, flood, or theft). The metric also balances the load on the machines so that each data chunk can be replicated within the required *backup window* (the time requirement for each chunk to be backed up). The precise metric is described in Section 11.4.1. The optimization mechanism is based on hill-climbing—in consecutive steps each peer performs locally optimal changes of its contracts. The optimization of the location of each single data chunk can be performed even if large fraction of peers is unavailable; to perform a single local optimization steps we require only 3 peers to be available. Thus the mechanism can proceed in unreliable environments, where peers are often unavailable.

When nodes' parameters (e.g., their availability) change, or when a large number of nodes is added to the system, the contracts are renegotiated. If each such change resulted in data migration, the network and the hosts could easily become overloaded. Therefore the process of changing a contract is more elaborate. The contracts are allowed to change frequently but such changes do not require data migration. Such temporary contracts are periodically (e.g., daily) committed; after a contract is committed, the data is migrated. The complete mechanism involves some

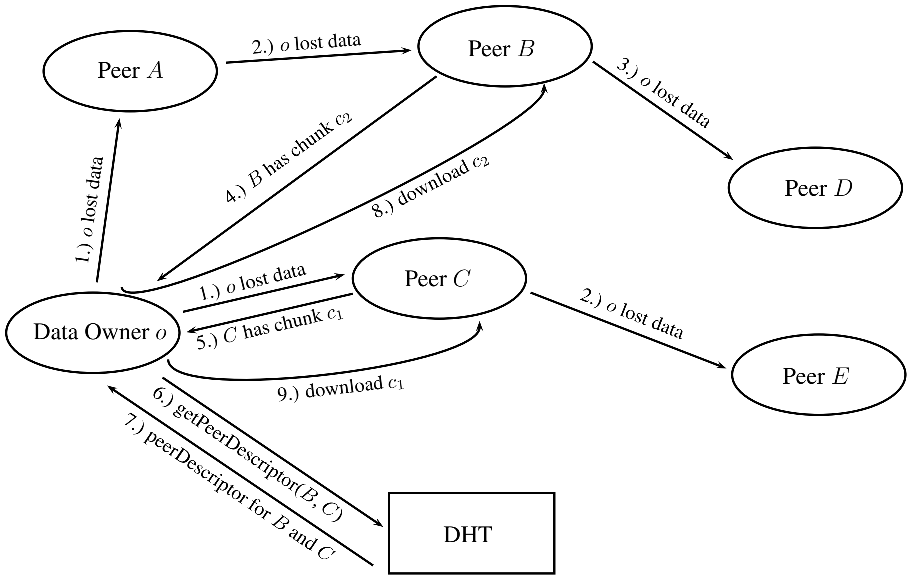

Figure 11.2: The diagram showing consecutive steps performed when retrieving a data after a crash failure (the worst-case failure when all the data, including the DataCatalog, is lost). When the DataCatalog, which is kept only locally, is lost, a message "o lost data" is gossiped (steps 1–3 in the diagram). Data replicators having data chunks of the given data owner answer this message. The replicator X having a chunk y answers with the message "X has chunk y" (steps 4 and 5). As a result, DataCatalog can be rebuilt and lost data can be located (steps 6 and 7) and retrieved (steps 8–9).

additional details as it must also take into account possible communication failures. The optimization mechanism is described in detail in Section 11.4.2.

11.3.3 Data Transmission and Updates

Every member of the network, before placing its replicas at a remote peer, must obtain this peer's permission. Once the peers reach an agreement, they mutually authorize each other using their *PeerDescriptor*'s identities (the public keys stored in the DHT).

The data is transmitted in an encrypted connection. In the current implementation, we use standard Linux tools for data transfers. Each peer runs an ssh daemon that acts as a server that accepts connections of data owners. When a peer initiates a connection to transfer its data, it uses an scp as a client.

When an owner modifies its local copy, the updated chunk must be propagated to the network. The replicators are informed of the changed versions of the data chunks through periodic control messages (version numbers are attached to the messages containing contracts sent between the owner and the replicator, described in the previous subsection). The unavailable peers are informed about the changes of data chunks through asynchronous messages, described below. Once the replicator finds there is a new version of a data chunk it replicates, it downloads the new version either from the owner or from the other replicators, achieving eventual consistency [295]. Note that if data owners were responsible for uploading the new versions to the replicators, a successful transfer would require both the owner and the replicator to be available. In our solution, the replicator is responsible for keeping replicas up to date so we only require that the replicator and any other replicator or the owner are available.

Unlike common backup systems, our system stores only the last version of each data chunk. A system storing many previous versions may be built in the same way as version control software (e.g., svn or git) often uses standard filesystems; more specifically, the previous versions (or the deltas) can be kept in the same data chunk; or the deltas can be kept in separate data chunks.

11.3.4 Asynchronous/Off-Line Messaging

We assume that the workstations may be unavailable for some time just because they are temporarily powered off. In contrast to many distributed storage systems (e.g., GFS [109]), in such a case our system does not rebuild the missing replicas immediately, in order not to generate unnecessarily load on other machines nor the network. Instead, when the unavailable peer eventually returns to the network, it efficiently updates its replicas. To inform the unavailable peers about the new version numbers of its replicas and about the contracts, we use asynchronous messaging. The control messages sent to the peer that is currently unavailable are cached at, so called, *synchro-peers*. We use the idea of group communication for synchronizing the messages within each (small) group of synchro-peers. As opposed to Defrance et al. [80], who present the mechanism of caching the messages on routers, we chose to design the concept of synchro-peers to limit the costs of additional hardware. A synchro-peer is just an additional process running on a standard peer in our network. The load imposed by the algorithm on the synchro-peers is low as we keep the messages small; thus it is relatively "cheap" for a peer to act as a synchro-peer for many nodes.

An asynchronous message from i to j is sent to the *synchro-peers* of j. Synchro-peers are peers, defined for every peer j (j is called in this context a *target peer*) that keep asynchronous messages for j. Synchro-peers of j include j, so every message will be delivered to the target peer by the same means as it is delivered to the other synchro-peers. Each synchro-peer periodically tries to send the asynchronous message to the synchro-peers that have not yet received the message; the IDs of synchro-peers that have not yet received the message are attached to the message (thus, the same message can be delivered multiple times to the same peer). These updates achieve eventual consistency.

The consecutive messages between any two peers are versioned with sequential numbers (logical clocks). If any synchro-peer k has not managed to send a message $m_1(i \rightarrow j)$ to all the requested synchro-peers before receiving a subsequent message $m_2(i \rightarrow j)$ with a higher version number, then the synchro-peer drops $m_1(i \rightarrow j)$, as the new message m_2 contains a superset of chunks' version numbers. Thus, the expected number of messages that are waiting for delivery on a single synchro-peer is bounded by $|R(\cdot)| \cdot |S(\cdot)|$, where $|R(\cdot)|$ is the average number of replicators per peer and $|S(\cdot)|$ is the number of synchro-peers per peer. The groups of synchro-peers are small (5 machines in our experiments), the messages contain only the version numbers of the data chunks (thus, the messages are small as well), and the old undelivered messages can be safely replaced with the newer versions of the messages. As a result, the mechanism of asynchronous messaging is cheap from the perspective of the system.

The target peer executes the commands from the messages immediately after the first reception, but it remembers for each sender the latest version number of the received message. This information protects against multiple execution of the orders from a single message. Figure 11.3 shows the idea of synchro-peers delivering asynchronous messages.

The mechanism of versioning through asynchronous messages reduces the amount of information replicators keep regarding the structure of replication contracts. In an alternative solution, replicators synchronize directly between each other. However, this requires replicators to know the IDs of all the other replicators for each data chunk they store. If this information is stored at the replicators, changes in replication contracts require multiple updates; if it is stored as meta-data, the size of the meta-data becomes proportional to the number of data chunks in the system. Moreover, exchanging the messages between all replicators is highly inefficient. With asynchronous messages, the owner keeps the information about its replication contracts, updates are simple, and meta-data is small.

Using asynchronous messaging has two advantages. First, the asynchronous message is delivered with high probability even when the sender is unavailable. Second, the data may be downloaded concurrently from multiple replicators.

11.4 Replica Placement

The goal of the replica placement policy is to find and dynamically adapt the locations of the replicas in response to changing conditions (new peers joining, permanent failures, changing characteristics of existing replicators). Finding possible locations for replicas is not trivial given that peers differ in availability and amounts of free

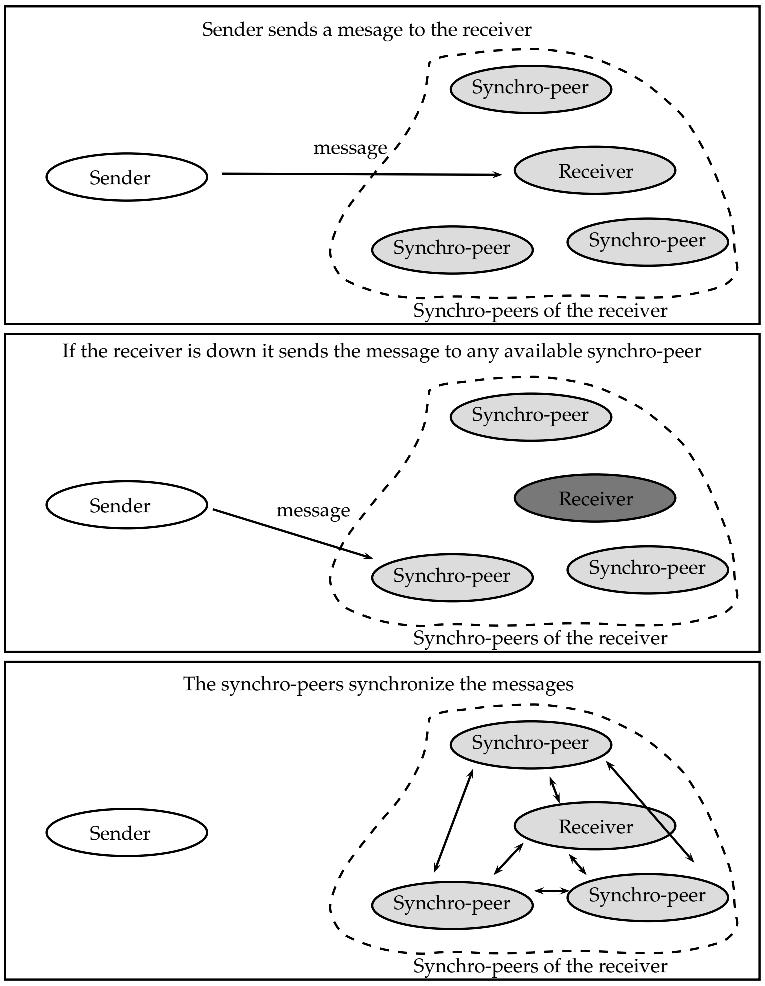
Sender sends a mesage to the receiver
Synchro-peer
message
Sender
Receiver
Synchro-peer
Synchro-peer
Synchro-peers of the receiver
If the receiver is down it sends the message to any available synchro-peer
Synchro-peer
Sender
message
Receiver
Synchro-peer
Synchro-peer
Synchro-peers of the receiver
The synchro-peers synchronize the messages
Synchro-peer
Sender
Receiver
Synchro-peer
Synchro-peer
Synchro-peers of the receiver

Figure 11.3: The diagram depicting the use case of synchro-peers delivering an asynchronous message.

disk space. Moreover, the replica placement policy should take advantage of peers' heterogeneity in terms of availability, geographic locations, etc..

Our policy consists of two main parts. First, a utility function (in short, *utility*) scores and compares replica placements. Utility is a function that for a given data owner and a set of possible replicators returns the score proportional to expected quality of replicating data. Second, a protocol manages replica placement in the network in order to maximize the utility of the currently worst placement (max min optimization).

11.4.1 Utility Function

In this section we describe our example utility function that takes into account peers' availability, bandwidth, and geographical distribution. The notation used in this and the following section is introduced when necessary, and summarized in Table 11.1.

A user of a backup application is interested in the resiliency level of her data (defined by the desired number of replicas N_r and their proper geographic distribution); and the time needed to retrieve the data in case the local copy is lost (expressed as the desired data read time $Des(T_r)$). Additionally, each user must be able to backup her data (propagate the local updates to replicas) during the time the user is online (this time is expressed as the backup window $Des(T_b)$).

Optimizing the backup time of the data is one of the key targets of backup systems. If backup is slower than data modification, then, in case of a failure, the data cannot be retrieved. A short backup window is also desired for easy system maintenance. As a further evidence, we note that many benchmarks comparing secondary storage systems focus on their throughputs as one of the primary metric. Similarly, many commercial storage systems advertise themselves as high-throughput [89,313].

Every peer j dedicates bandwidth B_j to the background backup activities (B_j is bounded by network and disk bandwidth, but can be further reduced by the user).

Utility function $U : \mathcal{P} \rightarrow \mathcal{R}$ is a scoring function mapping a replica placement $P_k \in \mathcal{P}$ to its score $u_k = U(P_k)$. Hereinafter, by $R(d_k)$ we denote the set of replicators of data chunk d_k.

The utility function is a sum of utilities expressing geographic distribution U_{geo}, backup time (performance) U_{perf}, and the number of replicas $|R(d_k)|$ (with the latter two treated essentially as constraints):

$$U(P_k) = U_{geo}(P_k) - L \cdot ||R(d_k)| - N_r| - M \cdot U_{perf}(P_k), \tag{11.1}$$

where M and L are (large) scaling factors. L penalizes for insufficient number of replicas. M penalizes for backups that cannot be finished within the time window. If the backup cannot be done on time, it means that for some data we can give no resiliency guarantees and, so, even very good geographic distribution properties are useless.

U_{perf} is computed as follows. The average duration of data backup to replicator j, $E(T_b, j)$ is estimated by:

$$E(T_b, j) = \frac{\sum_{k:j \in R(d_k)} size(d_k)}{p_{av}(j) B_j} \tag{11.2}$$

The backup duration is proportional to the congestion on the receiving peer $\sum_{k:j \in R(d_k)} size(d_k)$; and inversely proportional to the bandwidth B_j that peer j dedicates for the background backup activities. We use a simplified model that does not explicitly consider the network congestion, but this issue is addressed in the further part of the utility function, described in this section. Moreover, successful

Table 11.1: The summary of the notation used in Sections 11.4.1 and 11.4.2.

Symbol	**Meaning**
N_r	desired number of replicas
$Des(T_r)$	desired data read time
$Des(T_b)$	backup window (desired data backup time)
B_j	bandwidth that the j-th peer dedicates to backup activities
$R(d_k)$	replicators of data chunk d_k
P_k	replica placement of data chunk d_k (replicators and data owner)
$u_k = U(P_k)$	utility of placement of replica for data chunk d_k
$size(d_k)$	size of data chunk d_k
U_{perf}	part of the utility u_k expressing backup time (performance)
U_{geo}	part of the utility u_k expressing resiliency due to geographical distribution of the replicas
L, M	weights denoting the impact of $\|\|R(d_k)\| - N_r\|$ and U_{perf} on u_k, respectively
$E(T_b, j)$	average backup time from the considered peer to the j-th peer
$E(T_r, j)$	average restore time from the j-th peer to the considered peer
$remote_{min}$	minimal required TTL distance between considered peer and most distant replicator
$remote_{max}$	maximal required TTL distance between considered peer and most distant replicator
$close_{max}$	maximal required TTL distance between considered peer and all but most distant replicator
$jmax$	replicator from $R(d_k)$ that is most distant to the considered peer
$dist_{TTL}(j)$	TTL distance between the replicator j and considered peer
N	estimated number of peers in the network
α	desired number of messages that peer wants to get in a time unit without being overloaded

write on peer j is possible only when peer j is available (hence $p_{av}(j)$). Assuming that the restoring operation has no priority over the backup, the average data restoration time $E(T_r, j)$ is computed in the same way.

The backup time penalty $U_{perf}(P_k)$ is the sum over the utilities per replica location:

$$U_{perf}(P_k) = \sum_{j \in R(d_k)} U_{perf}(j),$$

where $U_{perf}(j)$ penalizes for insufficient backup window on the j-th replicator:

$$U_{perf}(j) = \min(Des(T_b) - E(T_b, j), 0) + \min(Des(T_r) - E(T_r, j), 0)$$

To compute U_{geo}, we approximate the geographical distribution of the data by TTL values. Most of the replicas should be near the owner to reduce the network usage; $close_{max}$ denotes the desired distance for the "nearby" replicas. However, to cope with geographically correlated disasters, one replica should be far: its distance should be between $remote_{min}$ and $remote_{max}$.

The "geographic" utility $U_{geo}(P_k)$ considers both "near" and "far" replicas:

$$U_{geo}(P_k) = \min(0, dist_{TTL}(j_{max}) - remote_{min}) + \min(0, remote_{max} - dist_{TTL}(j_{max}))+ \\ + \sum_{j \in P_k - \{j_{max}\}} \min(0, close_{max} - dist_{TTL}(j)),$$

where j_{max} denotes the replicator from $R(d_k)$ that is most distant to the data owner, and $dist_{TTL}(j)$ denotes the TTL distance between replica j and the data owner.

In an enterprise backup system, we assume that all data chunks are equally valuable. Thus, the utility of the whole system is the utility of the worst placement ($\max \min_k u_k = \max \min_k U(P_k)$).

11.4.2 Distributed Optimization Protocol

When designing our system, we have considered several approaches for maximizing system utility $\max \min_k u_k$. Perhaps the most straightforward idea is that each peer optimizes its own utility, u_k, by deciding with whom to form replication contracts [69,262]. Game-theoretic strategies would give the system extra protection against malicious spammers, but they have the following drawbacks: (i) every peer has to compete with the other participants; in effect, peers with low availability or bandwidth could never achieve satisfactory replication; (ii) even if contracts for low-quality peers are accepted at the cost of rejecting the contracts of the high quality peers, such frequent contracts rejections would result in protocol inefficiencies. Considering these drawbacks, and taking into account that in a single enterprise peers should cooperate to achieve the social optimum, we decided to turn to a cooperative (not in the game-theoretic sense), proactive approach described below.

Every peer i with free storage space periodically chooses a data chunk d_k with low utility, and proposes a new replication agreement with the data chunk's owner o (peers share information on low utility data chunks in a distributed priority queue). The owner either tentatively adds i to its replication set $R(d_k)$ (if the number of replicas $|R(d_k)|$ is lower than the desired resiliency level N_r); or tentatively replaces $j \in R(d_k)$, one of its current replicas, with i (all possible $j \in R(d_k)$ are tested). If the resulting utility $U(P'_k)$ is significantly higher than the current value $U(P_k)$, the owner o tries to change the contracts (see the next section). In our experiments, we require $U(P'_k)$ to be higher than $U(P_k)$ by at least 10% to reduce data movements that do not significantly improve data distribution. When the owner rejects the proposition, or when it is unavailable, i puts o in a (temporary) taboo list in order to avoid bothering it later with the same proposal.

As a result of continuous corrections of the replica placements, each peer can end up having replicas of different chunks at different peers. Such a machine has many replicators and their monitoring becomes expensive. However, the monitored information (the availability and the size of replicated data) are gossiped; thus the cost of distribution of information is independent of the number of replicators. On the other hand, storage contracts with multiple peers allow to parallelize data transfers, and the cost of replica rebuilding is amortized.

If every peer proposed storage for the owner of the data with the lowest utility, the owner would get overloaded with storage offers (and the remaining data chunks would be ignored). Therefore, each peer sends a message to o with probability p_p such that $p_p = \frac{\alpha}{N}$, where N is the estimated number of peers in the network, and α is the desired number of messages that a peer wants to get in a time unit without being overloaded. Given such probability, the expected value of the number of messages, E_m, the owner of data gets in a time unit is: $E_m = p_p \cdot N = \alpha$.

The system keeps the data chunks with the lowest utility in a distributed priority queue. In our prototype, we implemented the distributed priority queue by a gossip-based protocol. Each peer keeps a fixed number of data pieces with the lowest priorities. It updates this information with its own data pieces and distributes the information to the randomly chosen peers.

11.4.3 Changing Replication Contracts

The contracts in our system are continuously renegotiated. To not to overloaded the hosts nor the network, every such change should result in data migration. A mechanism for efficient changes of the contracts are described below.

Finding the Best Replicators

Below, we describe two aspects of the protocol: revocation of inefficient contracts; and recovery from transient failures.

When peer i offers its storage to data owner o, and when o decides that i should replace one of its existing replicators j (as the resulting value of the utility function U is higher), then o has to explicitly revoke the contract with j. Thus, changing location of the data of o, from i to j, requires these three peers being on-line. The example below illustrates why revocation of the contracts cannot be realized asynchronously.

Example 11.1. Consider peer j storing many data chunks of several owners. From the perspective of each owner, as j is comparably overloaded, any new peer joining the network is a better replicator than j. If the contracts could be revoked asynchronously, all the peers would revoke the contract on j during its unavailability. Now j, having no data, can take over all data stored at some other peer k during k's unavailability, by offering storage space to all the data owners replicating their data at k. Such situation can repeat indefinitely. Each peer is not aware that j has already revoked some of its contracts and that it is not overloaded any more.

However, if the existing replicator j has low availability, on-line revocation of its contract is also improbable. Thus, an owner can revoke a contract with such a low-available replica (e.g., the first decile of the population) also through an asynchronous message.

Additionally, since the peers are unreliable, the process of contract negotiation can break at any point leading to inconsistency of the contracts. Two types of inconsistency are possible: an owner o believes j is its replicator, while j is not aware of such contract; or a peer j believes to be o's replicator, while o thinks j is not. The inconsistent contracts can be easily detected by periodically exchanged messages and fixed by adopting the owner's or the replicator's state.

Committing Contracts and Transferring Data

In order to reduce the load on the network, replicators cannot change too often; but to maintain high performance, replicators must eventually follow the negotiated contracts. A *non-committed* contract between an owner and a replicator is negotiated, but no data has been transferred. Contracts are committed periodically. For each data chunk, if there is a new contract (negotiated, but not committed), the contract is committed when the time that passed since the last committed contract for this chunk is large enough (e.g., 24 hours). This guarantees that the data is transferred at most once in each time period (e.g., at most once every 24 hours). However, even when (non-committed) contracts change often, data is replicated (as the committed contracts represent a snapshot of utility optimization).

After committing a contract, the owner sends a message to the new replicator that requests data transfer. As soon as the new replicator downloads requested chunk, it sends an acknowledgment to the owner. Finally, the owner notifies the old location to remove the chunk.

11.5 Experimental Evaluation of the Prototype

11.5.1 Experimental Environment

We performed the experiments in two environments: (i) on the computers in the student computer labs at the University of Warsaw; and (ii) on PlanetLab. The aim of choosing these two environments was not to compare them, but rather to show how our system uses and reacts to different degrees of heterogeneity and different problems present in these systems (geographical decentralization in PlanetLab and low availability in student computer labs). We run the prototype software for over 4 weeks in the labs and for 3 weeks on PlanetLab. Each computer acted as a full peer: owned some data and acted as a replicator. The data was considered as modified at the beginning of each day; thus each day we expected the system to perform a complete backup. If the transfer of a particular data chunk did not succeed within a day, the following day we transferred a newer version of the chunk. In both test environments we used chunks of a equal sizes (50MB).

The computers were centrally monitored; the central monitoring server experienced several failures which slightly influenced the presented results (the real backup times are slightly shorter than presented).

Students Computer Lab

We run our prototype software on all 150 machines of the students' computer lab. The availability pattern might be considered as a worst case scenario for an enterprise setting. The lab is open each week, from Monday to Friday, between 8:30am and 8pm, and on Saturdays between 9am and 2:30pm. The students frequently (i) switch off or (ii) reboot machines; each day at 8pm the computers are (iii) switched off by the administrators (the machines are not automatically powered on the next day); each of these events was considered a transient failure.

The computers in students lab have very low average availability (the median is equal to 13%). Figure 11.5 presents the distribution of the availabilities of the computers in the lab. Figure 11.6 presents the distribution of the up time of the computers and the time between their consecutive availability periods within a single day (the nights are filtered out). Low availability coupled with long session times constitute a worst-case scenario for a backup application: in contrast to short, frequent sessions, here machines are rather switched on for a day, then switched off when the lab closes, and remain off for a long time.

The amount of local data was sampled from the distribution of storage space used by the students on their home directories. The students in our faculty are divided into three groups and each student is assigned a quota that depends on his or her group affiliation. The distribution of data sizes for the three groups are presented in Figure 11.4. We took the distribution of the sizes for the group with the highest quota and scaled this distribution so that the average value was 3GB. Thus the sizes of local data were varying approximately between 0 and 8GB.

The local storage space depended on machines' local hard disks; and varied between 10GB (50% machines), 20GB (10% machines), and 40GB (40% machines).

PlanetLab

The experiments on PlanetLab were conducted using 50 machines scattered around Europe. Each machine was provided with 10GB of storage space and had 1GB of local data intended to be backed up. The machines were almost continuously available (the median availability is equal to 0.91).

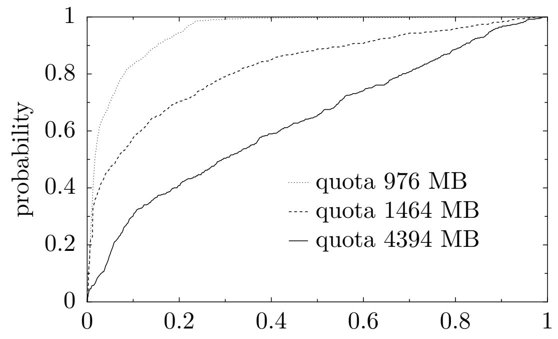

Figure 11.4: The cumulative distribution functions of the sizes of storage space used by the students on their home directories. The students in our faculty are divided into three groups and each student is assigned a quota that depends on his or her group affiliation. Three lines describe cumulative distribution of these three groups of students. The size of the data is presented on the horizontal coordinate. We scaled this size so that the three distributions had the same maximal value equal to 1.

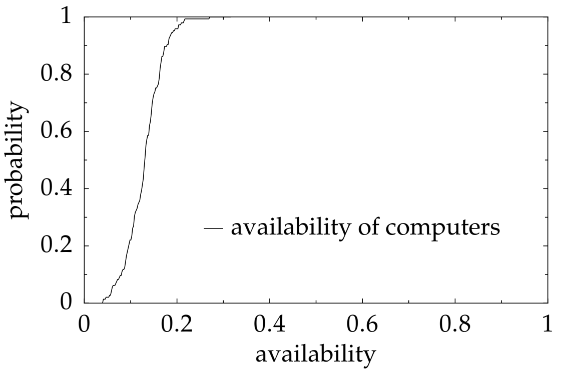

Figure 11.5: The cumulative distribution function of the availability of the computers in students lab.

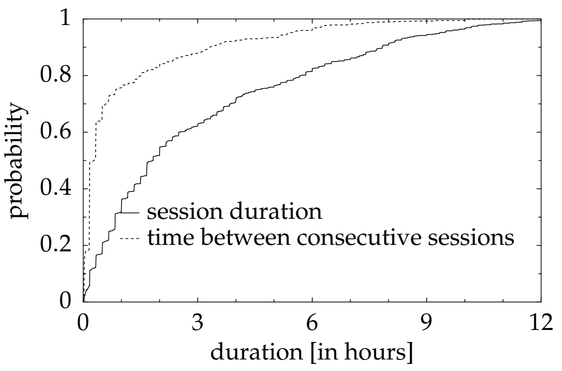

Figure 11.6: The cumulative distribution function of the session durations and the times between consecutive sessions for the computers in students lab (the nights are filtered out).

11.5.2 Asynchronous Messages

In this subsection we present how the asynchronous messaging influence message delivery time and the probability that the message is delivered. For our analyzes we used traces of availability from the students computer lab. We varied the number of synchro-peers per peer between 0 and 30. For each number of synchro-peers, we generated 100,000 messages with a randomized source, destination and the sending time. Figures 11.7 and 11.8 show the results.

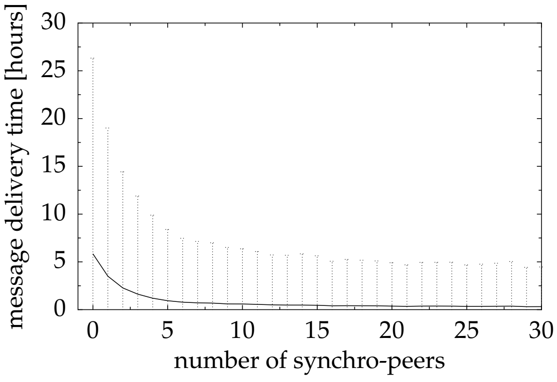

Figure 11.7: The dependency between the number of synchro-peers and the delivery time of the asynchronous message. Delivery time measured from the first time of availability of the receiver after the message was sent.

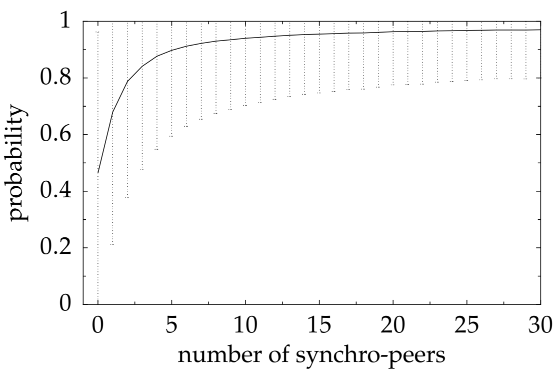

Figure 11.8: The dependency between the number of synchro-peers and the probability of delivery a message to any synchro-peer.

Figure 11.7 shows dependency between the number of synchro-peers and the delivery time of the message. Because the message delivery can be accomplished only when the receiver is active, we present delivery time measured starting from the first online appearance of the receiver after the message was sent. Ideally, the message should be delivered just after the receiver becomes online, resulting in a low delivery time. Our results show that the delivery time decreases significantly when using synchro-peers. Additionally, the standard deviation decreases even more significantly (high standard deviations are caused by peers that have low availabilities). If the number of synchro-peers is higher than 5, the advantage of using more of them becomes less significant. Taking into account that the higher number of synchro-peers results in higher number of messages required for synchronization, we decided to use 5 synchro-peers in the remaining experiments.

Figure 11.8 shows dependency between the number of synchro-peers and the probability of a successful delivery of a message to any synchro-peer. We are interested in calculating such probability because a message delivered to a synchro-peer is, in fact, a replica of the original message. Thus, synchro-peers should enable message delivery even in case of long term absence of the sender (e.g., caused by a non-transient failure). The results show that synchro-peers significantly increase this probability: with 5 synchro-peers the system delivers 90% of the messages, while without synchro-peers, more than half of the messages are lost.

	Utility (weighted replicated data)		
day	average	std dev	(std dev)/average
1	34487	6086	0.18
2	60489	8141	0.13
3	69658	5496	0.08

Table 11.2: The utility, i.e. the ratio of total size of replicated data (in MB) to the availability for the first 3 days of experiments in the lab environment.

11.5.3 Replica Placement

Students Computer Lab

The goal of the tests in the labs was to verify how the system copes with low availability of the machines. For each machine i, we set the bandwidth B_i to the same arbitrary value and the backup window $Des(T_b)$ to 0. As all the machines are in the same local network, there is no geographical distribution of the data. Thus, the utility function (Eq. 11.1) degrades to the number of replicas and the backup duration (Eq. 11.2). This means that the system minimizes the maximal time required for transferring a data chunk, which means minimizing the load on the maximally loaded machine. As a result, we expected the machines to be loaded proportionally to their availabilities. By Eq. 11.2, the load on the machine is proportional to the size of data it replicates; thus, for each machine the total size of replicated data should be proportional to the machine's availability. Additionally, storage constraints should influence the amount of data stored.

During the first 3 days of experiments we measured the ratio of the total size of data replicated by a peer (in MB) to the peer availability. For each day we considered only the peers that were switched on at least once. We also restricted the measurements only to peers with at least 9GB storage space (that could accommodate, on the average, 3 replicas), to separate the effect of insufficient storage space. The average values and the standard deviations of the ratios for the 3 days are presented in Table 11.2. The standard deviation is low in comparison to the average (the deviations are 18%, 13% and 8% of the corresponding average) which shows that the replicas were distributed according to our expectations.

PlanetLab

The goal of the PlanetLab tests was to verify how our system handles geographic distribution and heterogeneity of the machines. In this environment we required the far replicas (i.e., that implementing geographical dispersion) to have TTL distance from the owner in range $\langle 3, 8\rangle$ ($remote_{min} = 3$ and $remote_{max} = 8$), and the other replicas to be as close to the owner as possible ($close_{max} = 0$). Additionally we

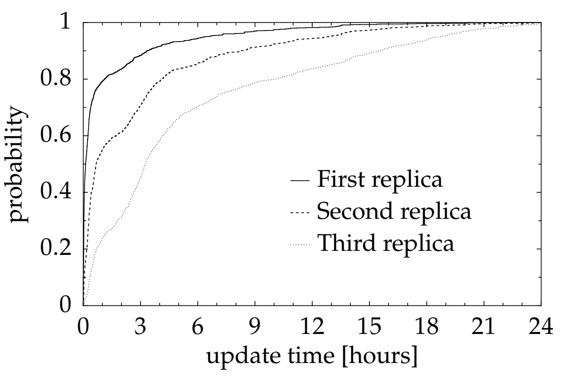

Figure 11.9: The cumulative distribution function for the time of creating a replica of a single data chunk. The time is relative to the data owner. Students' Lab; data collected over 4 weeks of running time.

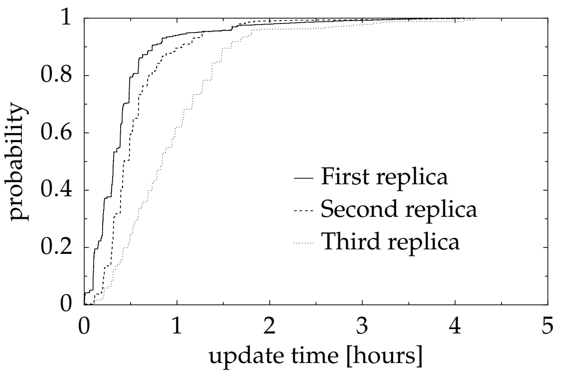

Figure 11.10: The cumulative distribution function for the time of creating a replica of a single data chunk. The time is relative to the data owner. PlanetLab; data collected over 3 days.

set the bandwidth B_i to 500 KB/s for half of the machines, and 1000 KB/s for the other half. We also set the backup window $Des(T_b)$ to 4500s. Each machine had the same amount of local data (1GB); the disk space limit was 4GB. We expected that the low-bandwidth machines will be less loaded than those from the high-bandwidth group. Assuming that machines are continuously available, a low-bandwidth machine should replicate at most 2.25GB; and a high-bandwidth machines at most 4.5GB.

We tested two parameter settings that differed by the weight assigned to geographical distribution of replicas (see Section 11.4.1). For $M = 1$ (which means increasing the backup duration of a single chunk by 1s is equally unwanted as increasing the TTL distance of this chunk by 1), the average TTL distance between the replica and the owner was equal to 11.6 (with standard deviation of 3.7). In this case only two machines exceeded their backup window (by at most 108 s). For $M = 0.01$ (which means increasing the backup duration of a single chunk by 100s is as bad as increasing the TTL distance of a single chunk by 1), the replicas were geographically closer with mean TTL 8.1 (with standard deviation of 3.8). However, the backup duration was increased—13 machines exceeded their backup window. The average excess of the backup window was equal to 222s (5% of the backup window) and the maximal 415s (9% of the backup window).

11.5.4 Duration of Backup of a Data Chunk

We measured the time needed to achieve the consecutive redundancy levels (the number of replicas) for each data chunk. The time is measured relative to the data chunk owner online time: we multiplied the absolute time by the owner's availability.

We consider the relative time as a more fair measure because: (i) the transfer to at least the first replica requires the owner to be available; (ii) data can be modified (and, thus, the amount of data for backup grows) only when the owner is available; (iii) we are able to directly compare results from machines having different availabilities.

The distribution of time needed to achieve the consecutive redundancy levels is presented in Figure 11.9 (lab) and Figure 11.10 (PlanetLab).

Students' Computer Lab

The average time of creating the first, the second and the third replica of a data chunk are equal to, respectively, 1.1h, 2.7h and 5.5h (the average time needed to create a single replica is equal to 3.1h). We consider these values to be satisfactory as the average relative time for transferring a single asynchronous message holding no data (message with 0 synchro-peers), calculated based on the availability traces, is equal to 2.6h.

The maximal times needed to create a single replica, though, are higher: 24h, 29h and 32h. These long times of replication are almost entirely the consequence of peers' unavailability. The maximal time needed to deliver an asynchronous message with 3 synchro-peers is of the same order (21.5h, measured relatively to source online time, see Section 11.5.2). Moreover, if we measure only the nodes with more than 20% average availability, the times needed to create the replicas are equal to 1h, 1.6h and 3h and maximal values are equal to 12h, 18h and 20h.

PlanetLab

The average times needed for creating the first, the second and the third replica are equal to, respectively, 0.5h, 0.7h and 1.1h. The maximal values are equal to 4.0h, 4.2h, and 4.2h. These values are significantly better than in case of the students computer lab even though the distance between the machines is much higher and the computers in the students' lab are connected with a fast local network. This once again proves that the unavailability of the machines is the dominating factor influencing the backup duration.

The average time needed for creating a replica of a data chunk is equal to 0.76h (45 minutes). Let us assume that the transfer times of chunks are similar between each other. If the three replicas are transferred sequentially then, to achieve the average backup time of a chunk equal to 0.76h, the transfer of all data (3GB) should take 1.52h. If the three replicas are transferred concurrently then, to achieve the average backup time of a chunk equal to 0.76h, the transfer of all data (3GB) should take 0.76h. In both cases we can assume that 3GB of data needs at most 1.52h for transfer. This gives an estimated throughput of 4.49Mb/s (PlanetLab uses standard Internet connections).

11.6 Conclusions

We present an architecture of a P2P backup system based on pair-wise replication contracts. In contrast to storing the data in a DHT, in our approach the placement can be optimized to a specific network topology, which allows to take into account e.g., geographical dispersion of the nodes.

We have implemented a prototype and tested it on 150 computers in the Faculty of Mathematics, Informatics and Mechanics of the University of Warsaw and on 50 computers in PlanetLab.

During implementation and initial tests we encountered numerous issues we did not expect: e.g., updating data catalog remotely whenever any contract is changed is highly inefficient; revoking the contracts cannot be done asynchronously; changing contracts too often is inefficient; each contract must be kept by both the data owner and the replicator and the two versions have to be kept consistent. We think that these problems should motivate others to verify their ideas, in addition to simulations, by constructing prototype implementations.

Our most important result is that the backup time increases significantly if machines are weakly-available. From 0.76h for nearly-always available PlanetLab nodes to 3.1h for our lab with just 13% average availability. This *cost of unavailability* makes some environments less suitable for P2P backup. The irregular environments negatively influence the maximal durations of data transfer. Choosing machines with better availability strongly reduces this effect (for instance, by restricting our lab environment to machines with more than 20% availability, the average backup time decreases from 3.1h to 1.9h). Moreover, in enterprise environments such irregular availabilities should not be the case. There, however, the machines may have their specific, regular availability patterns. In such case it may be valuable to use more sophisticated availability models.

Yet, as our main conclusion we must stress that it is possible to build an efficient and reliable backup system, even for the environment with weakly-available machines having irregular session times. We built a scientific prototype and we managed to run it on 150 machines—these results are promising and might be considered as the proof of the concept for designing the full efficient and reliable P2P backup system.

Discussion & Conclusions

In this dissertation we presented a comprehensive and diversified view on resource allocation problems in distributed systems. Our perspective was diversified in the several following aspects. First, we considered systems that are distributed in a couple of various ways. For instance, we studied physically distributed systems, in particular the computer systems that are built from a large number of computational units (Chapters 8–10, and Chapter 11), and the geographically distributed computer systems (Chapters 9, and 11). We studied computer systems that serve large numbers of users (Chapters 8, 9, and 11), in particular multi-organizational systems (Chapter 8). We considered general multi-agent systems (Chapters 3–6, and Chapter 7) and a specific, particularly interesting, example of the multi-agent systems—a society participating in various kinds of referendums and elections (Chapter 5 and 6).

Second, we presented a comprehensive comparison of the variety of distributed systems and a number of models describing these systems. These systems and these models differ in their scope and in their level of generality. For instance, in the first part of this dissertation we considered a general model that describes selecting a collective set of items. This general model has broad applications, ranging from allocation of sharable resources, through recommendation systems, to election systems. We described these applications in detail in Chapter 3. In the second part we described more specific models that one needs to consider when studying job scheduling and load balancing problems in distributed computer systems. In the third part of this work we studied designing resource allocation mechanisms in real-life complex distributed computer systems.

Third, we considered different types of problems in distributed systems. We mostly focused on an algorithmic view and, in particular, we considered algorithmic optimization problems. In general, in such problems our goal was to design algorithms that compute efficient resource allocations. On one hand, we classified computational complexity of a number of resource allocation problems. On the other hand, we showed how to deal with computationally hard problems by applying parameterized complexity theory, designing approximation algorithms, and, in case of particularly hard problems, designing heuristic algorithms that work effectively in practice. However, we also studied game theoretic problems, where we aimed at designing stable and fair resource management mechanisms which guarantee that individual users have incentives to operate in such managed systems. Additionally, in case of real-life complex distributed computer systems, we showed how to design other elements of resource allocation mechanisms, such as their architecture, their communication protocols, their monitoring services, etc.

We believe that this multi-perspective and diversified view on resource allocation problems is the core strength of this dissertation. In consecutive chapters we explained the specific elements of the considered models and the key aspects that governed our decisions regarding the formal questions we asked and the methodologies that we used to obtain satisfactory answers. We think that by presenting diversified methodologies, summarizing them, and by sharing our experiences, we will help other engineers and scientists in choosing the approaches that would most suitably fit their resource allocation problems. There is a number of take-home messages that we presented in this dissertation and below we recall some of them. One of our main observations was that, in many cases, the theoretically hard problems can be effectively solved using relatively simple, yet insightful, algorithms. Somewhat surprisingly, we found this observation true in each of the considered models and in each of the considered systems. For example, in Part I we showed that greedy algorithms and their intuitive enhancements can efficiently solve a theoretically hard problem of selecting a collective set of items in many of its most interesting variants. In Part II we showed that a natural scheduling algorithm that collects virtual money for computing someone's jobs and uses this money to pay for processing their own jobs is fair for the users. We showed that gossip-based algorithms can be effectively used to balance the load even in geographically distributed systems.

Our second interesting observation concerned methodologies used to evaluate the quality of resource allocation mechanisms in real-life complex computer systems. Our experiences suggest that the most insightful methodology is to run experimental evaluation of such mechanisms in real systems. Both theoretical analysis of such systems, and analysis through simulations often introduce simplification that significantly affect their credibility. Yet, we admit that in many relatively simpler contexts, specifically when some part of the system can be isolated and studied separately from other elements, theoretical analysis and simulations provide a good tool for evaluation, and allow one to get a deeper understanding of the nature of the considered problems.

Last but not least, due to our multi-perspective approach, we gave an overview of the desired properties of resource management mechanisms in several systems. In the abstract model our main criterium was how close are the results returned by our algorithms to the optimal solutions. Additionally, we argued that it is desirable to look for algorithms that work well with limited information—this is because very often finding complete information for a given model is either impossible or expensive. In the second part, we focused on fairness and stability of the algorithms. We argued that such criteria are important to consider in systems that gather different organizations, or simply serve a number of different, independent users. In Chapters 9 and 11 we analyzed fault-tolerance of our algorithms, and in Chapter 10 we described a scenario where it is important to ensure predictability of an algorithm and its stability when confronted with dynamic changes in the workload.

Apart from giving a comprehensive view on resource allocation in distributed systems, our work brings other high-level contributions. In case of multi-agent and multi-organization systems our algorithms allow one to better organize resource acquisition and resource management among groups of agents and organizations. Thus, our algorithms improve certain processes involving cooperation and communication between agents and organizations. When viewed from this perspective, we believe that our work is a step toward promoting cooperation between people and between organizations, and toward improving their communication.

We also believe that by linking concepts from different domains we popularize certain classes of problems to a broader scientific community. One example of such a link, presented in this dissertation, is that we show how to view election systems as resource allocation, and vice-versa (giving also the relation to some fundamental theoretical problems such as the MaxCover problem). Another example is that we continue the trend that takes the concepts from non-cooperative and cooperative game theory and shows how to apply them in the domain of distributed systems.

We emphasize that the results presented in this dissertation have many direct applications. For instance, we recall that our resource management mechanism from Chapter 10 have already been used in the production version of the commercial storage system HYDRAstor for several years. We already work on integrating our scheduling algorithms from Chapter 8 with CometCloud [65], a framework for running real-world applications on supercomputers and in data-centers. Finally, we believe that our algorithms from Chapter 5 will, eventually, make it possible to use the two appealing election systems in practice.

The technical contributions of our work are numerous. As we already discussed, in the third part of this work we discussed resource allocation mechanisms in real complex distributed systems, and we shared our experiences from considering resource allocation problems in real target environment. Further, in Chapters 3 and 7 we introduced new mathematical models that formalize some fundamental resource allocation problems. In the first two parts of the dissertation we established computational complexity and proposed exact, approximation, and heuristic algorithms that work very well in practice, for a number of variants of several important resource allocation problems. In Chapters 5, 8, and 9 we additionally confirmed our theoretical results by experiments. The contribution of the theoretical part of this dissertation is not limited to analysis of the complexity and of the approximation properties of the optimization algorithms. In Chapters 7, and 8 we presented how game theoretic solution concepts can be applied to resource allocation mechanism, and we showed how to design resource allocation mechanisms that are incentive-compatible, and, in effect, stable in the context of multi-agent and multi-organizational systems.

Finally, for each of the considered questions, we suggested interesting future directions of research and we shared the ideas on how to extend the presented results.

In this dissertation we also addressed a more general problem. We tried to alleviate the negative impact of impossibility results given by the complexity theory. Even though many interesting computational problems that we face when trying to optimize various business processes are NP-hard, or even inapproximable, we argue that not everything is lost. Indeed, the worst-case complexity of a problem does not preclude that such a problem can be effectively handled in practice. The hardest instances of many problems are often rare and, since in many business processes we just want to be able to find sufficiently good results in sufficiently many cases, sometimes these rare hard instances can simply be ignored. This problem has already been addressed; for instance Leyton-Brown el al. [175] considered statistical hardness of problems. In some way we continue this line of research and, by showing that certain problems are, on one hand, theoretically hard, and, on the other, we can solve most real-life instances of them effectively, we exposed the tension between different methodologies of solving and evaluating solutions of computational problems. We believe that there is no clear answer as to which methodology is superior. However, when addressing the problems we should not be discouraged by the hardness results and we should follow the curiosity that fits so well in the human nature. Taking another look at the hard problem and finding a new approach helps understanding the true complexity of the problem.[4]

[4]Taking this line of research we already extended some of the presented results. For instance, for the problem of finding winners in parliamentary elections, considered in Chapter 5, we analyzed a domain restriction describing the class of “reasonable” votes’ distributions [283]. For some of such natural restrictions we showed that the original problem becomes solvable in polynomial time.

Bibliography

[1] H. Ackermann, S. Fischer, M. Hoefer, and M. Schöngens. Distributed algorithms for qos load balancing. In *Proceedings of the 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA-2009)*, pages 197–203, 2009.

[2] C. P. J. Adolphs and P. Berenbrink. Distributed selfish load balancing with weights and speeds. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC-2012)*, pages 135–144, 2012.

[3] C. P. J. Adolphs and P. Berenbrink. Improved bounds for discrete diffusive load balancing. In *Proceedings of the 26th International Parallel and Distributed Processing Symposium (IPDPS-2012)*, pages 820–826, 2012.

[4] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

[5] A. Ageev and M. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In G. Cornuéjols, R. Burkard, and G. Woeginger, editors, *Integer Programming and Combinatorial Optimization*, volume 1610 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 1999.

[6] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):Article 23, 2008.

[7] S. Airiau and U. Endriss. Multiagent resource allocation with sharable items: Simple protocols and nash equilibria. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, pages 167–174, May 2010.

[8] P. Alimonti and V. Kann. Some apx-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1–2):123–134, 2000.

[9] N. Alon, S. Arora, R. Manokaran, D. Moshkovitz, and O. Weinstein. Inapproximabilty of densest k-subgraph from average case hardness. Technical report, 2011.

[10] B. Amann, B. Elser, Y. Houri, and T. Fuhrmann. Igorfs: A distributed p2p file system. In *Proceedings of the 8th International Conference on Peer-to-Peer Computing (P2P-2008)*, pages 77–78, 2008.

[11] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID-2004)*, pages 4–10, 2004.

[12] N. Andersson. *The Single And Multi Project Approach To Planning And Scheduling*. CBS, 2008.

[13] S. S. Aote and M. U. Kharat. A game-theoretic model for dynamic load balancing in distributed systems. In *Proceedings of the International Conference on Advances in Computing, Communication and Control (ICAC3-2009)*, pages 235–238, 2009.

[14] A. Archer and E. Tardos. Frugal path mechanisms. *ACM Transactions on Algorithms*, 3(1):3:1–3:22, Feb. 2007.

[15] R. J. Aumann. Acceptable points in general cooperative N-person games. In *Contribution to the theory of game IV, Annals of Mathematical Study 40*, pages 287–324. University Press, 1959.

[16] D. Baccarini. The concept of project complexity–a review. *International Journal of Project Management*, 14(4):201–204, 1996.

[17] Y. Bachrach, D. C. Parkes, and J. S. Rosenschein. Computing cooperative solution concepts in coalitional skill games. *Artificial Intelligence*, 204(0):1–21, 2013.

[18] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[19] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Transactions on Parallel and Distributed Systems*, 16(3):207–218, 2005.

[20] P. Berenbrink, M. Hoefer, and T. Sauerwald. Distributed selfish load balancing on networks. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA-2011)*, pages 1487–1497, 2011.

[21] S. Berg. Paradox of voting under an urn model: The effect of homogeneity. *Public Choice*, 47:377–387, 1985.

[22] S. Bernard and F. Le Fessant. Optimizing peer-to-peer backup using lifetime estimations. In *Proceedings of EDBT/ICDT Workshops-2009*, pages 26–33, 2009.

[23] B. D. Bernheim, B. Peleg, and M. D. Whinston. Coalition-proof nash equilibria i. concepts. *Journal of Economic Theory*, 42(1):1–12, 1987.

[24] D. P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1:7–66, 1992.

[25] N. Betzler, M. Fellows, J. Guo, R. Niedermeier, and F. Rosamond. Fixed-parameter algorithms for Kemeny scores. *Theoretical Computer Science*, 410(45):4554—-4570, 2009.

[26] N. Betzler, J. Guo, and R. Niedermeier. Parameterized computational complexity of Dodgson and Young elections. *Information and Computation*, 208(2):165–177, 2010.

[27] N. Betzler, A. Slinko, and J. Uhlmann. On the computation of fully proportional representation. *Journal of Artificial Intelligence Research*, 47:475–519, 2013.

[28] R. Bhagwan, S. Savage, and G. Voelker. Replication strategies for highly available peer-to-peer storage systems. In *Proceedings of Future Directions in Distributed Computing-2003*, pages 153–158, 2002.

[29] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $o(n^{1/4})$ approximation for densest k-subgraph. In *Proceedings of the 42nd Symposium on Theory of Computing (STOC-2010)*, pages 201–210, 2010.

[30] A. Bhaskara, M. Charikar, A. Vijayaraghavan, V. Guruswami, and Y. Zhou. Polynomial integrality gaps for strong sdp relaxations of densest k-subgraph. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA-2012)*, pages 388–405, 2012.

[31] S. Bhat, S. Nath, O. Zoeter, S. Gujar, Y. Narahari, and C. Dance. A mechanism to optimally balance cost and quality of labeling tasks outsourced to strategic agents. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2014)*, pages 917–924, 2014.

[32] M. Bläser. Computing small partial coverings. *Information Processing Letters*, 85(6):327–331, 2003.

[33] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki. Improving duplicate elimination in storage systems. *ACM Transactions on Storage*, 2(4):424–448, 2006.

[34] W. J. Bolosky, J. R. Douceur, and J. Howell. The Farsite project: a retrospective. *ACM SIGOPS Operating Systems Review*, 41:17–26, April 2007.

[35] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz. Disk scheduling with quality of service guarantees. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS-1999)*, page 400, 1999.

[36] J. Brutlag. Speed matters. `http://googleresearch.blogspot.com/2009/06/speed-matters.html`, 2009.

[37] J. Busca, F. Picconi, and P. Sens. Pastis: A highly-scalable multi-user peer-to-peer file system. In *Proceedings of the 11th European Conference on Parallel Computing (Euro-Par-2005)*, pages 1173–1182, 2005.

[38] R. Buyya, D. Abramson, and S. Venugopal. The grid economy. In *Special Issue on Grid Computing*, volume 93, pages 698–714, 2005.

[39] L. Cai. Parameterized complexity of cardinality constrained optimization problems. *The Computer Journal*, 51(1):102–121, 2008.

[40] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (GRID-2005)*, pages 99–106, 2005.

[41] I. Caragiannis, J. Covey, M. Feldman, C. Homan, C. Kaklamanis, N. Karanikolas, A. Procaccia, and J. Rosenschein. On the approximability of Dodgson and Young elections. *Artificial Intelligence*, 187:31–51, 2012.

[42] I. Caragiannis, C. Kaklamanis, N. Karanikolas, and A. Procaccia. Socially desirable approximations for Dodgson's voting rule. In *Proceedings of the 11th ACM Conference on Electronic Commerce (ACM-EC-2010)*, pages 253–262, June 2010.

[43] S. K. Card, T. P. Moran, and A. Newell. *The psychology of human computer interaction.* Routledge, 1983.

[44] V. Cardellini. Geographic load balancing for scalable distributed web systems. In *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS-2000)*, pages 20–27, 2000.

[45] T. E. Carroll and D. Grosu. Divisible load scheduling: An approach using coalitional games. In *Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC-2007)*, pages 36–36, 2007.

[46] M. Cesati. The Turing way to parameterized complexity. *J. Comput. Syst. Sci.*, 67(4):654–685, 2003.

[47] B. Chamberlin and P. Courant. Representative deliberations and representative decisions: Proportional representation and the borda rule. *American Political Science Review*, 77(3):718–733, 1983.

[48] E. Chan-Tin and N. Hopper. Accurate and provably secure latency estimation with treeple. In *Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS-2011)*, 2011.

[49] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26:4:1–4:26, June 2008.

[50] P. Charrel and D. Galarreta. *Project Management and Risk Management in Complex Projects*. Springer, Nov. 2010.

[51] H. M. Chaskar and U. Madhow. Fair scheduling with tunable latency: a round-robin approach. *IEEE/ACM Transactions on Networking*, 11(4):592–601, 2003.

[52] N. Chen, E. Elkind, N. Gravin, and F. Petrov. Frugal mechanism design via spectral techniques. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS-2010)*, pages 755–764, 2010.

[53] N. Chen and A. R. Karlin. Cheap labor can be expensive. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA-2007)*, pages 707–715, 2007.

[54] Y. Chevaleyre, P. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J. Rodríguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica*, 30:3–31, 2006.

[55] T. C. K. Chou and J. A. Abraham. Load balancing in distributed systems. *IEEE Transactions on Software Engineering*, 8(4):401–412, 1982.

[56] G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In *Proceedings of the 37th Symposium on Theory of Computing (STOC-2005)*, pages 67–73, 2005.

[57] F. Chudak and D. P. Williamson. Improved approximation algorithms for capacitated facility location problems. *Mathematical Programming*, 102(2):207–222, Mar. 2005.

[58] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[59] B. G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI-2006)*, volume 6, pages 4–4, 2006.

[60] V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[61] M. S. Chwe. Farsighted coalitional stability. *Journal of Economic Theory*, 63:299–325, 1994.

[62] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.

[63] www.cleversafe.com/.

[64] M. Colajanni, P. S. Yu, and V. Cardellini. Dynamic load balancing in geographically distributed heterogeneous web servers. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS-1998)*, pages 295–302, 1998.

[65] `http://nsfcac.rutgers.edu/CometCloud/`.

[66] V. Conitzer, M. Rognlie, and L. Xia. Preference functions that score rankings and maximum likelihood estimation. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-2009)*, pages 109–115, July 2009.

[67] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Symposium on Theory of Computing (STOC-1971)*, pages 151–158, 1971.

[68] D. Coppersmith, L. Fleisher, and A. Rurda. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *ACM Transactions on Algorithms*, 6(3):Article 55, 2010.

[69] L. P. Cox and B. D. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (ACM-SOSP-2003)*, pages 120–132, 2003.

[70] F. Croce and V. Paschos. Efficient algorithms for the max k-vertex cover problem. *Journal of Combinatorial Optimization*, To appear (available as an online-first article).

[71] E. Cronin, S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the internet. *IEEE Journal on Selected Areas in Communications*, pages 31–40, 2002.

[72] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2):279–301, Oct. 1989.

[73] M. Cygan, Ł. Kowalik, and M. Wykurz. Exponential-time approximation of weighted set cover. *Information Processing Letters*, 109(16):957–961, 2009.

[74] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. *ACM SIGOPS Operating Systems Review*, 35(5):202–215, Oct. 2001.

[75] A. Darmann, E. Elkind, S. Kurz, J. Lang, J. Schauer, and G. Woeginger. Group activity selection problem. In *Proceedings of the 8th Workshop on Internet & Network Economics (WINE-2012)*, pages 156–169, Dec. 2012.

[76] http://www.cs.vu.nl/das3/.

[77] M. D. Davis. *The Kernel of a Cooperative Game*. Princeton University, 1963.

[78] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008.

[79] B. Debord. An axiomatic characterization of Borda's k-choice function. *Social Choice and Welfare*, 9(4):337–343, 1992.

[80] S. Defrance, A. Kermarrec, E. L. Merrer, N. L. Scouarnec, G. Straub, and A. van Kempen. Efficient peer-to-peer backup services through buffering at the edge. In *Proceedings of the 11th International Conference on Peer-to-Peer Computing (P2P-2011)*, pages 142–151, 2011.

[81] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. Yang, and D. A. Bader. Dynamic load balancing in distributed systems in the presence of delays: A regeneration-theory approach. *IEEE Transactions Parallel Distributed Systems*, 18(4):485–497, Apr. 2007.

[82] E. Diamantoudi and L. Xue. Farsighted stability in hedonic games. *Social Choice and Welfare*, 21(1):39–61, 2003.

[83] Y. Diao, C. M. Garcia-Arellano, J. L. Hellerstein, S. S. Lightstone, S. S. Parekh, A. J. Storm, and M. Surendra. Systems and methods for providing constrained optimization using adaptive regulatory control, December 2005. US patent application no 20050268063.

[84] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the 9th Conference on File and Storage Technologies (FAST-2011)*, pages 2–2, 2011.

[85] J. R. Douceur and J. Howell. Byzantine fault isolation in the Farsite distributed file system. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS-2006)*, 2006.

[86] J. R. Douceur and R. P. Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC-2001)*, volume 2180, pages 48–62, 2001.

[87] R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[88] M. Drozdowski and M. Lawenda. Scheduling multiple divisible loads in homogeneous star systems. *Journal of Scheduling*, 11(5):347–356, 2008.

[89] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. HYDRAstor: a Scalable Secondary Storage. In *Proceedings of the 7th Conference on File and Storage Technologies (FAST-2009)*, pages 197–210, 2009.

[90] P. Dutot, F. Pascual, K. Rzadca, and D. Trystram. Approximation algorithms for the multi-organization scheduling problem. *IEEE Transactions on Parallel and Distributed Systems*, 22:1888–1895, 2011.

[91] P. F. Dutot, L. Eyraud, G. Mounié, and D. Trystram. Bi-criteria algorithm for scheduling jobs on cluster platforms. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS-2004)*, pages 125–132, 2004.

[92] D. J. Edwards. Accident trends involving construction plant: An exploratory analysis. *Journal of Construction Research*, 04(02):161–173, 2003.

[93] http://egee2.web.cern.ch/egee2/.

[94] L. Eggert and J. D. Touch. Idletime scheduling with preemption intervals. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (ACM-SOSP-2005)*, pages 249–262, 2005.

[95] M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, 7:2000, 2000.

[96] E. Elkind, P. Faliszewski, P. Skowron, and A. Slinko. Properties of multiwinner voting rules. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2014)*, May 2014. Also presented in FIT-2013.

[97] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Multimode control attacks on elections. *Journal of Artificial Intelligence Research*, 40:305–351, 2011.

[98] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[99] U. Feige and S. Kogan. Hardness of approximation of the balanced complete bipartite subgraph problem. Technical report, 2004.

[100] D. G. Feitelson. Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/.

[101] M. R. Fellows and H. Fernau. Facility location problems: A parameterized view. *Discrete Applied Mathematics*, 159(11):1118–1130, July 2011.

[102] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[103] M. J. Freedman. Experiences with coralcdn: A five-year operational view. In *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI-2010)*, 2010.

[104] R. M. Freund and J. R. Vera. Equivalence of convex problem geometry and computational complexity in the separation oracle model. *Mathematics of Operations Research*, 34(4):869–879, 2009.

[105] M. Gallet, Y. Robert, and F. Vivien. Divisible load scheduling. In Y. Robert and F. Vivien, editors, *Introduction to Scheduling*. CRC Press, Inc., 2009.

[106] A. Galluccio and P. Nobili. Improved approximation of maximum vertex cover. *Operations Research Letters*, 34(1):77–84, 2006.

[107] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[108] R. Garg, V. Kumar, A. Rudra, and A. Verma. Coalitional games on graphs: Core structure, substitutes and frugality. In *Proceedings of the 4th ACM Conference on Electronic Commerce (ACM-EC-2003)*, pages 248–249, 2003.

[109] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43, 2003.

[110] P. Ghosh, K. Basu, and S. K. Das. A game theory-based pricing strategy to support single/multiclass job allocation schemes for bandwidth-constrained distributed computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(3):289–306, 2007.

[111] P. Ghosh, N. Roy, S. K. Das, and K. Basu. A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework. *Journal of Parallel and Distributed Computing*, 65(11):1366–1383, 2005.

[112] D. B. Gillies. Solutions to general non-zero-sum games. In A. W. Tucker and R. D. Luce, editors, *Contributions to the Theory of Games*, Annals of mathematics studies. Princeton University Press, 1959.

[113] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15:430–466, July 1990.

[114] D. Goldfarb and S. Liu. An o(n3) primal interior point algorithm for convex quadratic programming. *Mathematical Programming*, 49:325–340, January 1991.

[115] P. Goyal, H. M. Vin, and H. Chen. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *Proceedings of the ACM SIGCOMM 1996 Conference on Communications Architecture & Protocols (SIGCOMM-1996)*, pages 157–168, 1996.

[116] S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller. Self-similarity in file systems. *ACM SIGMETRICS Performance Evaluation Review*, 26(1):141–150, 1998.

[117] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of Queueing Theory*. Wiley-Interscience, New York, NY, USA, 4th edition, 2008.

[118] D. Grosu and A. T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. In *Proceedings of the 2002 IEEE International Conference on Cluster Computing (CLUSTER-2002)*, pages 445–445, 2002.

[119] D. Grosu and A. T. Chronopoulos. A game-theoretic model and algorithm for load balancing in distributed systems. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS-2002)*, pages 146–153, 2002.

[120] D. Grosu and A. T. Chronopoulos. A truthful mechanism for fair load balancing in distributed systems. In *Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications (NCA-2003)*, pages 289–289, 2003.

[121] D. Grosu and A. T. Chronopoulos. Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, 65(9):1022–1034, 2005.

[122] D. Grosu, A. T. Chronopoulos, and M. Y. Leung. Cooperative load balancing in distributed systems. *Concurrency and Computation: Practice and Experience*, 20(16):1953–1976, Nov. 2008.

[123] D. Grosu and A. Das. Auction-based resource allocation protocols in grids. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS-2004)*, pages 20–27, 2004.

[124] A. Gulati and I. Ahmad. Towards distributed storage resource management using flow control. *ACM SIGOPS Operating Systems Review*, 42(6):10–16, 2008.

[125] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access. In *Proceedings of the 7th Conference on File and Storage Technologies (FAST-2009)*, February 2009.

[126] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of vertex cover variants. *Theoretical Computer Science*, 41(3):501–520, 2007.

[127] A. Hać. Load balancing in distributed systems: A summary. *ACM SIGMETRICS Performance Evaluation Review*, 16(2-4):17–19, Feb. 1989.

[128] Q. Han, Y. Ye, H. Zhang, and J. Zhang. On approximation of max-vertex-cover. *European Journal of Operational Research*, 143(2):342–355, 2002.

[129] S. Hart. Shapley value. In J. Eatwell, M. Milgate, and P. Newman, editors, *The New Palgrave: A Dictionary of Economics*. Palgrave Macmillan, 1987.

[130] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *Proceedings of the 2005 International Conference on Information Technology Coding and Computing (ITCC-2005)*, pages 205–213, 2005.

[131] M. M. Hayat, S. Dhakal, C. T. Abdallah, J. Douglas, and J. Chiasson. Dynamic time delay models for load balancing, part ii: A stochastic analysis of the effect of delay uncertainty. In *CNRS-NSF Workshop: Advances in Control of Time-Delay Systems*, 2003.

[132] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005.

[133] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. 2004.

[134] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *J. ACM*, 44(6):806–825, 1997.

[135] D. Hochbaum. Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 94–143. PWS Publishing, 1996.

[136] D. S. Hochbaum. Approximation algorithms for np-hard problems. chapter Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems, pages 94–143. PWS Publishing Co., Boston, MA, USA, 1997.

[137] D. S. Hochbaum. Complexity and algorithms for nonlinear optimization problems. *Annals of Operations Research*, 153(1):257–296, 2007.

[138] C. Homan and L. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. *Journal of Heuristics*, 15(4):403–423, 2009.

[139] L. Huang, G. Peng, and T. Chiueh. Multi-dimensional storage virtualization. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):14–24, 2004.

[140] Independent contractors: How many? (Australia). www.independentcontractors.net.au/Research/How-Many/independent-contractors-how-many.

[141] A. Inoie, H. Kameda, and C. Touati. Pareto set, fairness, and nash equilibrium: A case study on load balancing. In *Proceedings of the 11th International Symposium on Dynamic Games and Applications (ISDG-2004)*, pages 386–393, 2004.

[142] A. Iwasaki, D. Kempe, Y. Saito, M. Salek, and M. Yokoo. False-name-proof mechanisms for hiring a team. In *Proceedings of the 3rd Workshop on Internet & Network Economics (WINE-2007)*, pages 245–256, 2007.

[143] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC-2002)*, pages 213–222, 2002.

[144] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the maui scheduler. In *Proceedings of the 7th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP-2001)*, pages 87–102, 2001.

[145] K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, Mar. 2001.

[146] A. Jameson and B. Smyth. Recommendation to groups. In *The Adaptive Web*, pages 596–627, 2007.

[147] K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. In *Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT-2010)*, pages 260–272, 2010.

[148] M. Jelasity and Ö. Babaoğlu. T-man: Gossip-based overlay topology management. In *Engineering Self-Organising Systems*, pages 1–15. Springer, 2006.

[149] X. Jia, D. Li, X. Hu, and D. Du. Optimal placement of web proxies for replicated web servers in the internet. *Computer Journal*, 44(5):329–339, 2001.

[150] W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):37–48, 2004.

[151] J. Kacprzyk, H. Nurmi, and S. Zadrozny. The role of the OWA operators as a unification tool for the representation of collective choice sets. In *Recent Developments in the Ordered Weighted Averaging Operators*, pages 149–166. 2011.

[152] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th IEEE International Conference on Autonomic Computing (ICAC-2009)*, pages 117–126, 2009.

[153] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal Load Balancing in Distributed Computer Systems*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[154] T. Kamishima. Nantonac collaborative filtering: Recommendation based on order responses. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, pages 583–588, 2003.

[155] A. R. Karlin, D. Kempe, and T. Tamir. Beyond vcg: Frugality of truthful mechanisms. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS-2005)*, pages 615–626, 2005.

[156] M. Karlsson and C. Karamanolis. Non-intrusive performance management for computer services. In *Proceedings of the 7th ACM/IFIP/USENIX International Middleware Conference (Middleware-2006)*, pages 22–41, 2006.

[157] M. Karlsson, C. Karamanolis, and J. Chase. Controllable fair queuing for meeting performance goals. *Performance Evaluation*, 62(1–4):278–294, 2005.

[158] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance differentiation for storage systems using adaptive control. *ACM Transactions on Storage*, 1(4):457–480, 2005.

[159] J. Kay and P. Lauder. A fair share scheduler. *Communications of the ACM*, 31(1):44–55, 1988.

[160] C. Kenyon and G. Cheliotis. Grid resource commercialization: economic engineering and delivery scenarios. In J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors, *Grid resource management: state of the art and future trends*, pages 465–478. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[161] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. In *Proceedings of the 39th Symposium on Theory of Computing (STOC-2007)*, pages 95–103, June 2007.

[162] S. Keshav. A control-theoretic approach to flow control. *ACM SIGCOMM Computer Communication Review*, 21(4):3–15, 1991.

[163] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 244, 1979.

[164] S. Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36:1025–1071, 2006.

[165] C. Kintala and P. Fisher. Refining nondeterminism in relativized polynomial-time bounded computations. *SIAM Journal on Computing*, 9(1):46–53, 1980.

[166] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10:3–25, 1992.

[167] M. M. Kostreva, W. Ogryczak, and A. Wierzbicki. Equitable aggregations and multiple criteria analysis. *European Journal of Operational Research*, 158(2):362–377, 2004.

[168] S. Kraus, O. Shehory, and G. Taase. Coalition formation with uncertain heterogeneous information. In *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, pages 1–8, 2003.

[169] S. Kraus, O. Shehory, and G. Taase. The advantages of compromising in coalition formation with incomplete information. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2004)*, pages 588–595, 2004.

[170] V. Krishna. *Auction Theory*. Academic Press, Mar. 2002.

[171] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 35:190–201, November 2000.

[172] C. B. Lee and A. Snavely. On the user–scheduler dialogue: studies of user-provided runtime estimates and utility functions. *International Journal of High Performance Computing Applications*, 20(4):495–506, 2006.

[173] F. Leighton and D. Lewin. Global hosting system. US Patent No. 6,108,703.

[174] L. A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[175] K. Leyton-Brown, H. H. Hoos, F. Hutter, and L. Xu. Understanding the empirical hardness of np-complete problems. *Communications of the ACM*, 57(5):98–107, May 2014.

[176] Y. Li and Z. Lan. A survey of load balancing in grid computing. In *Proceedings of CIS-2004*, pages 280–285, 2004.

[177] D. Liben-Nowell, A. Sharp, T. Wexler, and K. Woods. Computing shapley value in supermodular coalitional games. In *Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON-2012)*, volume 7434, pages 568–579, 2012.

[178] J. T. Lim and S. M. Meerkov. Distributed load balancing. In *Proceedings of the 27th IEEE Conference on Decision and Control (CDC-1988)*, volume 2, page 1487, 1988.

[179] M. Lin, Z. Liu, A. Wierman, and A. L. H. Lachlan. Online algorithms for geographical load balancing. In *Proceedings of the 5th International Green Computing Conference (IGCC-2012)*, pages 1–10, 2012.

[180] P. Linga, I. Gupta, and K. Birman. A churn-resistant peer-to-peer web caching system. In *Proceedings of the 2003 ACM workshop on Survivable and Self-Regenerative Systems (SSRS-2003)*, pages 1–10, 2003.

[181] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. H. Andrew. Greening geographical load balancing. In *Proceedings of the 2011 ACM SIGMETRICS International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS-2011)*, pages 233–244, 2011.

[182] Z. Liu, M. Lin, A. Wierman, S. H. Low, and A. L. H. Lachlan. Geographical load balancing with renewables. *ACM SIGMETRICS Performance Evaluation Review*, 39(3):62–66, Dec. 2011.

[183] Z. Liu, A. Wierman, Y. Chen, B. Razon, and N. Chen. Data center demand response: Avoiding the coincident peak via workload shifting and local generation. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):341–342, June 2013.

[184] B. Lockwood. Pareto efficiency. In J. Eatwell, M. Milgate, and P. Newman, editors, *The New Palgrave: A Dictionary of Economics*. Palgrave Macmillan, Basingstoke, 1987.

[185] S. Lohr. For impatient web users, an eye blink is just too long to wait. *New York Times*, 2012.

[186] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *Proceedings of the 1st Conference on File and Storage Technologies (FAST-2002)*, page 21, 2002.

[187] T. Lu and C. Boutilier. The unavailable candidate model: a decision-theoretic view of social choice. In *Proceedings of the 11th ACM Conference on Electronic Commerce (ACM-EC-2010)*, pages 263–274, 2010.

[188] T. Lu and C. Boutilier. Budgeted social choice: From consensus to personalized decision making. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011)*, pages 280–286, 2011.

[189] T. Lu and C. Boutilier. Learning Mallows models with pairwise preferences. In *Proceedings of the 28th International Conference on Machine Learning (ICML-2011)*, pages 145–152, June 2011.

[190] R. Luce and H. Raiffa. *Games and Decisions: Introduction and Critical Survey*. Dover Publications, 1989.

[191] C. R. Lumb, A. Merchant, and G. A. Alvarez. Facade: Virtual storage devices with performance guarantees. In *Proceedings of the 2nd Conference on File and Storage Technologies (FAST-2003)*, pages 131–144, 2003.

[192] C. H. M. Hefeeda and K. Mokhtarian. pcache: A proxy cache for peer-to-peer traffic. In *Proceedings of the ACM SIGCOMM 2008 Conference on Communications Architecture & Protocols (SIGCOMM-2008)*, 2008.

[193] T. Mager, E. Biersack, and P. Michiardi. A measurement study of the Wuala on-line storage service. In *P2P*, Sept. 2012.

[194] R. Mahajan. How akamai works? `http://research.microsoft.com/en-us/um/people/ratul/akamai.html`.

[195] C. L. Mallows. Non-null ranking models. I. *Biometrika*, 44(1-2):114–130, June 1957.

[196] D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

[197] L. Mashayekhy and D. Grosu. A merge-and-split mechanism for dynamic virtual organization formation in grids. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):540–549, Mar. 2014.

[198] J. Masthoff. Group recommender systems: Combining individual models. In F. Ricci, L. Rokach, B. Shapira, and P. Kantor, editors, *Recommender Systems Handbook*, pages 677–702. Springer, 2010.

[199] N. Mattei, J. Forshee, and J. Goldsmith. An empirical study of voting rules and manipulation with large datasets. In *Proceedings of the 4th International Workshop on Computational Social Choice (COMSOC-2012)*, 2012.

[200] N. Mattei and T. Walsh. Preflib: A library for preferences http://www.preflib.org. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT-2013)*, pages 259–270, 2013.

[201] N. Mattei and T. Walsh. PrefLib: A library of preference data. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT-2013)*, pages 259–270, 2013.

[202] A. McCabe. *Frugality in Set-System Auctions*. PhD thesis, University of Liverpool, August 2012.

[203] J. McCabe-Dansted, G. Pritchard, and A. Slinko. Approximability of Dodgson's rule. *Social Choice and Welfare*, 31(2):311–330, 2008.

[204] J. McCabe-Dansted and A. Slinko. Exploratory analysis of similarities between common social choice rules. *Decision and Negotiation*, 15(1):77–107, 2006.

[205] D. Meister and A. Brinkmann. Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of the Israeli Experimental Systems Conference 2009 (SYSTOR-2009)*, pages 1–12, 2009.

[206] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *Proceedings of the 9th Conference on File and Storage Technologies (FAST-2011)*, pages 1–1, 2011.

[207] N. Mi, A. Riska, X. Li, E. Smirni, and E. Riedel. Restrained utilization of idleness for transparent scheduling of background tasks. In *Proceedings of the 2009 ACM SIGMETRICS International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS-2009)*, pages 205–216, 2009.

[208] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel. Efficient management of idleness in storage systems. In *ACM Transactions on Storage*, volume 5, pages 1–25, June 2009.

[209] D. Mishra and B. Rangarajan. Cost sharing in a job scheduling problem using the shapley value. In *Proceedings of the 6th ACM Conference on Electronic Commerce (ACM-EC-2005)*, pages 232–239, 2005.

[210] B. Monroe. Fully proportional representation. *American Political Science Review*, 89(4):925–940, 1995.

[211] A. Montresor, H. Meling, and Ö. Babaoğlu. Messor: Load-balancing through a swarm of autonomous agents. In *Agents and Peer-to-Peer Computing*, pages 125–137. Springer, 2003.

[212] O. Morgenstern and J. V. Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, May 1944.

[213] J. Moules. Number of freelancers on the increase. *Financial Times*, 2011.

[214] H. Moulin. On scheduling fees to prevent merging, splitting, and transferring of jobs. *Mathematics of Operations Research*, 32(2):266–283, May 2007.

[215] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31–44, 2002.

[216] D. Nanongkai. Simple {FPTAS} for the subset-sums ratio problem. *Information Processing Letters*, 113(19–21):750–753, 2013.

[217] J. F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 36, pages 48–49, 1950.

[218] NEC Corporation. HYDRAstor Grid Storage System, 2008. http://www.hydrastor.com.

[219] www.nec.com/.

[220] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, December 1978.

[221] G. F. Newell. *Applications of queueing theory.* Monographs on applied probability and statistics. Chapman and Hall, 1971.

[222] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[223] N. Nisan and A. Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the 31st Symposium on Theory of Computing (STOC-1999)*, pages 129–140, 1999.

[224] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*, chapter Selfish Load Balancing. Cambridge University Press, 2007.

[225] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*, chapter Routing Games. Cambridge University Press, 2007.

[226] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44:2–19, August 2010.

[227] M. O'Connor, D. Cosley, J. Konstan, and J. Riedl. Polylens: A recommender system for groups of user. In *Proceedings of the 12th European Conference on Computer-Supported Cooperative Work (ECSCW-2010)*, pages 199–218, 2001.

[228] F. E. Oggier and A. Datta. Self-repairing homomorphic codes for distributed storage systems. In *Proceedings of the 30th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-2011)*, pages 1215–1223, 2011.

[229] J. Oren. Personal communication, 2012.

[230] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Operations Research*, pages 377–387, 1988.

[231] J. M. Osborne and A. Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, 1994.

[232] P. Padala, K. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th European Conference on Computer Systems (EuroSys-2009)*, pages 13–26, 2009.

[233] G. Pallis and A. Vakali. Content delivery networks. *Communications of the ACM*, 49(1):101, 2006.

[234] L. Pàmies-Juárez, P. García-López, and M. Sánchez-Artigas. Enforcing fairness in P2P storage systems using asymmetric reciprocal exchanges. In *Proceedings of the 11th International Conference on Peer-to-Peer Computing (P2P-2011)*, pages 122–131, 2011.

[235] L. Pamies-Juarez, P. G. Lopez, and M. S. Artigas. Availability and redundancy in harmony: Measuring retrieval times in p2p storage systems. In *Proceedings of the 11th International Conference on Peer-to-Peer Computing (P2P-2011)*, pages 1–10, 2010.

[236] L. Pamies-Juarez, F. E. Oggier, and A. Datta. Decentralized erasure coding for efficient data archival in distributed storage systems. In *Proceedings of the 14th International Conference on Distributed Computing and Networking (ICDCN-2013)*, pages 42–56, 2013.

[237] O. Papapetrou, S. Ramesh, S. Siersdorfer, and W. Nejdl. Optimizing near duplicate detection for p2p networks. In *Proceedings of the 10th International Conference on Peer-to-Peer Computing (P2P-2010)*, pages 1–10, 2010.

[238] S. Park and M. Humphrey. Feedback-controlled resource sharing for predictable escience. In *Proceedings of the ACM/IEEE conference on Supercomputing (SC-2008)*, pages 1–11, 2008.

[239] R. Peeters. The maximum edge biclique problem is np-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.

[240] S. Penmasta and A. T. Chronopoulos. Price-based user-optimal job allocation scheme for grid systems. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS-2006)*, pages 336–336, 2006.

[241] S. Penmatsa and A. T. Chronopoulos. Cooperative load balancing for a network of heterogeneous computers. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS-2006)*, pages 162–162, 2006.

[242] http://www.plgrid.pl/en.

[243] A. Popescu and S. Ghanbari. A study on performance isolation approaches for consolidated storage. Technical report, May 2008.

[244] R. F. Potthoff and S. J. Brams. Proportional representation: Broadening the options. *Journal of Theoretical Politics*, 10(2):147–178, 1998.

[245] A. Procaccia, J. Rosenschein, and A. Zohar. On the complexity of achieving proportional representation. *Social Choice and Welfare*, 30(3):353–362, April 2008.

[246] A. D. Procaccia. Cake cutting: not just child's play. *Communications of the ACM*, 56(7):78–87, 2013.

[247] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-2001)*, pages 1587–1596, 2001.

[248] J. Quiggin. *Generalized Expected Utility Theory. The Rank-Dependent Model.* Kluwer Academic Publishers, 1993.

[249] P. Raghavendra and D. Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the 42nd Symposium on Theory of Computing (STOC-2010)*, pages 755–764, 2010.

[250] R. Rahman, T. Vinkó, D. Hales, J. Pouwelse, and H. Sips. Design space analysis for modeling incentives in distributed systems. In *Proceedings of the ACM SIGCOMM 2011 Conference on Communications Architecture & Protocols (SIGCOMM-2011)*, pages 182–193, 2011.

[251] T. Rahwan, T. P. Michalak, E. Elkind, P. Faliszewski, J. Sroka, M. Wooldridge, and N. R. Jennings. Constrained coalition formation. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI-2011)*, pages 719–725, 2011.

[252] T. Rahwan, T. P. Michalak, and N. R. Jennings. A hybrid algorithm for coalition structure generation. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI-2012)*, pages 1443–1449, 2012.

[253] T. Rahwan, T. P. Michalak, M. Wooldridge, and N. R. Jennings. Anytime coalition structure generation in multi-agent systems with positive or negative externalities. *Artificial Intelligence*, 186:95–122, 2012.

[254] A. Rapoport and A. M. Chammah. *Prisoner's Dilemma: A Study in Conflict and Cooperation.* University of Michigan Press, Ann Arbor, 1965.

[255] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21:2003, 2001.

[256] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS-2005)*, pages 226–239, 2005.

[257] J. Rosenmüller and P. Sudhölter. The nucleolus of homogeneous games with steps. *Discrete Applied Mathematics*, 50(1):53–76, 1994.

[258] A. Roth. The price of malice in linear congestion games. In *Proceedings of the 4th Workshop on Internet & Network Economics (WINE-2008)*, pages 118–125, 2008.

[259] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner problem for Young elections. *ACM Trans. Comput. Syst.*, 36(4):375–386, 2003.

[260] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, 2001.

[261] D. Russell. Emc acquires data domain, becomes deduplication leader, signals deduplication as a "must have" capability. Gartner Research, ID Number: G00170233, 2009.

[262] K. Rzadca, A. Datta, and S. Buchegger. Replica placement in p2p storage: Complexity and game theoretic analyses. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS-2010)*, pages 599–609, 2010.

[263] K. Rzadca, D. Trystram, and A. Wierzbicki. Fair game-theoretic resource management in dedicated grids. In *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGRID-2007)*, 2007.

[264] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the pangaea wide-area file system. *ACM SIGOPS Operating Systems Review*, 36(SI):15–30, Dec. 2002.

[265] L. S. Shapley. A value for n-person games. *Contributions to the theory of games*, 2:307–317, 1953.

[266] R. Sharma, A. Datta, M. D. Amico, and P. Michiardi. An empirical study of availability in friend-to-friend storage systems. In *Proceedings of the 11th International Conference on Peer-to-Peer Computing (P2P-2011)*, pages 348–351, August 2011.

[267] W. Shi and Y. Mao. Performance evaluation of peer-to-peer web caching systems. *Journal of Systems and Software*, 79(5):714–726, 2006.

[268] D. Shiryaev, L. Yu, and E. Elkind. On elections with robust winners. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2013)*, May 2013.

[269] D. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the 29th Symposium on Theory of Computing (STOC-1997)*, pages 265–274, 1997.

[270] D. B. Shmoys and E. Tardos. Scheduling unrelated machines with costs. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms (SODA-1993)*, pages 448–454, 1993.

[271] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[272] H. A. Simon. Rational choice and the structure of the environment. *Psychological Review*, 63:129–138, 1956.

[273] P. Skowron, M. Biskup, L. Heldt, and C. Dubnicki. Fuzzy adaptive control for heterogeneous tasks in high-performance storage systems. In *Proceedings of the 6th International Systems and Storage Conference (SYSTOR-2013)*, page 13, 2013.

[274] P. Skowron and P. Faliszewski. Approximating the maxcover problem with bounded frequencies in fpt time. *CoRR*, abs/1309.4405, 2013. Also presented in FIT-2014 and in The Workshop on Economic and Computational Aspects of Game Theory and Social Choice-2014.

[275] P. Skowron, P. Faliszewski, and A. Slinko. Achieving fully proportional representation is easy in practice. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2013)*, May 2013.

[276] P. Skowron, P. Faliszewski, and A. Slinko. Fully proportional representation as resource allocation: Approximability results. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-2013)*, 2013.

[277] P. Skowron, P. Faliszewski, and A. M. Slinko. Fully proportional representation as resource allocation: Approximability results. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-2013)*, 2013. Also presented in M-PREF-2012, CoopMAS-2013, and FIT-2013.

[278] P. Skowron and K. Rzadca. Exploring heterogeneity of unreliable machines for p2p backup. In *Proceedings of the 2013 International Conference on High Performance Computing & Simulation (HPCS-2013)*, pages 91–98, 2013.

[279] P. Skowron and K. Rzadca. Fair share is not enough: Measuring fairness in scheduling with cooperative game theory. In *Proceedings of the 10th International Conference on Parallel Processing and Applied Mathematics (PPAM-2013)*, pages 38–48, 2013.

[280] P. Skowron and K. Rzadca. Network delay-aware load balancing in selfish and cooperative distributed systems. In *Proceedings of IPDPS Workshops*, pages 7–18, 2013. Also presented in New Challenges in Scheduling Theory Workshop-2012.

[281] P. Skowron and K. Rzadca. Non-monetary fair scheduling: a cooperative game theory approach. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA-2013)*, pages 288–297, 2013. Also presented in CoopMAS-2013 and in New Challenges in Scheduling Theory Workshop-2014.

[282] P. Skowron, K. Rzadca, and A. Datta. People are processors: Coalitional auctions for complex projects (extended abstract). In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2014)*, May 2014. Also presented in The Workshop on Economic and Computational Aspects of Game Theory and Social Choice-2014.

[283] P. Skowron, L. Yu, P. Faliszewski, and E. Elkind. The complexity of fully proportional representation for single-crossing electorates. In *Proceedings of the 6th International Symposium on Algorithmic Game Theory (SAGT-2013)*, pages 1–12, Oct. 2013.

[284] A. Su, D. R. Choffnes, A. Kuzmanovic, and F. Bustamante. Drafting behind Akamai. *SIGCOMM Computer Communication Review*, 36:435–446, August 2006.

[285] M. Szymaniak, G. Pierre, and M. Steen. Scalable cooperative latency estimation. In *Proceedings of the 10th International Conference on Parallel and Distributed Systems (ICPADS-2004)*, 2004.

[286] K. Talwar. The price of truth: Frugality in truthful mechanisms. In *Proceedings of the 20th Symposium on Theoretical Aspects of Computer Science (STACS-2003)*, pages 608–619, 2003.

[287] A. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *J. ACM*, 32(2):445–465, 1985.

[288] R. G. Tinedo, M. S. Artigas, and P. G. López. Analysis of data availability in f2f storage systems: When correlations matter. In *Proceedings of the 12th International Conference on Peer-to-Peer Computing (P2P-2012)*, pages 225–236, 2012.

[289] L. Toka, M. D. Amico, and P. Michiardi. Online data backup : a peer-assisted approach. In *Proceedings of the 10th International Conference on Peer-to-Peer Computing (P2P-2010)*, pages 1–10, August 2010.

[290] L. Toka, M. D. Amico, and P. Michiardi. Data transfer scheduling for p2p storage. In *Proceedings of the 11th International Conference on Peer-to-Peer Computing (P2P-2011)*, pages 132–141, August 2011.

[291] D. N. Tran, F. Chiang, and J. Li. Friendstore: Cooperative online backup using trusted nodes. In *Proceedings of the 1st Workshop on Social Network Systems (SocialNets-2008)*, pages 37–42, 2008.

[292] C. Ungureanu, A. Aranya, S. Gokhale, S. Rago, B. Atkin, A. Bohra, C. Dubnicki, and G. Calkowski. Hydrafs: A high-throughput file system for the hydrastor content-addressable storage system. In *Proceedings of the 8th Conference on File and Storage Technologies (FAST-2010)*, pages 225–239, 2010.

[293] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[294] B. Veeravalli and G. Barlas. Efficient scheduling strategies for processing multiple divisible loads on bus networks. *Journal of Parallel and Distributed Computing*, 62(1):132–151, 2002.

[295] W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.

[296] K. D. Vohs, N. L. Mead, and M. R. Goode. The Psychological Consequences of Money. *Science*, 314(5802):1154–1156, Nov. 2006.

[297] K. D. Walsh, A. Sawhney, and H. H. Bashford. Cycle-time contributions of hyper-specialization and time-gating strategies in us residential construction. In *Proceedings of 11th Annual Conference on Lean Construction-2003*, pages 390–397, 2003.

[298] T. Walsh. Where are the hard manipulation problems? *Journal of Artificial Intelligence Research*, 42:1–29, 2011.

[299] C. Walter. Kryder's Law. *Scientific American*, 293:32–33, 2005.

[300] Y. Wang and A. Merchant. Proportional-share scheduling for distributed storage systems. In *Proceedings of the 5th Conference on File and Storage Technologies (FAST-2007)*, pages 4–4, 2007.

[301] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS-2003)*, pages 4–4, 2003.

[302] D. B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, Aug. 2000.

[303] A. Wierman. Fairness and scheduling in single server queues. *Surveys in Operations Research and Management Science*, 16:39–48, 2011.

[304] T. M. Williams. The need for new paradigms for complex projects. *International Journal of Project Management*, 17(5):269 – 273, 1999.

[305] E. Winter. The Shapley value. In R. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 3, pages 2025–2054. 2002.

[306] M. Wooldridge and P. E. Dunne. On the computational complexity of coalitional resource games. *Artificial Intelligence*, 170(10):835–871, July 2006.

[307] Y. Xing, Z. Li, and Y. Dai. Peerdedupe: Insights into the peer-assisted sampling deduplication. In *Proceedings of the 10th International Conference on Peer-to-Peer Computing (P2P-2010)*, pages 1–10, 2010.

[308] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the Use of Fuzzy Modeling in Virtualized Data Center Management. In *Proceedings of the 4th IEEE International Conference on Autonomic Computing (ICAC-2007)*, June 2007.

[309] R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190, 1988.

[310] H. Yaïche, R. R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Transactions on Networking*, 8(5):667–678, Oct. 2000.

[311] J. Zhang, A. Sivasubramaniam, A. Riska, Q. Wang, and E. Riedel. An interposed 2-level i/o scheduling framework for performance virtualization. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS-2005)*, pages 406–407, 2005.

[312] Z. Zhang, Q. Lian, S. Lin, W. Chen, Y. Chen, and C. Jin. Bitvault: a highly reliable distributed data retention platform. *ACM SIGOPS Operating Systems Review*, 41:27–36, April 2007.

[313] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th Conference on File and Storage Technologies (FAST-2008)*, pages 18:1–18:14, 2008.

[314] X. Zhu, X. Liu, S. Singhal, and M. Arlitt. Resource entitlement control system, January 2010. US patent application no 7644162.

[315] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin. What does control theory bring to systems research? volume 43, pages 62–69, 2009.