DISS. ETH NO. 23589

Adversarial Input in Games and Markets

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

PHILIPP BRANDES

M. Sc., University of Paderborn, Germany

born on 01.08.1986

citizen of

Germany

accepted on the recommendation of

Prof. Dr. Roger Wattenhofer, examiner
Prof. Dr. Martin Hoefer, co-examiner

2016

Abstract

We start by considering the trade-off between space and write overhead of flash memory. Every flash memory has additional space for wear leveling; we denote the space overhead with σ. Furthermore, every flash memory is forced to rewrite valid data when a block is erased; we denote the write overhead with ω. We show that space and write overhead are inversely proportional with $\sigma\omega \geq 1$. Our analysis is tight, as our algorithm achieves $\sigma\omega \leq 1$ even if the data is updated adversarially.

We study the dynamics of social networks in Chapter 3. Each node has to decide locally which other node it wants to befriend, i.e., to which other node it wants to create a connection in order to maximize its welfare, which is defined as the sum of the weights of incident edges. With the limitation that each node can only have a constant number of friends, we show that every local algorithm is arbitrarily worse than a globally optimal solution. Furthermore, we show that there cannot be a best local algorithm, i.e., for every local algorithm there exists an initial social network in which the algorithm performs arbitrarily worse than some other local algorithm. However, one can simulate a constant number of local algorithms in order to be competitive with the best of them. After looking for good friends, we try to hide unwanted content in the next chapter. Unwanted content on web pages can take many forms, be it ads, malicious code, or specific topics that the user does not want to read about (yet). We focus on the latter. The user can define terms based on which we prevent the disclosure of undesired information (e.g., the latest sports result). We define this formally as the node removal problem and show its equivalence to the $\mathcal{NP}$-hard knapsack problem. Our evaluation shows that our heuristic correctly distinguishes between wanted and unwanted content in approximately 9 out of 10 cases.

In the two last chapters, we analyze markets. First, we consider online auctions with buyback; a form of auction where bidders arrive sequentially and the bidders have to be accepted or rejected immediately. Each bidder has a valuation for being allocated the good and a preemption price for which the good can be bought back from the bidder. We study the clairvoyant model, a model sitting between the traditional offline and online models. In the clairvoyant model, a sequence of all potential customers (their bids and compensations) is known in advance to the seller, but the seller does not know when the sequence stops. We present an algorithm for computing the difficulty Δ, the optimal ratio between the clairvoyant mechanism and the offline mechanism. If the number of goods is unbounded, then the problem in the clairvoyant model is $\mathcal{NP}$-hard. We show that there is a tight gap of $\Theta(\Delta^5)$ between the offline and the online model if the mechanisms know the difficulty Δ of the input sequence. In our last chapter, we analyze the job market. Frey and Osborne quantified the automation of jobs, by assigning each job a probability to be automated. We refine their results in the following way: Every job consists of a set of tasks, and these tasks can be related. We use a linear program to assign probabilities to tasks, such that related tasks have a similar probability and the computerization probabilities of

the tasks can explain the computerization probability of a job. Analyzing jobs on the level of the much more concrete tasks helps comprehending the results. Our approach allows us to automatically detect inconsistencies in the results of Frey and Osborne.

Zusammenfassung

Wir beginnen, indem wir den Kompromiss zwischen Speicher- und Schreib-Overhead von Flashspeicher betrachten. Jeder Flashspeicher hat zusätzlichen Speicherplatz für Wear Leveling; diesen bezeichnen wir Speicher-Overhead σ. Weiterhin ist jeder Flashspeicher gezwungen gültige Daten erneut zu schreiben, wenn ein Block gelöscht wird; wir bezeichnen den Schreib-Overhead mit ω. Wir zeigen, dass Speicher- und Schreib-Overhead invers proportional sind mit $\sigma\omega \geq 1$. Unsere Analyse ist scharf, da unser Algorithmus $\sigma\omega \leq 1$ erreicht, selbst wenn die Daten nachteilig aktualisiert werden.

Wir analysieren zusätzlich die Dynamik in sozialen Netzwerken in Kapitel 3. Jeder Knoten muss lokal entscheiden, welchen Knoten er befreunden möchte. Er muss also entscheiden zu welchem anderen Knoten er eine Verbindung erstellen will um seine Wohlfahrt, die als Summe der inzidenten Kanten definiert ist, zu maximieren. Mit der Beschränkung, dass jeder Knoten nur eine konstante Anzahl von Freunden haben kann, zeigen wir, dass jeder lokale Algorithmus beliebig viel schlechter als eine globale, optimale Lösung ist. Ferner zeigen wir, dass es keinen besten lokalen Algorithmus gibt. Es gibt für jeden lokalen Algorithmus ein initiales soziales Netzwerk, bei welchem er beliebig viel schlechter als ein anderer lokaler Algorithmus ist. Man kann jedoch eine konstante Anzahl von lokalen Algorithmen simulieren um mit dem besten von ihnen kompetitiv zu sein. Nachdem wir gute Freunde gesucht haben, versuchen wir im folgenden Kapitel unerwünschten Inhalt zu verstecken. Unerwünschter Inhalt auf einer Webseite kann viele Formen annehmen, sei es Werbung, bösartiger Code oder spezifische Themen über die der Nutzer (noch) nichts lesen will. Wir konzentrieren uns auf den letztgenannten Punkt. Der Nutzer kann Begriffe angeben und basierend auf diesen verhindern wir das Anzeigen von unerwünschtem Inhalt (z.B. das aktuelle Sportergebnis). Wir definieren dieses Problem formal als das Knoten-Entfern-Problem und zeigen seine Äquivalenz zu dem $\mathcal{NP}$-harten Rucksack-Problem. Unsere Evaluation zeigt, dass unsere Heuristik in 90% der Fälle korrekt zwischen erwünschtem und unerwünschtem Inhalt unterscheidet.

In den letzten zwei Kapiteln analysieren wir Märkte. Zuerst betrachten wir Online-Auktionen mit Rückkauf; eine Form von Auktion bei der Bieter nacheinander erscheinen und die Bieter unmittelbar akzeptiert oder abgelehnt werden müssen. Jeder Bieter hat eine Bewertung dafür das Gut zu erhalten und einen Verdrängungspreis für welchen das Gut vom Bieter zurückgekauft werden kann. Wir analysieren das hellseherische Modell, ein Modell welches sich zwischen dem traditionellen Offline- und dem Online-Modell befindet. In dem hellseherischen Modell ist dem Verkäufer die Sequenz an potentiellen Kunden im voraus bekannt, aber der Verkäufer weiß nicht, wann die Sequenz endet. Wir präsentieren einen Algorithmus um den Schwierigkeitsgrad Δ, das optimale Verhältnis zwischen dem hellseherischen Mechanismus und dem Offline-Mechanismus, zu berechnen. Wenn die Anzahl Güter unbeschränkt ist, dann ist das Problem im hellseherischen Modell $\mathcal{NP}$-hart. Wir zeigen, dass es eine scharfe Lücke von

$\Theta(\Delta^5)$ zwischen dem Offline- und dem Online-Modell gibt, wenn der Mechanismus den Schwierigkeitsgrad Δ weiß. In unserem letzten Kapitel analysieren wir den Arbeitsmarkt. Frey und Osborne haben die Automatisierung von Arbeit quantifiziert, indem sie jedem Beruf eine Automatisierungswahrscheinlichkeit gegeben haben. Wir verfeinern ihre Ergebnisse auf die folgende Art: Jeder Beruf besteht aus einer Menge von Aufgaben und diese Aufgaben können verwandt sein. Wir nutzen ein lineares Programm um diesen Aufgaben eine Wahrscheinlichkeit zuzuweisen, so dass verwandte Aufgaben eine ähnliche Wahrscheinlichkeit haben und dass die Automatisierungswahrscheinlichkeiten der Aufgaben die eines Berufs erklären. Berufe auf diesem viel konkreteren Level zu analysieren hilft dabei die Ergebnisse zu verstehen. Unser Ansatz erlaubt es uns automatisch Inkonsistenzen in den Ergebnissen von Frey und Osborne zu detektieren.

Acknowledgements

My time as a PhD student in the Distributed Computing Group was an interesting experience. I enjoyed the opportunity to explore all kinds of weird ideas, even if most of them cannot exactly be called successful. The thesis that you are now reading would not have been possible without the help of many people that I would like to thank in the following.

First, I want to thank my supervisor Roger Wattenhofer for giving me the opportunity to write my thesis in his group and supporting me throughout the time. He allowed me to test my ideas on lots of students, which resulted in quite a few interesting projects.

Then, I would also like to thank my co-referee Martin Hoefer for taking the time to review this thesis and to serve on my committee.

Furthermore, there are also the other people that made my time in the Distributed Computing Group a wonderful experience; my colleagues and coworkers. I want to thank (in alphabetical order) Barbara Keller for always remembering Philipp street, Beat Futterknecht for having a guy for every problem I ever encountered, Benny Gächter for "svn pls", Christian Decker for explaining Bitcoin to me and helping me with so many of my programming problems, Darya Melnyk for entertaining me during my last few months at disco, David Stolz for his wisdom, Friederike Brütsch for offering free chocolate and always lending me an ear, Georg Bachmeier for the rage, Jara Uitto for the extremely entertaining first few months in Zurich, Jochen Seidel for having BBQ at his place more often than I can count, Katie Howard for showing me the pubs and bars of Zurich, Klaus Förster for being the kindest person that I know (and for having pitch black sense of humor), Laura Peer for her amazing driving skills, Manuel Eichelberger for being our 9am alarm clock, The King (Michael König) for creating great software that does not even require a database, Pankasch Khanchandani for nearly spelling my name correctly, Pascal Bissig for making me cringe regularly; years after I thought you couldn't do that anymore, Tobias Langner for showing me that it is possible, Samuel Welten for being one of the founding members of our morning coffee and great emails, Sebastian Brandt for always getting it again, Yuval Emek for being an excellent nanny, Yuyi Wang for nearly always going to the correct exercise.

Last but not least, I want to thank the people most important in my life: My parents for nurturing the nerd in me; Christoph and Sibylle for allowing me

to visit sunny Florida whenever the weather here was horrible and being perfect hosts; and Andre, Eva, and Ole whose (near) constant willingness to provide me with (funny) cats and moral support was vital to help me get through the more difficult times. I will always be grateful for that. Thank you!

Contents

1 Introduction **1**
1.1 Collaborations and Contributions 4

2 Space and Write Overhead **6**
2.1 Introduction 6
2.2 Related Work 7
2.3 Model 8
2.4 Cycling Algorithm 10
2.5 Lower Bound 11
2.6 Different Access Pattern 14
2.7 Conclusion 15

3 Finding Friends **16**
3.1 Introduction 16
3.2 Model 19
3.3 On Welfare 20
3.4 Conclusion 30

4 Hiding Adversarial Content **31**
4.1 Introduction 31
4.2 Related Work 32
4.3 Model 33
4.4 $\mathcal{NP}$-hardness 34
4.5 Concept 36
4.6 Evaluation 37
4.7 Conclusion 39

5 Clairvoyant Mechanisms **41**
5.1 Introduction 41
5.2 Related Work 43
5.3 Model 44
5.4 Auctioning Off a Single Good 46
5.5 Auctions with Several Goods 54
5.6 Conclusion 58

6 Automatable Jobs and Automatable Tasks **59**
6.1 Introduction 59
6.2 Related Work 60
6.3 Model 61
6.4 From Task Frequencies to Task Shares 62
6.5 From Jobs to Tasks 64
6.6 Linear Program Results 65
6.7 Further Analysis 71
6.8 Conclusion 73

7 Conclusion **74**

1

Introduction

> *To the extent we have been successful, it is because we concentrated on identifying one-foot hurdles that we could step over rather than because we acquired any ability to clear seven-footers.*
>
> Warren Buffet

Warren Buffet has been very successful with his approach of only tackling easy problems. He has made billions of dollars by investing in the right companies. But not all of us have the luxury to be able to pick and choose our problems. We all face difficult problems from time to time. Solving those correctly takes a lot of time and effort, but neither guarantees success. Luckily, we can make the dreaded seven-foot hurdles as easy as one-footers simply by practicing. We know that "practice makes perfect". A problem that a beginner considers difficult is no challenge for an expert; and to become an expert one needs to practice. Thus, people spend hours practicing – playing a musical instrument, writing a research paper, solving a Rubik's cube, or performing any other skill. It even has been quantified that it takes about 10,000 hours of practice to become an expert [57]. As a comparison, we work around 2,000 hours each year. This approach is therefore only feasible for at most a handful of skills.

Practicing a song for so long just to be able to play it once is clearly not the most efficient way. There are ways for a beginner to solve a difficult problem by using shortcuts or tricks; tricks that do not require putting in that many hours. Instead of playing the original song, a simplified version might sound nearly as good – or maybe one could hire a band. Instead of $\mathcal{P}$ vs $\mathcal{NP}$, one could tackle easier problems or develop heuristics. Instead of solving a Rubik's cube, one could just tear it apart and rebuild it from scratch. Not every trick works for every problem; but for most problems, there are shortcuts. This raises the question: Why deal with the worst case if there are good and easy ways around

it? And more importantly, how do we find those? This is the main thread of this thesis. We analyze a wide set of games with a focus on adversarial input. After showing that these are indeed difficult problems, we look for shortcuts or tricks. We find the equivalent of playing a simplified version of a song for some problems; others, we take apart to rebuild them from scratch.

We start by considering write overhead in flash memory in Chapter 2. A problem that has a simple, yet effective solution. Flash memory has become ubiquitous due to its ever falling price and massive speed advantage over traditional hard disk drives. In its early days, flash memory had the reputation of having a short lifespan since every block of the disk can only be written to about 10,000 times. Flash memory has another peculiar property. The data is stored in pages, which are grouped in blocks; and a page cannot be erased individually. A block must always be erased as a whole. Data stored in a block that is being erased must be moved to other pages to avoid data loss. Hence, more data is being written and the lifespan of the disk is decreased even further. So called flash translation layer (FTL) algorithms apply heuristics to even out the usage of each block and thus alleviate this problem. Nevertheless, every FTL algorithm causes write overhead ω – intuitively unnecessary writes – and every flash memory must have space overhead σ – intuitively unused space, but space available for FTL algorithms. These two must fulfill $\sigma\omega \geq 1$ if the data is updated adversarially. Thus, one should either reserve a lot of free space for the FTL algorithm or accept that the lifespan of the disk is very short. We also present a simple algorithm with $\sigma\omega \leq 1$, i.e., that matches this bound.

We continue with a game that at a first glance does not include an adversary: how to find and stay friends. Our analysis starts with the observation that we spend time with people whose company we enjoy, i.e., good friends, and not with people whose company we do not enjoy, i.e., bad friends. Furthermore, we can only spend a certain amount of time with our friends. Combined, this means that we need to prioritize and thus choose the best friends possible. We formalize this setting in Chapter 3. People are modeled as nodes in a graph in which edges represent friendships. A friendship between two nodes u, v has a quality $q(u, v) \in [0, 1]$. Every node can only have a limited number of friends and might need to sever an old friendship to create a new one. A new friendship between nodes can only be created if they both want to be friends and if they might end up being invited to the same event; formally if they are within ℓ hops of each other. If the input, i.e, the initial friendship graph, is created by an adversary, then any local algorithm to find friends performs arbitrarily worse than a global algorithm that is not restricted to finding friends at most ℓ hops from its current friends. We also prove that for any local algorithm, there exists an initial friendship graph such that it performs arbitrarily worse than another local algorithm. We slightly alleviate this problem by proposing an algorithm that uses its memory to simulate several algorithms at the same time. The output of this is, despite the local view of the nodes, at least half as good as the best solution of the simulated algorithms.

In the next chapter, Chapter 4, we look at the more practical problem of filtering content on web pages. Whereas this already works well for ads and

malware, topic specific filtering only exists for a small set of topics. We want to give the user the possibility to personalize her web experience by entering the topics that she does not want to read about. This problem is analyzed from a theoretical and practical point of view. We show that a formalized version of this problem is $\mathcal{NP}$-hard. Thus, if the content is placed on the web page by an adversary, solving it exactly would make the web surfing experience very unpleasant. Hence, the Firefox extension that we developed uses a heuristic to remove unwanted content. In our evaluation we show that it removes more than 90% of the unwanted content while removing less than 15% of the good content.

In Chapter 5 we turn towards markets, more specifically towards online auctions. A good is sold at an auction – without requiring all the bidders to be present at the same time. If a bidder shows up and bids for the good, we want to give her an answer immediately. After the good is "sold", our mechanism can buy back the good from the bidder by paying her preemption price. Thus, if we want to reallocate the good to another bidder, the new bidder must pay enough to allow the mechanism to buy back the good. Any online auction fares badly against an offline mechanism that knows the sequence of bidders and simply picks the best one. The following example illustrates the problem: the first bidder bids \$1 for the good and has an extremely high preemption price. If the good is not allocated to her, no other bid is made. If she is allocated the good, a second bidder is willing to pay much more, but not enough to pay the preemption price of the first bidder. An online mechanism will thus always make the wrong choice. We do the equivalent of taking the problem apart to rebuild it from scratch by analyzing it in the clairvoyant model, a hybrid between an online and offline model. In this model the mechanism knows the input sequence, but has to be prepared that it stops at any time. Thus, it must always have a "good" solution. We present a mechanism that solves this setting optimally and is always at most a factor of Δ worse than the optimal offline solution. Note that Δ depends on the input – an input that is chosen by an adversary. We also analyze online mechanisms that know Δ, but nothing else about the input sequence. This class of mechanisms is $\Theta(\Delta^5)$ competitive with respect to the optimal offline solution.

In the next chapter, Chapter 6, we analyze one of the largest markets in the world – the job market. With the recent advance in machine learning there are jobs where no one spends 10,000 hours to become an expert any more – a cheap and easily available expert already exists: our computer. This phenomenon, called computerization, has already left its marks on the job market and is expected to cause an upheaval in the job market and thus the economy as a whole. Frey and Osborne were the first to quantify this change and came to the conclusion that 47% of US employment is at risk of being automated in the next few decades [52]. They provide a probability for every job in the O*NET database that this job will be automated. Frey and Osborne may state that a job has a 87% automation probability, but leave the reader wondering what this actually means. We reinterpret their results and break them down to the tasks of a job. We assign each task an automation probability such that 87% of the job are automatable. Thus, the reader knows which parts of a job will be automated

in the next few decades. The additional depth of automation probabilities for the much more concrete tasks sheds light on their results. During our evaluation, we detect a few inconsistencies in the probabilities by Frey and Osborne.

We summarize our findings and draw a conclusion in Chapter 7.

1.1 Collaborations and Contributions

This thesis is based on several publications and drafts I worked on during my time as a PhD student at the Distributed Computing group at ETH Zurich under the supervision of Prof. Dr. Roger Wattenhofer. Research often is a collaborative effort. I shall list in the following the publications or drafts underlying the respective chapters along with a list of the respective co-authors. Note that the authors of the papers are listed in alphabetical order and therefore do not represent the degree of contribution of the individual authors. Besides this, all articles listed below are joint work with my supervisor Prof. Dr. Roger Wattenhofer.

Chapter 2 is based on the publication *Space and Write Overhead are Inversely Proportional in Flash Memory* [23].

Chapter 3 is based on the publication *On Finding Better Friends in Social Network* [22].

Chapter 4 is based on the publication *Spoilers Ahead - Personalized Web Filtering* [17]. Co-authors were Pascal Bissig and Roman Willi.

Chapter 5 is based on the publication *Clairvoyant Mechanisms for Online Auctions* [21]. Co-authors were Zengfeng Huang and Hsin-Hao Su.

Chapter 6 is based on the publication draft *Opening the Frey/Osborne Black Box: Which Tasks of a Job are Susceptible to Computerization?*.

2

Space and Write Overhead

2.1 Introduction

We start with a game that features high cost if the input is adversarial.

Flash memory is omnipresent in smartphones and cameras. Solid state disks (SSDs) based on NAND flash increasingly become the default choice for computers of any kind, from laptops to servers. They massively outperform traditional hard disk drives, especially in random access. Unfortunately, NAND flash is also known for having a limited lifetime, as the multi-level cells used to physically store data can only be written and erased about 10,000 times [2]. Afterwards, cells can no longer reliably store data. As a result, despite not having moving mechanical parts, flash memory is perceived as more failure prone than traditional hard disk drives.

This problem is aggravated due to another technical property of flash memory. The smallest storage unit, a page, cannot be erased on its own, but only with the rest of the pages of a block. A block usually consists of 64 pages, with each page being able to store up to 2,048 bytes [94–96]. When a block is erased to create free pages, the valid data stored in the block needs to be preserved, i.e., written to other pages in different blocks to avoid data loss. These writes are not issued by the user but by the system, and are as such not necessary from the user's point of view. We call them *write overhead*. Since additional writes decrease the lifetime of the disk, write overhead should be minimized. Flash memory is *worn out*, if data can no longer be written on it reliably, i.e., after at least one block has been erased 10,000 times.

If some data gets updated often and is always stored in the same physical block, this block will be worn out quickly. Such an access pattern is common in a real world environment [30, 31]. To alleviate this, a logical data address is mapped to a physical page address using a flash translation layer (FTL). This

allows to distribute writes more evenly over the blocks. Using a FTL can also improve write overhead significantly.

In Section 2.2, we will discuss various advanced heuristic algorithms that manage to extend the lifetime of flash memory by moving data in a sophisticated way. The internal storage capacity of the hardware is generally higher than announced to the consumer, and this *space overhead* is used to move data less often. It is known that write overhead and space overhead are related; in general this relation depends on the application that is using the flash memory.

We study the trade-off between *write overhead* ω and *space overhead* σ in Section 2.5. We show that there is an inversely proportional law connecting the two. If we assume that the data is updated adversarially (by an application that accesses data in a worst-case way), any FTL algorithm must obey

$$\sigma\omega \geq 1.$$

Moreover, in Section 2.4 we show that this inequality is tight: Our FTL algorithm, the cycling algorithm, can achieve equality, and hence is worst-case optimal.

In Section 2.6 we analyze the cycling algorithm with an average-case analysis. More concretely, we assume that some data is static and never updated. The remaining data is dynamic and updated uniformly at random. In this setting the cycling algorithm achieves $\sigma\omega < 1$, more precisely the write overhead is at most $\frac{\mu \cdot T + \delta \cdot T \cdot e^{\frac{1-\alpha}{\alpha\delta}}}{T - (\mu \cdot T + \delta \cdot T \cdot e^{\frac{1-\alpha}{\alpha\delta}})}$ with μ, δ, and φ being the fraction of the flash memory that is filled with static data, dynamic data, and no data, respectively.

2.2 Related Work

As mentioned in the introduction, FTLs have been around as long as flash memory. Thus, we can only provide a shortened overview in this section.

The work closest to ours, as in that it also considers worst cases analysis, is by Ben-Aroya and Toledo [16]. In their model the flash memory has T blocks out of which only V are visible to the user, and the remaining ones solely exist for the sake of wear leveling. They show that there exists for every algorithm an input sequence such that the algorithm can fulfill at most $(T - V + 1) \cdot H$ write requests, with H being the maximal number of writes a block can endure. They assume that every block has exactly one page. Since this guarantees that there is always a block that has either only free pages or only invalid pages, this greatly simplifies the problem. It is therefore not necessary preserve the valid data stored in the block, i.e., write it to a page in another block to avoid data loss. Hence, write overhead does not exist.

Some papers assume a uniform distribution of write accesses and then try to find a closed form expression for write amplification depending on the over-provisioning [2, 118]. They use powerful (and therefore in practice rather slow) FTL algorithms that can erase the block with the most invalid pages if no block with a free page exists. Some work analyses specific strategies, e.g., choosing the "best" block from a sliding window, again using a uniform distribution to

model the write accesses [73]. In Section 2.6, we use a similar access pattern. We assume that some data is static and never updated and the remaining data is updated uniformly at random. But our focus is the trade-off between space and write overhead in an adversarial setting. The trim command, i.e., marking pages as invalid, and its effects on write amplification, are also studied under a uniform write distribution [51]. Non-uniform write distributions are analyzed in [44]. They employ the least recently written strategy, which is similar to our cycling algorithm.

FTL heuristics have been around right from the start. In addition to the mostly secret implementations by hardware manufacturers, FTL algorithms have been published as well. We can only list a small subset of these algorithms. The so called dual-pool algorithm tries to store cold data, i.e., data that is not accessed much, in blocks which have been erased often [28]. This work has been refined later [29–31]. Another approach is to use a log buffer based FTL. This is a hybrid between page-level mapping and block-level mapping. Requests are written to log blocks. Once such a block is filled, the data is merged with other data from the same LBA and written into a new block, thus allowing block-level mapping. A popular FTL algorithm is commonly referred to as FAST [88]. A survey about FTL algorithms can be found in [35].

With the emergence of PCM-based memory, wear leveling remains an important topic even beyond current NAND flash [106]. Low level technical details about flash memory can be found in [94–96]. There exists a journaling file system that is developed exclusively with flash drives in mind [117].

Since details about flash memory and SSDs in particular were initially trade secrets and thus not accessible to research, simulators were written to compensate for the secrecy of the manufacturers. One of them, CPS-SIM [87], takes the number of buses and low level clocks into account. DiskSim, a disk simulator framework, was extended to include energy usage by [80] and parallelism and write ordering by [3].

2.3 Model

The flash memory we consider consists of n blocks $b_0, \ldots, b_{n-1}$. Each block b_i in turn consists of m pages $p_0^i, \ldots, p_{m-1}^i$. A page is the smallest accessible unit, i.e., a page can only be accessed (read or written) as a whole. The physical pages in flash memory store data.

A physical page continuously cycles through three distinct states: *free*, *valid*, and *invalid*. First the page is free, and data can be written into it. After data was written into the page, the page is valid as it stores useful data. Later the data might be updated, and stored in a free flash memory page. The old page storing the old version of that data becomes invalid. However, the old page cannot be written again, first it needs to be erased. Because of technical limitations of NAND flash, all pages in a block must be erased together. Before a block can be erased, all valid pages must first be moved (written to other pages). After a block is erased, all pages in the block are free again, and a life cycle of the page is complete.

Let T be the total number of pages of our flash memory, that is, $T = m{\cdot}n$. Let V denote the maximum number of valid pages that can be stored, the capacity available to the user. The ratio $\alpha = \frac{T}{V} > 1$ is referred to as *over-provisioning*. We denote the *space overhead* with $\sigma = \frac{T-V}{V} = \alpha - 1$. This σ is the relative amount of storage that is hidden from the user, and used to improve the lifetime of our flash memory.

Since blocks can be erased only about 10,000 times, unbalanced data access will wear out some blocks earlier than others. The flash translation layer (FTL) maps logical addresses to physical pages.[1] This mapping is not static but evolves over the life time of the flash memory, and is determined by the FTL algorithm. Note that the FTL resides inside the flash memory and is invisible to the user.

To write data on flash memory, a *write request* must to be issued. A write request contains the data and a logical address. The task of an FTL algorithm is to accept write requests and choose for each write request a physical page, where the data will be stored. We distinguish between two types of write requests. To write new data or update existing data, an *external* write request will be issued. Apart from Section 2.6, we assume that an adversary issues these external write requests. The adversary knows the FTL algorithm and the current state of the flash memory at any moment. Whenever a block b_i that still contains valid pages will be erased, these valid pages first need to be written to a free page to avoid data loss. Thus, any FTL algorithm also needs to issue *internal* write requests to avoid data loss.

Let E_k^i and I_k^i be the number of external and internal write requests written into a physical page p_k^i of block b_i, respectively. Thus, the total number of external write requests is $E = \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} E_k^i$ and the total number of internal write requests is $I = \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} I_k^i$. This allows to define the average number of external write requests per physical page as E/T. The *local write overhead* ω_k^i of page p_k^i is $\frac{E_k^i + I_k^i}{\frac{E}{T}} - 1$, i.e., the total number of write requests over the average number of write requests minus 1. The minus 1 is included in the term since we focus on the overhead. We can now define the *write overhead* ω as

$$\begin{aligned} \omega &= \max_{0 \le i \le n-1, 1 \le k \le m} \omega_k^i \\ &= \max_{0 \le i \le n-1, 0 \le k \le m-1} \left\{ \frac{E_k^i + I_k^i}{\frac{E}{T}} - 1 \right\}. \end{aligned}$$

Intuitively, this is means that we are interested in the worst page. In the literature, *write amplification* A is defined as as the total number of write requests W over the total number of external write requests, i.e., $A = \frac{W}{E} = \frac{E+I}{E}$.

We assume that the flash memory has a volatile memory (RAM) of two blocks to temporarily store m incoming write requests and every page of one single block. Hence, after buffering m write requests a block b_i can be read and its valid pages stored in the buffer, then block b_i can be erased, and finally the m write requests can be written into block b_i.

[1] Some FTLs instead use a block based mapping scheme, where flash memory is addressed on the level of blocks. In this chapter we address memory on the page level.

2.4 Cycling Algorithm

We now present the *cycling algorithm*. It keeps track of the most recent block b_i it has written data into. After m write requests, these write requests are written as a whole to the next block b_j with $j = i + 1 \bmod n$. For every valid page of this block a new internal write request is issued to avoid data loss. Note that if b_j contains only valid pages, then all these pages are written into the next block $b_{j'}$ (with $j' = i + 2 \bmod n$). This process is repeated until less than m write requests are buffered. This must eventually happen because we have $T > V$. The pseudocode is presented in Algorithm 2.1. This algorithm is also known as circular buffer scheme and a special case of the algorithm presented in [73]. Note that buffering the data and writing m write requests at once is not necessary, but it simplifies the proof. Before we start, we need a small helper lemma.

Algorithm 2.1 Cycling

```
1: i ← 0
2: for every write request do
3:     while number of outstanding write requests ≥ m do
4:         issue write request for every valid page from block b_i
5:         erase block b_i
6:         write data from write requests into b_i
7:         i ← i + 1 mod n
8:     end while
9: end for
```

Lemma 2.2. *The write overhead ω is always larger than the ratio between the total number of internal and external write requests, i.e., $\omega \geq \frac{I}{E}$.*

Proof. To see that the inequality holds consider the average local write overhead of an arbitrary page p_ℓ^j of block b_ℓ. We have $E_\ell^j = E/T$ and $I_\ell^j = I/T$ and therefore $\omega_\ell^j = \frac{E_\ell^j + I_\ell^j}{\frac{E}{T}} - 1 = \frac{E/T + I/T}{E/T} - 1 = \frac{I}{E}$. The pigeon hole principle now yields that there must be a physical page that has local write overhead of at least $\frac{I}{E}$. □

Theorem 2.3. *The cycling algorithm guarantees $\sigma\omega \leq 1$.*

Proof. To see this, consider one run from "left to right", i.e., from block b_0 to block b_{n-1}. We briefly show data that is not updated can only cause one internal write request per run. W.l.o.g. let b_i be the block in which the data is stored. We now analyze what happens when this block is erased. Since the corresponding page is valid, this causes an internal write request. Thus, the data is later written to some block b_j with $j > i$. The next write requests are written into block b_{j+1}. Thus, no second write request for the data is issued in this run.

Data that is updated does not cause an internal write request after it has been updated. W.l.o.g. let us assume that this data was written in block b_i. The next block into which data is written in this run is block b_j with $j > i$. Thus, the data stored in b_i does not cause an internal write request.

In the beginning, there are V valid pages and thus $T - V$ free or invalid pages. While writing T pages, we have to issue up to V internal write requests to avoid data loss. Thus, in one cycle the T write requests are made up by V internal and E external write requests. Since exactly the same number of write requests is written into each block, the average local write overhead is identical to the write overhead. Thus, we obtain $\omega = \frac{I/n+E/n}{E/n} - 1 = \frac{(V)/n+(T-V)/n}{(T-V)/n} - 1 = \frac{V}{T-V} = \frac{V}{V(\alpha-1)} = \frac{1}{\alpha-1} = \frac{1}{\sigma}$. □

2.5 Lower Bound

We now show that there is no algorithm that can be better than our simple algorithm in the worst case, i.e., when the data is updated adversarially. But before we continue, let us introduce a few terms. We define d to be $\frac{1}{\alpha} \cdot m$. A block is called *sparse* if it has less than d valid pages, and is called *dense* if it has at least d valid pages. We assume w.l.o.g. that d is an integer. Note that each block can be dense, i.e., contain at least d valid pages because we have up to V valid pages in total. We assume for technical reasons that there are m additional valid pages stored on the flash memory. Thus, even when the algorithm has buffered m write requests, every block can still be dense.

Theorem 2.4. *No algorithm can guarantee $\sigma\omega < 1$.*

Proof. We will first make a few simplifying assumptions that will be lifted later on. We assume that every block contains exactly the same number of valid pages, i.e., $d = \frac{1}{\alpha} \cdot m$ valid pages in the beginning and we assume that every algorithm writes only once its buffer is full. Furthermore, we do not allow static wear leveling, i.e., moving valid pages from one block to another without an external write request.

The main idea of this proof is that the adversary can choose which page becomes invalid by updating the corresponding data. Thus, he is able to always invalidates pages in such a way that every block remains dense. Thus, only a "few" external write requests can be written into a block until there are no free pages left in a block. Because the block is still dense, "a lot" of internal write requests have to be issued to avoid data loss. Thus, the write overhead is "high" and therefore also the product of space and write overhead.

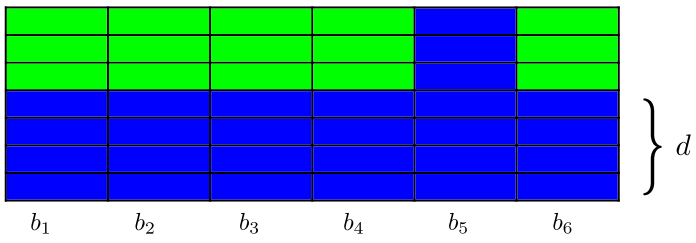


Figure 2.5: The green pages are free, the blue pages are valid. The adversary will invalidate pages from block b_5, since it is the only block with more than d valid pages.

An example is shown in Figure 2.5. The pages in block b_5 will be invalidated, since this is the only block that has more than d valid pages.

Recall that I denotes the total number of internal write requests and E denotes the total number of external write requests. As shown in Lemma 2.2, we have $\omega \geq \frac{I}{E}$. Thus, it suffices to show that $\frac{I}{E} \geq \frac{1}{\sigma}$ holds. We do this by carefully accounting for the number of external and internal write requests per block and showing that no block achieves a better ratio.

When any algorithm writes m write requests into a block, it has to issue d internal write requests to preserve the d valid pages. Note that this means that the buffer of the algorithm is not empty, but contains d write requests. From now on, it can accept $m-d$ external write requests until its buffer is filled and it writes m pages. Thus, the write overhead is $\frac{d}{m-d} = \frac{\frac{1}{\alpha}m}{m-\frac{1}{\alpha}m} = \frac{\frac{1}{\alpha}}{1-\frac{1}{\alpha}} = \frac{1}{\alpha-1} = \frac{1}{\sigma}$. This is equivalent to $\sigma\omega = 1$.

We start by lifting the assumption that any algorithm writes data only when its buffer is full. The proof above can easily be adapted such that the adversary no longer invalidates pages from the one dense block, but from any dense block. Note that such a block must always exist since the average number of valid pages in a block is d. If we now consider the write overhead of each block separately, it is easy to see that the same approach works with more fine grained writing/invalidating. Every block can accept at most $m - d$ external write requests and issues d additional internal write requests when it is erased. Thus, we obtain $\omega = d/(m - d) = \frac{1}{\sigma}$.

Next, we allow the data to be unevenly distributed on the flash memory in the beginning. Note that, once again, the adversary only invalidates pages of dense blocks. These pages now need to be written into a new block. Let b_i be the block selected by the algorithm. If b_i is dense, then the algorithm can only write $m - d$ pages into it before it is full. If b_i is now erased, then there were at most $m - d$ external write requests but m write requests in total. Thus, the write overhead is $d/(m - d) = \frac{1}{\sigma}$. Let b_i be a sparse block. The algorithm can write up to d pages per sparse block and thus up to $n \cdot d$ pages in total in sparse blocks until there is no sparse block left. Note that the adversary will not invalidate a page of a sparse block and thus will not create a sparse block. Hence, at most V external write requests can be written and therefore it does not affect the asymptotic behavior.

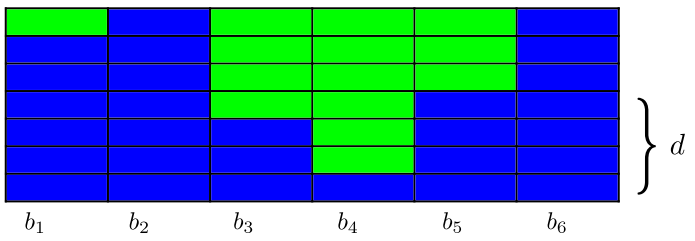


Figure 2.6: The green pages are free, the blue pages are valid. The adversary will invalidate pages from block b_1, b_2, b_6, since they have more than d valid pages.

An example is shown in Figure 2.6. Every block that has more than d valid

pages will have at least one page invalidated. In our example this means 2 pages from block b_1, 3 pages from block b_2, and 3 pages from block b_6.

We continue by showing how to lift the assumption that the algorithm does not perform static wear leveling. We start with the simple case that an algorithm erases blocks while they still have free pages. If algorithm $\mathcal{A}$ chooses to erase this block prematurely, then there were at most $m-d-1$ external write requests written to this block. Since the adversary ensures that every block is dense, there are d internal write requests. Thus, the write overhead is at least $d/(m-d-1) > d/(m-d)$ and therefore $\sigma\omega > 1$.

We now focus on static wear leveling, i.e., valid pages being moved between blocks. We prove our result by carefully accounting for the external and internal write requests. Let k be the number of pages that have been moved from block b_i before block b_i needs to be erased. Note that if this block is not erased, then it is clear that issuing the k internal write requests only increases the write overhead. Thus, we can assume that block b_i is erased later on.

If pages from a dense block are moved and the block remains dense, then it is easy to see that when this block is erased at most $m-d$ external write requests can be written into this block and d internal write requests are issued. Thus, we obtain the familiar expression $d/(m-d)$ for the write overhead. The same argument holds if the page becomes dense again before it is being erased.

Hence, let $s < d$ be the number of valid pages when block b_i is erased. We proceed by showing that between now and the last time this block was erased, its write overhead is $\frac{1}{\sigma}$. Since the block started with d valid pages, it could only accept $m-d$ external write requests. Because of the static wear leveling at least $d-s$ internal write requests were issued. To preserve the data another s internal write requests were issued. Thus, the total number of internal write requests issued for data from this block is at least $(d-s)+s = d$. Combined this yields $\omega = ((d-s)+s)/(m-d) = d/(m-d) = \frac{1}{\sigma}$. Hence, static wear leveling has not decreased the write overhead. But now the block contains only s valid pages. We will now proceed to show that the result of it also does not decrease the write overhead.

We now show that between now and until the next time block b_i is erased, the write overhead is at least $\alpha/(\alpha-1)$. Keep in mind that the adversary will continue to only invalidate pages from dense blocks. Thus, this block will inevitably become dense. Until the next time this block is erased, there can be $m-s$ many external write requests and there are at least d many internal write requests. We consider the $d-s$ pages that were moved and denote them *emigrant* pages. It is easy to see that each block that stores an emigrant page can accept one less external write request before being erased. We account these to block b_i. Thus, there are $m-s-(d-s) = m-d$ many external write requests. The number of internal write requests is d (simply to preserve the valid pages). Thus, we obtain $\omega = d/(m-d)$. This yields the claim. □

2.5.1 Total Number of Pages Written

Note that the blocks of a flash memory need to be used evenly to maximize the life of the flash memory. In a perfect scenario flash memory with T pages and

each block being able to withstand H writes, then this flash memory can endure up to $H \cdot T$ pages being written before its end of life. Due to the fact that not every block is worn equally and write overhead, this cannot be achieved.

It is easy to see that the cycling algorithm wears the blocks out evenly. Hence, the write overhead is the only criteria, which determines the how many write requests can be written. Since its write overhead is optimal, the cycling algorithm can write a $\frac{T \cdot H}{\omega+1}$ pages. This leads to the following corollary.

Corollary 2.7. *The cycling algorithm maximizes the amount of data written on the flash memory in the worst case.*

2.6 Different Access Pattern

Until now we have assumed that an adversary chooses which page is invalidated and we have given the adversary complete knowledge about the employed wear leveling algorithm. This is a rather pessimistic point of view. Thus, we now assume a simple access pattern and analyze the performance of the algorithm described above. It is easy to see that if data is simply written sequentially and the "oldest" data is always being invalidated, then the cycling algorithm is optimal and has no write overhead (independent of the space overhead). Hence, the effect of the access pattern should not be neglected.

2.6.1 Uniform Write Access Pattern

Let us describe a more realistic write access pattern, albeit a very simple one: the logical address of a write request is chosen uniformly at random. We say that a write request is random when its logical address is chosen uniformly at random. Let us give an intuition why this write access pattern should positively influence the write overhead. The worst case analysis assumed that while cycling from block b_0 to b_{n-1}, V internal write requests were issued. But if the write requests are uniform at random, some pages will inevitably be invalidated and thus it is not necessary to issue an internal write request for these.

Lemma 2.8. *The cycling algorithm has an expected write overhead of at most* $\frac{e}{\alpha e^{\alpha} - e}$ *if the external write requests are uniform at random.*

Proof. We first calculate the probability that a page in a block that is currently being written into is still valid when the algorithm has cycled through the flash memory once and then use this to calculate the expected write overhead.

Consider a fixed physical page. We use the same argumentation as in the proof of Theorem 2.3 to show that there are at least $T-V$ external write requests until the cycling algorithm writes again in this block. The probability that the logical address associated with this physical page is invalidated by one external write request is $\frac{1}{V}$. Thus, the probability that it is not invalidated and still valid after $T - V$ external write requests is at most $\left(1 - \frac{1}{V}\right)^{T-V} = \left(1 - \frac{1}{V}\right)^{V(\alpha-1)} \approx e^{-(\alpha-1)}$. We conclude that the expected number of internal write requests is at most $e^{-(\alpha-1)} \cdot V$ while writing V pages. Note that it suffices to consider V pages

because we only need to look at the V logical addresses that are valid. Thus, we obtain $\omega \leq \frac{e^{-(\alpha-1)\cdot V}}{T-e^{-(\alpha-1)\cdot V}} = \frac{e}{\alpha e^{\alpha}-e}$. □

Note that $\sigma\omega \leq \sigma\frac{e}{\alpha e^{\alpha}-e} = \sigma\frac{e}{e(\alpha e^{\alpha-1}-1)} = \frac{\sigma}{(\sigma+1)e^{\sigma}-1} = \frac{\sigma}{\sigma e^{\sigma}+e^{\sigma}-1} \leq \frac{1}{e^{\sigma}} < 1$.

In order to make our access pattern more realistic, we divide the data into two types: static and dynamic. Static data is data that is never invalidated whereas dynamic data is data that is invalidated uniformly at random. Let μ, δ, and φ be the fraction of static data, dynamic data, and free space, respectively. Thus, we have $\mu + \delta + \varphi = 1$.

Theorem 2.9. *If the external write requests are uniform at random, then the cycling algorithm has write overhead* $\omega \leq \frac{\mu\cdot T+\delta\cdot T\cdot e^{\frac{1-\alpha}{\alpha\delta}}}{T-(\mu\cdot T+\delta\cdot T\cdot e^{\frac{1-\alpha}{\alpha\delta}})}$.

Proof. It is easy to see that while cycling from "left" to "right", i.e., from block b_0 to block b_{n-1}, every page containing static data is still valid and thus causes an internal write request to avoid data loss. The dynamic data can be analyzed as before. Consider a fixed valid physical page containing dynamic data. The probability that one external write request invalidates this page is $\frac{1}{\delta\cdot T}$. While cycling from left to right, we can lower bound the number of external write requests by $T - V$. Thus, the probability that this page is not invalidated while cycling once from left to right is at most $(1 - \frac{1}{\delta\cdot T})^{T-V} \approx e^{\frac{1-\alpha}{\alpha\delta}}$. Hence, the number of internal write requests from dynamic data is $\delta \cdot T \cdot e^{\frac{1-\alpha}{\alpha\delta}}$ while cycling through the flash memory once. This leads to an expected write overhead of at most $\frac{\mu\cdot T+\delta\cdot T\cdot e^{\frac{1-\alpha}{\alpha\delta}}}{T-(\mu\cdot T+\delta\cdot T\cdot e^{\frac{1-\alpha}{\alpha\delta}})}$ □

2.7 Conclusion

We have analyzed the connection between write overhead ω and space overhead σ in flash drives. We have shown that they are fundamentally connected and that $\sigma\omega \geq 1$ must always hold if the data is being updated adversarially. This bound is tight as there exists a simple algorithm that achieves $\sigma\omega \leq 1$ independent of how the data is being updated. Thus, if we have to assume that data is being accessed adversarially and we thus face a difficult problem, our simple algorithm is optimal.

3

Finding Friends

3.1 Introduction

We now look at a game that does not seem to involve adversaries: how friendships evolve in social networks. Our focus is on initial friendship graphs that are created by an adversary.

Psychologists claim that you have a limit of how many friends you can handle [47]. Consequently, you should assess your current friends, and drop those that are unsatisfactory, to make room for new ones! We study the computational side of finding friends in social networks. We assume that people can only choose new friends among their current social environment, i.e., one can only become friends with friends of friends, or more generally with acquaintances in the ℓ-hop neighborhood of the current friendship graph. If people constantly improve on their friendships with this *local* strategy, will this eventually lead to a social optimum, or at least an approximate solution? What is the best strategy to find new friends? Should one just greedily pick the best available friends? Or should one rather try to be friends with a diverse set of people, in order to profit from a larger set of possible new friends?

Not so surprisingly, we show that any local friend-finding strategy will only converge to a solution that is arbitrarily worse than a global optimum. More surprisingly however, there is no best local strategy. No matter what the strategy is, there is always a possible input scenario where other local strategies are much better. We also study mixing strategies, i.e., we allow everyone to use several strategies to find their friends. Additionally, we investigate slightly changed valuation models. We show that judging a friend not on his own, but also by his friends, can lead to unstable states, i.e., nodes switch friends indefinitely. We also analyze a valuation model in which breaking up a friendship reduces the valuation of the friendship permanently.

3.1.1 Related Work

An early ancestor of our work is the stable marriage problem, introduced by Gale and Shapley [40] in 1962: We are given n nodes, partitioned into two sets commonly denoted as men and women. Each woman has a strictly ordered preference list over all men and vice versa. They now want to create a stable matching. A matching is called stable if there is no pair of man and woman such that, instead of being matched to their current partner, they would prefer to be matched to each other. The roommate problem [40] is another related research area, where the nodes are not partitioned into two disjoint sets. Each node again has a complete, strictly ordered preference list. In this basic setting there might not be a stable matching. The problem is further investigated in [1, 45], stating restrictions to allow and find stable matchings. An overview on stable marriage can be found in e.g., [63] and a more detailed analysis of matchings in bipartite graphs in [108]. Stable marriage has also been studied as an online problem where the preferences of the men are revealed one at a time [79]. In this setting there are $\Omega(n^2)$ initially unstable marriages in the worst case.

Much research has been done in the stable marriage area on preference lists with ties [48, 77], i.e., when the constraint of strictly ordered preference lists is lifted. In our model we assume locality of information, i.e., nodes do not know their complete preference list. Furthermore, we do not require the nodes to have a strict ordering. Finding a maximum matching for stable marriage with these extensions, ties and incomplete preference lists, is known to be $\mathcal{NP}$-hard [74,91]. It can be approximated within a factor of 2 [91]. It can also be approximated within a factor which depends on the number of ties in the preference lists [66].

There have been several approaches to solve stable marriage in a distributed way. In [49] the nodes can only try to be matched to a fixed set of adjacent nodes.

Generally related to our work are network formation games from the field of economics. The nodes create links, as a one shot game or dynamically, to generate welfare which depends on the created links. This welfare is allocated to the players according to some specific rules. These games include models of so called market sharing agreements, in which companies can agree not to sell goods on each others markets to increase their profits [15], and labor markets, where workers get jobs offered and pass the offer to one of their friends if they are already employed [26]. Another example is the model of a general buyer and seller market in which a link represents a potential transaction [83]. A survey on this area is in [75].

So far, the possible matching edges were a fixed set of edges. In [6], the nodes are partitioned into two sets, workers and firms. There are static connections between some workers which indicate friendship. The workers are matched using a local variant of the Gale-Shapley algorithm, but only to firms which are known to their friends. This introduces a dynamic set of matching partners. If a worker changes his company, this can change the set of possible matching partners for his friends. The model used by Martin Hoefer generalizes this [71]. The set of nodes is not partitioned and nodes can possibly have more than one matching partner. In his paper, Hoefer studies the convergence time of matching edges

in a social network, with a limited lookahead ℓ. For $\ell = 2$ this means that the nodes only know the neighbors of their neighbors. In general, nodes can only create a connection to nodes which are in a distance of at most ℓ hops. Hoefer's model distinguishes between social links and matching links. Social links are static edges, which already exist in the initial graph, and keep existing throughout the execution of the algorithm. Matching links on the other hand are created and possibly removed by the algorithm. We drop this difference, and only use one kind of (dynamic) edges. If needed, we can easily emulate social (static) links by adding edges with maximal quality, which will not be removed at Whereas Hoefer's focus was primarily on runtime, we primarily investigate the achieved welfare. Since our model is used to describe cooperation between different players or actual friendship, we also assume that both partners value a potential relationship identically.

Later, it was shown that deciding reachability, i.e., whether a certain fixed matching can be reached starting from a fixed initial friendship graph, is $\mathcal{NP}$-hard [72].

3.1.2 Our Contributions

We describe our model and our assumptions in Section 3.2. We first describe local algorithms that try to maximize the welfare of the participants, i.e., try to find good friends. This is done by selfishly maximizing the sum of the qualities of incident edges. We prove in Section 3.3 that there cannot be an optimal, local algorithm. We do this in two steps. First, in Section 3.3.1, we show that no local algorithm can compete with a global, optimal algorithm, i.e., any local algorithm will be arbitrarily worse in certain scenarios. Afterwards, in Section 3.3.2, in the spirit of [4], we compare local algorithms with other local algorithms. We prove that there is no best local algorithm, i.e., one which is always at least as good as every other local algorithm. For every local algorithm there exists an initial friendship graph where it is arbitrarily worse than another local algorithm. This includes randomized algorithms. In Section 3.3.3 we allow the nodes to execute several algorithms in parallel by allowing them to use memory. Although every local algorithm can be arbitrarily worse than a global optimum, we show that the nodes can achieve a factor 2 approximation in comparison to the best of the executed algorithms.

In Section 3.3.4 we study a slightly modified model, where friends of friends also matter how we value a friend. We show that there exist initial friendship graphs in which a simple algorithm no longer achieves a stable state. In Section 3.3.5 we assume that ending a friendship permanently damages the quality of a friendship. If we assume that a breakup reduces the quality of the friendship by a constant value, then the runtime of any algorithm is $\mathcal{O}(n^2)$. If we assume that a breakup reduces the quality of the friendship by a constant factor, then there are initial friendship graphs where the stable state that can be reach without using memory is a factor of $\Theta(n)$ worse than the stable state that can be reached when using memory.

3.2 Model

We model a social network with a set V of nodes (human beings), $n = |V|$. There exists a quality function $q : V \times V \to [0, 1]$ that represents the quality of a friendship for any two nodes $u, v \in V$. A higher value means a better friendship. We do not consider negative edge qualities since no node has an incentive to create an edge, which reduces its welfare. The quality is symmetric, i.e., $q(u, v) = q(v, u)$. Without symmetry we can create a similar cycle as in the roommate problem [40] and thus not reach a stable state. Initially, the nodes are connected by an arbitrary graph $G = (V, E)$, representing the initial knowledge graph. In other words, if two nodes are neighbors in G, they are initially friends. Nodes might decide to create new friendships (edges), and friendships can also be ended. The set of edges E is as such highly dynamic.

A node's welfare (happiness) depends on the quality of its friendships. Formally, the welfare of a node v is defined as $\sum_{u \in N(v)} q(u, v)$, where $N(v)$ denotes the set of neighbors (current friends) of v. Nodes try to maximize their personal welfare by finding new friends with high quality values.

In reality, one cannot be friends with everybody. However, since our edge qualities are non-negative, nodes could just accumulate more and more friends, until G is a clique. We do not want this effect in our study; as such each node v has a maximum number of possible friends k_v, an individual constant parameter. If a node v already has k_v friends, and fancies a new friend, it must first drop an old friend.

Nodes cannot choose their friends arbitrarily. Instead, they can only choose friends that are already within their visibility. More formally, we define the constant parameter ℓ which we call lookahead. A node can only initiate a friendship to nodes within ℓ hops of graph G. For example, if $\ell = 2$, apart from its friends (neighbors in G) a node can only see its 2-hop neighbors (friends of friends); new friends can only be found among these 2-hop neighbors. As such we deal with so-called ℓ-local algorithms. A node has all the information in its ℓ-hop neighborhood. In particular it knows about all the friendships and all the qualities in the ℓ-hop neighborhood.

Nodes run ℓ-local algorithm in order to optimize their friendship graph. Since friendships are a serious business with lots of potential for conflicts of interest, one needs to be careful about the issue which node can propose friendships to which other node at what time. There are various meaningful models here. Indeed, our proofs are relatively robust and work in different kinds of algorithmic models.

We suggest the *round-robin* model. In this model, all nodes take turns in a round-robin fashion. Whenever it is the turn of a node v, v can propose friendship to different nodes in its ℓ-hop neighborhood. The order in which it asks these nodes is completely up to v; this order is basically the friend-finding algorithm. If v has already k_v friends, it only proposes to nodes whose friendship is more valuable (edge quality is higher) than that of v's worst friend, i.e., to better friends. A node u that is asked by v evaluates the proposed friendship. If u still has room for a new friend, or if v's proposed friendship is better than

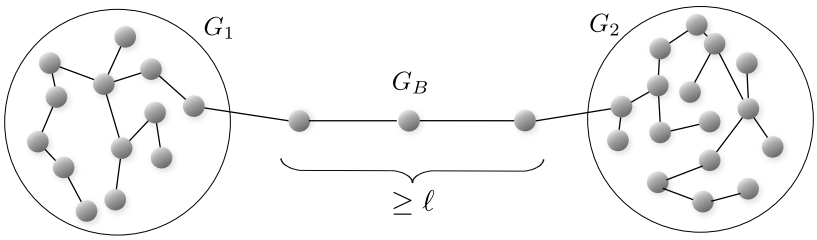


Figure 3.1: Two subgraphs G_1, G_2 which are never in contact with each other because they are separated by a bridge G_B with a diameter of at least ℓ.

the worst of u's current friendships, u will accept the edge (and drop its worst friendship if necessary). In other words, a new edge is only added to the graph if both nodes u, v adjacent to edge (u, v) want the edge. If a node gets rejected from a potential new friend, it continues to ask other candidates according to the ordering.

If a node does not find any better friend, the round-robin model asks the next node to find a friend. The procedure ends if no new friendships can be discovered, i.e., if a whole round-robin loop does not change the friendship graph anymore. We call this a stable state.

Note that the initial friendship graph G constrains the final outcome. If two nodes are in different connected components of the initial graph G, then they can never become friends as they cannot learn about each other. Also, connected components may partition into smaller connected components during the execution of the algorithm. For the sake of simplicity, we assume that the initial friendship graph G is connected; however, alternatively, just think of our analysis to take place in one of the original connected components.

We can imagine various ways to increase the power of the nodes. In particular, nodes might have additional, constant size memory. Memory allows nodes to remember special former partners, e.g., the best ones they dropped, nodes for which the creation of the corresponding edge had some specific properties, or any other mechanism imaginable. Nodes stored in the memory can be added to the list of candidates which will be asked by the algorithm. An algorithm might try to combine several algorithms into one by executing them in parallel. This can be done by performing one round of each algorithm alternately and using the memory to remember the states of the other algorithms. Note that due to the constant memory, only a constant number of algorithms can be combined in this way.

3.3 On Welfare

In this section we compare different algorithms. As a measurement we use the welfare in the stable states. We compare the globally achieved welfares, i.e., the sum of welfare achieved by all nodes. Algorithm A is said to be arbitrarily worse than algorithm B if the welfare in the stable state of A is $\mathcal{O}(n \cdot \varepsilon)$ whereas it is $\Omega(n)$ in the stable state of algorithm B. Note that ε can be arbitrarily small, e.g., as small as any function in n such as $\varepsilon = 2^{-n}$.

In the model section we have described algorithms, which only choose beneficial partners. Let us justify why we focus on this class of algorithms. We show that temporarily accepting up to c worse friends results in a constant increase in the lookahead from ℓ to ℓ^c.

Lemma 3.2. *If all nodes are allowed to temporarily choose c worse neighbors, the length of shortest path between two nodes u, v can only decrease by at most a constant factor.*

Proof. Let u, v be two nodes which are not neighbors. If the graph is connected, there exists a path of length k from u to v via nodes $u_1, \ldots, u_k$. If each node is only allowed to select one worse partner, the distance is minimized if every ℓ-th node connects to a node in distance ℓ bypassing the $\ell - 1$ now unnecessary nodes in between. The path now consists of the nodes $u, u_\ell, u_{2\ell}, \ldots, v$. Hence, the distance can be reduced by at most a factor of ℓ in total, i.e., to at least $\frac{k}{\ell}$ which is a constant factor reduction since ℓ is independent of n. Thus, if every node chooses a worse friend at most c times, then the distance decreases by a factor of at most ℓ^c, which is only a constant. □

Hence, if nodes are allowed to temporarily choose worse partners, all the proofs still hold by increasing the distances from ℓ to ℓ^c. Thus, we will not treat this separately but mention it briefly in the proofs.

3.3.1 Local vs Global Algorithms

Let us now analyze how local algorithms perform when compared against a global optimum, i.e., a graph which maximizes the sum of welfare achieved by all nodes.

Theorem 3.3. *For nodes with a constant lookahead ℓ and a constant size memory there exist initial friendship graphs for every local algorithm such that its reached welfare is arbitrarily worse than a global optimum.*

Proof. Consider an initial friendship graphs as depicted in Figure 3.1. The graph consists of three subgraphs G_1, G_2 and G_B. The two larger subgraphs G_1, G_2 are connected through a bridge G_B, which has a diameter of at least ℓ and each node v in the bridge has already k_v friends. The valuations are such that for every pair $(u, v) \in G_1 \times G_2$ the quality is 1, for every pair $(u, v) \in G_i \times G_i$ with $i \in \{1, 2\}$, we have $q(u, v) = \varepsilon$. Furthermore, for every $(u, v) \in G_i \times G_B$ with $i \in \{1, 2\}$, $q(u, v) = \varepsilon/2$, except for the nodes in G_i that are connected to nodes in G_B. These nodes value each other with ε as well. Every node $v \in G_B$ values every other node $u \in G_B$ with 2ε if the edge exists in the initial friendship graph and with 0 otherwise.

Hence, the nodes in the bridge already have their best possible friends and therefore will not change their friends. These valuations represent an initial friendship graph in which no node really likes his friends, but does not know any better candidates.

This initial friendship graph is a stable state, hence no local algorithm will create an edge between any node from G_1 with any node from G_2 because of

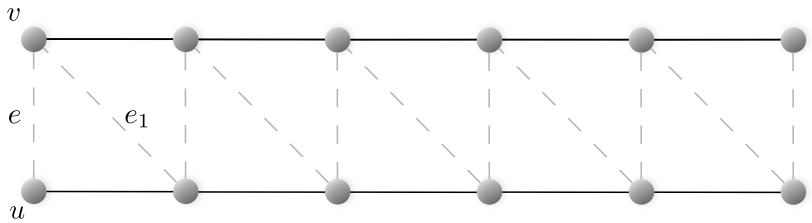


Figure 3.4: A track going from left to right. The dashed, gray edges are created by the execution of a local algorithm, the black edges are given in the initial friendship graph.

the distance between those subgraphs. This holds due to Lemma 3.2 even if the nodes are allowed to choose non-beneficial partners. Furthermore, the nodes in G_B are not appealing for any node and thus remain isolated. Therefore, the best achievable stable state is $\mathcal{O}(n \cdot \varepsilon)$. In the optimal solution the nodes from the sets G_1, G_2 are connected to each other to achieve a stable state with value $\Theta(n - |G_B|) = \Theta(n)$. □

Note that this result relies on the fact that any reasonable, local algorithm is only willing to create connections to beneficial partners or is only willing to accept a worse partner a constant number of times. Let us further remark that the initial friendship graph and any stable state reached by a local algorithm is not necessarily Pareto efficient. This means that there are initial friendship graphs where we can easily improve the welfare of some nodes without lowering the welfare of other nodes, e.g., by moving one node v from G_1 to G_2. Now v can make better friends. This increases v's welfare (and the welfare of the nodes which are now friends with v) but does not lower the welfare of any node.

3.3.2 Local vs Local Algorithms

We now show that there cannot be a best local algorithm, i.e., an algorithm, which can achieve a stable state whose welfare is at least as good as the welfare of any other local algorithm. We prove that for every local algorithm there exist initial friendship graphs in which it is arbitrarily worse than another, local algorithm. But first, we need a few more terms.

Definition 3.5. *A* track *of length j consists of two disjoint sets, each with j nodes. The nodes of each set are initially arranged in a line as shown in Figure 3.4 (connected to each other with the edges colored in black). The dashed, gray edges have a strictly monotonic increasing quality from left to right. The black edges from the initial friendship graph have a quality of $\mathcal{O}(\varepsilon)$. The remaining edges have a quality of 0 and are therefore never used. Every node v of a track has $k_v \geq 4$. After the* initial edge *e is created, any algorithm in our model will create the dashed, gray edges, starting with e_1, one by one from left to right, i.e., in increasing order regarding their quality. The creation of one dashed, gray edge allows the creation of the next dashed, gray edge since those nodes are now within ℓ hops of each other. We call the creation of the edges* exploration of a track.

Definition 3.6. *A* successful track *generates a welfare of* $\Omega(n)$ *if the initial edge* e *is created. In such a track we can set the quality of the edges in the initial friendship graph to* $\mathcal{O}(\varepsilon)$ *and the edge quality of the selected edges connecting those two sets, i.e., the dashed, gray edges in Figure 3.4 to a constant. A successful track must have length* $\Omega(n)$*. Upon creating the initial edge* e *on the left, the track can be explored. After every dashed, gray edge is created, the welfare is* $\Omega(n)$*; without the initial edge the welfare is generated only by the edges of the initial friendship graph and thus* $\mathcal{O}(n \cdot \varepsilon)$*.*

Definition 3.7. *A track* T *is said to be* blocked *if, during the exploration, no further edge is created because at least* ℓ *consecutive nodes have already friends that are better than those of the track* T*. This stops the exploration since the nodes have no incentive to continue to explore the track.*

Similarly, a track T_2 blocks *another track* T_1 *if nodes of* T_2 *are part of the reason why* T_1 *is blocked. An example of this can be seen in Figure 3.8.*

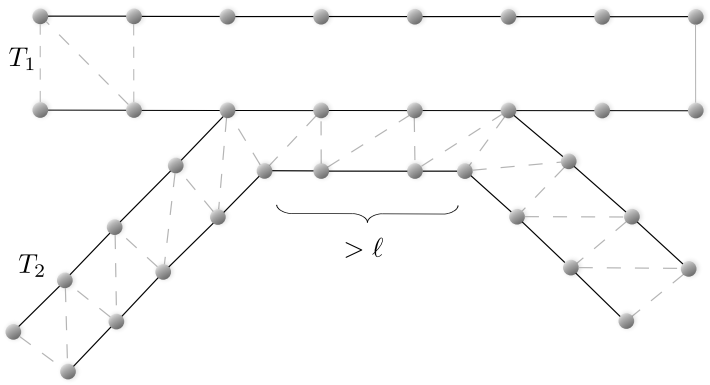


Figure 3.8: Two tracks T_1 and T_2 interacting. T_1 is blocked by T_2 since the shared nodes of both tracks have no incentive to create the edges of T_1 since they are content with the edges of T_2.

Theorem 3.9. *For nodes with a constant lookahead* ℓ *and a constant size memory there exist initial friendship graphs for every deterministic, local algorithm such that its reached welfare is arbitrarily worse than that of another deterministic, local algorithm.*

Proof. Consider two concatenated tracks T_1, T_2 each of length at least ℓ. On top of each of the ℓ nodes of the first track T_1 are four nodes in a line, i.e., each node u_i is connected to a different intermediate node v_i' which is connected to node v_i. This subgraph of the initial friendship graph is depicted in Figure 3.10. We define $k_{v_i} = 2$ and $k_{u_i} = 4$ for all nodes u_i, v_i with $i \in \{1, \ldots, \ell\}$, i.e., every node u_i can create a connection to exactly one more node whereas v_i must sever an edge to create a new edge. Let v_i' be v_i's worst friend. The qualities are such that $q(u_i, v_i) > q(u_i, x_i)$ holds. Let the edges from the initial friendship graph have a quality larger than $q(u_i, v_i)$ for all $i \in \{1, \ldots, \ell\}$ to guarantee that they will not be severed.

Node v_i has two options. It can either create a connection to node u_i or to node w_i; all other nodes are valued with 0. If every node v_i with $i \in \{1, \ldots, \ell\}$ decides to create a connection with u_i, then node u_i will not create the edge (u_i, x_i) because of $q(u_i, v_i) > q(u_i, x_i)$. Thus, track T_1 is blocked. Consequently, track T_2 will not be explored. On the other hand, if every node v_i with $i \in \{1, \ldots, \ell\}$ decides to create a connection with w_i, then track T_2 will be explored.

Note that it is unimportant if the nodes v_i may have the option to temporarily revise their decision by using their constant memory. It only matters whether the track T_2 is explored at some point execution of the algorithm.

Track T_2 is either explored or not explored – depending on the joint local decision of node v_i with $i \in \{1, \ldots, \ell\}$. If track T_2 is not explored, it is a successful track, i.e., its exploration generates $\Omega(n)$ welfare. If track T_2 is explored, then it blocks a successful track T_3. This subgraph is shown in Figure 3.8. Since these scenarios are indistinguishable for any algorithm, the remainder of the graph can be such that its choice is wrong. It is easy to see that there is another local algorithm which decides correctly for this particular scenario. Limiting the quality of the edges in the subgraph to $\mathcal{O}(\varepsilon)$ yields the theorem. □

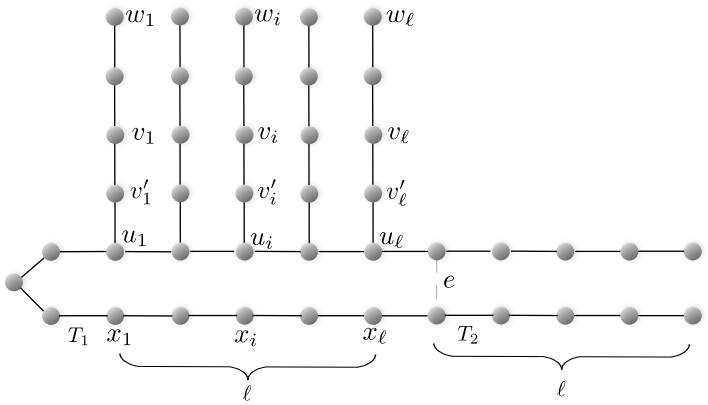


Figure 3.10: A subgraph of size $\mathcal{O}(1)$ with outgoing track. Node v_i must now decide if it wants to create a connection to node w_i or to node u_i. If all the connections to u_i are made, the track cannot be explored. Note that edge e is not part of the initial friendship graph.

We can prove a similar result for algorithms, which try to execute a constant number of different algorithms in parallel to avoid the aforementioned problem. This allows them to emulate algorithms where one might explore a track whereas another might not.

Theorem 3.11. *For nodes with a constant lookahead ℓ and a constant size memory there exist initial friendship graphs for every local algorithm, which executes several algorithms in parallel, such that its reached welfare is arbitrarily worse than that of another local algorithm.*

Proof. Imagine a subgraph with two for the nodes indistinguishable outgoing tracks T_1, T_2 of length $\Omega(\ell)$. These tracks are constructed as in the proof of Theorem 3.9. Any algorithm has four options available. Either it explores no

track, T_1, T_2 or both. W.lo.g. half of the nodes explore either T_1 (and possibly T_2 as well) or explore no track at all. The graph is such that T_1 blocks a successful track T_3. Furthermore, exploring T_2 is necessary in order to eventually start the exploration of T_3. Thus, an optimal algorithm explores only track T_2. Note that algorithms which explore both tracks will not explore T_3. Hence, half of the executed algorithms have chosen in a way such that they cannot be optimal anymore.

Track T_2 leads to another subgraph where the process is repeated. This subgraph has also two, for the algorithm indistinguishable, outgoing tracks. One of the tracks blocks the successful track T_3 and the other one leads to the next subgraph. Since we can create the remainder of the graph always such that at least half of the remaining algorithms "fail", this process only needs to be repeated a constant number of times. In the last subgraph one of the tracks is the successful track T_3 and the other track blocks track T_3.

By setting the edge qualities in the subgraphs to $\mathcal{O}(\varepsilon)$, we ensure that the local algorithm only chooses edges with quality $\mathcal{O}(\varepsilon)$ and thus has a welfare of $\mathcal{O}(n \cdot \varepsilon)$. This yields the claim. □

Theorem 3.12. *For nodes with a constant lookahead ℓ and a constant size memory there exist scenarios for every randomized, local algorithm such that its reached welfare is arbitrarily worse than that of another deterministic, local algorithm.*

Proof. We use a graph similar to the one in the proof of Theorem 3.9. Any randomized local algorithm either chooses to explore the track with probability at least $\frac{1}{2}$ or chooses not to explore the track with probability at least $\frac{1}{2}$. Hence, we can first concatenate b of these structures with $b = \Theta(\log n)$ such that the probability that the randomized algorithm chooses correctly is at most 2^{-b}, i.e., with probability $n^{-\alpha}$ for some constant $\alpha > 0$. There exists a local algorithm that always chooses deterministically and correctly that achieves an optimal solution. By choosing the edge values accordingly, we can ensure that the achieved stable states differ sufficiently. □

3.3.3 Executing Local Algorithms in Parallel

We have seen that the welfare in the stable states that different algorithms reach differs significantly. As seen in Theorem 3.11, there is no optimal algorithm.

Nevertheless, if we execute several algorithms in parallel, we want to be able to get the best outcome of any of these algorithms. Due to the fact that the nodes only have a local view, they cannot know which of their friendships are part of the best achieved solution. With their limited knowledge, we let the nodes pick greedily. This does not yield the best solution as shown in the example depicted in Figure 3.14. Every node in this graph can only have one friend. The optimal solution is to pick the edges (u, v) and (x, w), but the greedy approach picks (v, w) and (u, x). We show that greedily picking yields a factor 2 approximation compared to the welfare achieved by any of the executed algorithms. To be able to pick edges greedily, we need to prove an upper bound on the runtime, which allows the nodes to know when any algorithm has terminated.

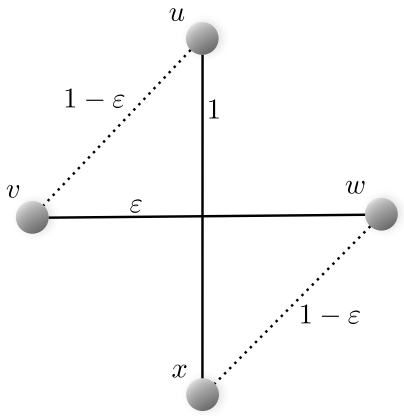


Figure 3.14: The best solution consists of the edges (u, v) and (w, x) whereas our greedily picked solution consists of (u, x) and (v, w) and is thus a factor of 2 worse than the best solution.

Lemma 3.13. *The runtime of any local algorithm that only chooses higher quality edges is* 2^{n^2}.

Proof. We use a potential function to prove this. Consider a bit string of length n^2. The i-th bit represents the edge with the i-th largest quality. There is a 1 at position i if and only if the edge with the i largest quality exists in the graph. The bit string can be regarded as a counter. Since we only allow beneficial changes, this potential function increases with every change. This limits the total runtime of any algorithm of this type to 2^{n^2}. □

Note that the nodes cannot know when exactly the execution has terminated because of their limited view, but only know the rather weak upper bound of 2^{n^2}. Hence, greedily selecting the edges will start after 2^{n^2} rounds. Since at least one edge is picked every round (the edge with the highest remaining quality), this allows the nodes to output a valid solution after $\mathcal{O}\left(2^{n^2} + n^2\right)$ rounds.

Theorem 3.15. *By running several algorithms in parallel and greedily selecting the best edges at the end, we obtain a factor 2 approximation compared to the best of the executed algorithms.*

Proof. This proof is similar to the proof that any maximal matching is a factor 2 approximation of a maximum matching. Consider the union of edges of all solutions. If two nodes mutually agree to pick an edge, then this edge can either be part of the best solution in which case the choice is good. Otherwise, choosing this edge prevents the nodes from picking at most two edges from the best solution. But both of these edges must have a lower quality than the chosen edge. Hence, our solution is at least half as good the best solution in the union of all solutions and therefore must be at least half as good as the best solution. Continuing this inductively yields the claim. □

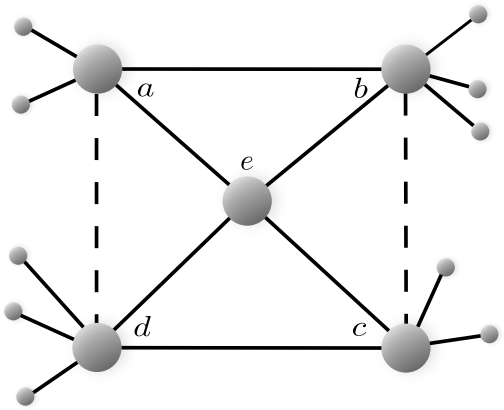


Figure 3.16: The edges $(a, b), (c, d)$ exist. In this setting b prefers to be paired up with c and c would be happier with that matching. Afterwards a would match with c and thus b with d. In the next round, the cycle would start anew.

3.3.4 Valuing Friends of my Friends

In a real social network it might not be sufficient to evaluate a friend on her own. In order to evaluate a friendship, it is necessary to consider the friends of a friend. Thus, we introduce this friendship valuation variant. An edge continues to represent an existing friendship, but the new edge quality is a weighted, combined value of the node and its neighbors. Formally, this can be expressed as $Q(u, v) := q(u, v) + \alpha \sum_{x \in N(v) \setminus \{u\}} q(u, x)$ where q denotes the quality function as defined before and α is some constant. In this model every edge quality $Q(\cdot, \cdot)$ is directed, i.e., $Q(u, v) \neq Q(v, u)$ is possible.

In this model, there may not be a stable state. This is due to the asymmetric valuations of the nodes, which can be used to create valuations similar as in the roommate problem in [40].

Theorem 3.17. *If we include the neighbors of a node in the valuation function, there are initial friendship graphs in which a local algorithm does not reach a stable state.*

Proof. It is sufficient to consider the nodes a, b, c, d, e and their friends, which we denote with $V_a, V_b, V_c.$ and V_d, respectively. This graph is depicted in Figure 3.16. The valuations of node e are such that it neither has it any incentive to choose another friend nor do any of his friends want to sever the edge with e. The other edge qualities are as follows: $q(a, b) = q(a, c) = q(b, c) > q(a, d) = q(b, d) = q(c, d)$. This means the three nodes a, b, c prefer each other over node d. Now we can set the edge qualities of the friends of each node such that the final edge qualities are $Q((a, b)) > Q((a, c)) > Q((a, d))$. Node a prefers the friends of b over those of node c and over those of node d. Furthermore, we require $Q(b, c) > Q(b, a) > Q(b, d)$ and $Q(c, a) > Q(c, b) > Q(c, d)$. A similar construction as above yields these inequalities. The evaluations of node d do not matter.

A brief technical remark has to be made. The nodes in V_u with $u \in \{a, b, c, d\}$ have to be content with being friends with u. Furthermore, the nodes in V_u with

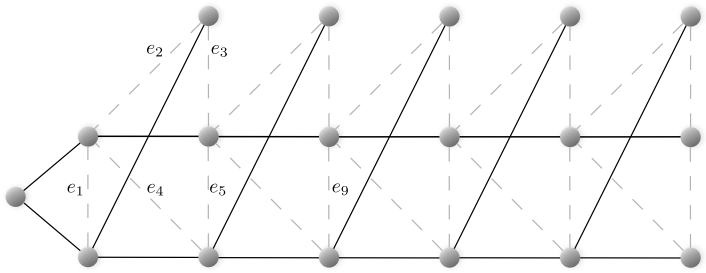


Figure 3.19: A counter as presented by Hoefer [71]. Edges which are created from the algorithm in dashed, gray, the others in black.

$u \in \{a, b, c, d\}$ have no incentive to initiate a connection with any other node than u.

This enables us to create a cycle.

If node a is friends with node b and node c is friends with node d, then both node b and node c prefer each other over their current matching partner.

If node a is friends with node c and node b is friends with node d, then both node a and node b prefer each other over their current matching partner.

If node a is friends with node d and node b is friends with node c, then both node a and node c prefer each other over their current matching partner.

Thus, no matter which node is matched with node d, it wants to change to another node which is willing to do so. Hence, no stable state can be reached. □

Clearly, neither a statement about the convergence time nor about the globally achieved welfare is possible in this setting.

3.3.5 Breaking Up a Friendship is Expensive

In a real social network, breaking up friendships is seldom without consequences to that friendship. To model this, we assume that the edge quality decreases every time the corresponding edge is removed. We consider two ways to reduce the quality of an edge: Either reduce it by a constant term or by a constant factor.

Let us first analyze the effect of this to the runtime of any local algorithm. In [71] it has been proven that a simple greedy algorithm which uses memory has a runtime of $\Omega\left(2^{\Theta(n)}\right)$. We obtain a dramatically smaller runtime if the quality of the edge gets reduced by a constant.

Theorem 3.18. *If the edge quality $q(e)$ gets reduced by a constant term every time the edge e is removed, the runtime of any local algorithm is $\mathcal{O}\left(n^2\right)$.*

Proof. Every edge can only be created and severed $\mathcal{O}(1)$ times before the nodes have no incentive to create that edge because its quality is 0. Hence, after $\mathcal{O}\left(n^2\right)$ any algorithm must terminate. □

Before we describe the effects of reducing the edge qualities by a constant factor, let us describe an initial friendship graph. It is the same used in [71] to

prove the lower bound for the simple algorithm. We will show that the result still holds if the edge quality is decreased and consider the consequences for the welfare.

Imagine the nodes arranged as show in Figure 3.19. The edge qualities are such that $q(e_1) < q(e_4) < q(e_3) < q(e_2) < q(e_5)$ holds. The graph is constructed by concatenating these blocks. We assume that the edge qualities are $\mathcal{O}(1)$ in the beginning. For every node v its value k_v is such that it can create one additional friendship. For simplicity's sake, we now assume that the lookahead ℓ is limited to 2. We analyze a greedy local algorithm which always chooses the best available option.

Let us describe what happens in this friendship graph. At the beginning edge e_1 can be created which enables the partnership e_2. While creating e_3, this edge will also be severed. Thus, the creation of edge e_1 is again possible. Since the edge e_2 cannot be created, the simple algorithm creates e_4, thereby severing e_1 the second time. Finally, the partnership e_5 is formed. Note that edge e_1 was created twice in order to create e_5.

Lemma 3.20. *The runtime of the greedy algorithm that does not use memory is* $\Omega\left(2^{\Theta(n)}\right)$.

Proof. The inequalities stated above still hold even if we decrease the quality by a constant factor every time we remove the corresponding edge. We can concatenate b of those structures with $b = \Theta(n)$. Since e_1 must be created twice in order to create e_5 once and e_5 in turn must be created twice in order to enable the partnership e_9, the concatenation of these blocks requires e_1 to be created $\Omega(2^b)$ times and thus the lemma follows. □

Theorem 3.21. *If the edge quality $q(e)$ gets reduced by a constant factor every time the edge e is removed, there is an initial friendship graph such that the combined quality of the edges created by the greedy algorithm can be upper bounded by a constant. But in the same friendship graph, using memory enables an algorithm to create edges with a combined quality of $\Theta(n)$.*

Proof. As explained above, every edge gets recreated often, i.e., in the i-th building block $b - i$ times. Let $1/x$ be the factor by which each edge quality gets reduced upon removing. The total welfare of the edges created in the i-th block is at most $c \cdot 2^{b-i}$ with c being some constant. Summing up over all blocks yields a total welfare of

$$c \cdot \sum_{i=1}^{b} 1/x^{b-i} = c \cdot \left(\frac{x}{x-1} - \frac{x^{1-b}}{x-1} \right) = \mathcal{O}(1).$$

We now consider an algorithm that remembers every friend it had (which is a constant in this scenario). This means that after the first creation of e_5, it is no longer necessary to sever and recreate e_1. Thus, every edge gets only removed and recreated a constant number of times. This allows the total welfare to be $\Theta(n)$. □

Note that we can adapt this proof for $\ell > 2$ by using a slightly different way to construct the graph. But the main idea remains unchanged: we concatenate blocks in which one edge needs to be created twice in order to create another edge.

3.4 Conclusion

We have analyzed local algorithms for finding better friends. Our model allowed a friendship if it is beneficial to both participants. We have proved that local algorithms can be arbitrarily worse than global algorithms if the initial friendship graph represents a worst case, i.e., is adversarial. We also showed that there exists no best local algorithm. For every local algorithm there exists an initial friendship graph such that it performs arbitrarily worse than another local algorithm. If the nodes have constant memory, they can execute several local algorithms at once. Even though this does not help in the worst case, the nodes can achieve a factor 2 approximation compared to the best executed algorithm by choosing their friends greedily after the execution of all algorithms.

Thus, we have shown that no matter how you choose your friends, be it greedily or in a way that maximizes the number of acquaintances, if the initial friendship graph is adversarial, it might be the worst strategy possible.

4

Hiding Adversarial Content

4.1 Introduction

In the previous chapter we tried to find good friends. In this chapter, instead of finding something good, we want to avoid seeing something bad.

Be it ads, spam or malicious code, unwanted content on the web can take many forms. While there exist great filters for ads, spam and malicious code, all these problems share the property that one filter mostly fits all users. For example, if one user considers content to be an ad, most other users will do so as well. The same holds for malware and spam. This means that filters can be constructed once and applied to all users which is how ad- or malware filters usually work. In addition to that, ad filters [105] as well as spam filters heavily rely on blacklisting of hosts that serve either content type.

We focus on filtering user specific content. For scalability reasons it is impossible to have a specialized filter for each user specific topic. Even worse, classical host blacklisting techniques do not apply in our case since undesired content may very well be served by the same host as the content the user wants to see. Think of a news site that reports on the latest sports results. Since you did not watch the event yet, you do not want to know the result but may still be interested in the latest political developments, which are shown on the same web page. The upside in our scenario is that we are not competing with companies selling ads or black-hats distributing malware.

We present a personalized filter that removes content from a web page based on user specified terms or topics. The running example in this chapter is a spoiler such as a plot development in a TV show or a sports result. Some communities have dealt with this by requiring spoiler tags but this requires manually tagging every post and is topic but not user specific. However, our filter is universally applicable to remove undesired content and we use the notion of a spoiler just

as an example.

HTML objects are by definition organized in a tree structure. The Document Object Model Tree (DOM Tree) maps an HTML document to a tree structure whose nodes can be addressed and manipulated easily. These nodes include text, links, images, and all other content displayed on a website. We replace nodes that contain undesired content with placeholder nodes that reveal the original content when clicked. This preserves the overall layout of a web page when removing undesired content.

We explore the trade-off between removing as many spoilers as possible while not removing too much unrelated content. Only hiding all user defined terms on a web site does not lead to the desired outcome since spoilers will be revealed by text or pictures nearby. For example, sentences like "[redacted] arrested for drunk driving" or "The [redacted] [redacted] won the Super Bowl 28–24" still reveal a lot of information even though the spoiler terms are hidden. Especially since the user herself defined which terms to filter. Blocking the whole web page greatly diminishes the user experience when applied to sites that serve content that is mostly unrelated to the users filter terms, yet contains a small section that reveals spoilers. Thus, neither of the two extremes – solely hiding all user defined terms or blocking the whole web page – are desirable. Unfortunately, the middle ground – removing as many bad words from the web page while removing as little good words as possible – turns out to be an $\mathcal{NP}$-hard problem.

Hence, we use a heuristic to exploit the locality of content in the DOM tree to filter entire paragraphs or sections of a website that may contain spoilers.

4.2 Related Work

There is a large number of different content filters. Designed to filter specific undesired content types, they span from lightweight browser based filters up to search-engine filters, which may offer safety filters to exclude inappropriate links from the search results. In between those two approaches, there are solutions such as network-based filtering and content-limited filters offered by Internet service providers [41].

Classic Content Filtering Problems. Advertisements and malware are content types that received a lot of attention. Both content types share the property that different people share a common view of what qualifies as malicious content or ads. This means that filtering such content comes down to the task of identifying content that should be filtered once. The obtained list of elements which should be hidden can then be shared with all users such that the filtering logic on the client side is simple and fast.

One example of such a filter is Prophiler [27]. The focus of the system is set on finding websites that serve malicious JavaScript code to its visitors. Malicious behavior is detected by executing JavaScript in a virtual environment which tracks behavior such as drive-by downloads. While such systems identify malicious scripts with high accuracy, it is computationally intensive to sift through large amounts of websites. To reduce the computational requirements, Prophiler

uses machine learning to quickly discard benign websites. The results can be used to efficiently blacklist websites that serve malicious code to its visitors.

ZOZZLE [39] is a browser plug-in that recognizes JavaScript malware through static code analysis, while a user is visiting websites. Our method follows ZOZZLE's idea of filtering content within the clients browser without using global blacklists. However, since malware is the same for most users, we do not see the advantage of such a solution when filtering malware instead of user specific content. The growing amount of junk email has led to the development of email spam filters [110]. The arms race between spammers and filter providers has fueled the development of a large number of different filtering mechanisms ranging from host blacklisting [46] to content analysis [100].

User Specific Content Filtering. Systems that allow users to filter content based on personal preferences or circumstances have also been proposed before. While malware- and Ad-filters are vastly popular, user specific filters, to our knowledge, are rarely applied. Since user specific topics should be filtered, building a global blacklist is usually not practical since each topic would require a custom blacklist. This is the main reason why such filters, in general, are implemented to filter content on the client machine. Existing filters do not apply to both: general websites and arbitrary topics.

Goldbeck et al. [58] filter tweets according to user defined TV shows or sporting events. However, their filtering mechanism is only applied to tweets. Using blacklist creation methods that are specific to the given scenario (TV shows or sporting events), the system hides undesired content with high success rates. However, to achieve this performance, many tweets that do not contain any spoilers were hidden. While our approach uses user defined keywords similar to the ones presented in this chapter, we do not limit the application scenario to twitter and topics can be chosen freely. Guo et al. [62] use Latent Dirichlet Allocation to hide spoilers in movie reviews found on IMDB. The effectiveness of the system is remarkable. However, its application range is strongly limited due to the focus on IMDB in both design and evaluation of the method. Boyd-Graber et al. [20] show that crowd-sourcing is a viable option to obtain annotated training data for their spoiler filter.

4.3 Model

We model a website as a rooted tree $G = (V, E)$ with n being the number of nodes $|V|$. Each node $v \in V$ contains $b(v)$ many keywords (bad words) and $g(v)$ many unrelated (good) words. Note that inner nodes of the tree can and do contain good and bad words. We can remove any node v from the graph, we denote this with *cutting*. Upon removing v, by extension, all its children are removed from the graph as well. We denote with $\beta(v)$ and $\gamma(v)$ the sum of the bad and good words of v and all of its children, respectively. Hence, $\beta(v) = b(v)$ for every leaf v of the tree. Let R denote the set of nodes that were removed. This set includes the children of explicitly removed nodes. The user has to specify a threshold t. Our goal is to remove as many bad words as

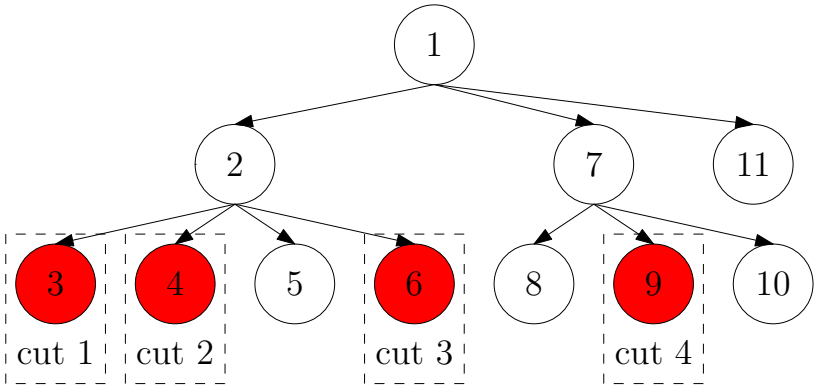


Figure 4.1: Placing the cuts only at the leaf nodes to filter the keywords

possible subject to $\frac{\sum_{v \in R} g(v)}{\sum_{v \in R} b(v)} \leq t$, i.e., without removing too many good words. We call this the *node removal* problem.

4.3.1 Example

We show a simplified example in Figure 4.1. The four nodes containing keywords are shown in red and the corresponding text of a node is replaced by the ID of that node.

In order to filter all nodes containing keywords at the leaf node level, we need to perform four cuts in the tree at nodes 3, 4, 6, and 9 (indicated by a box surrounding the corresponding node). If we use the minimal number of cuts, namely one cut at the root node 1, we remove the whole web page.

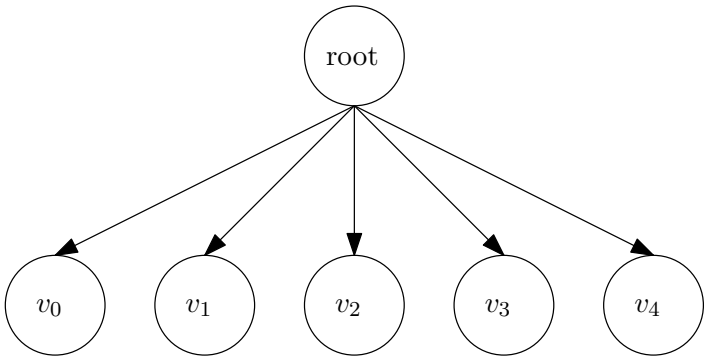
4.4 $\mathcal{NP}$-hardness

We now show that finding the best set of cuts for a given tree is $\mathcal{NP}$-hard. We show this by a reduction from the knapsack problem, which is well known to be $\mathcal{NP}$-hard [113].

Theorem 4.2. *The node removal problem is $\mathcal{NP}$-hard.*

Proof. We now describe the traditional knapsack problem before we present the main idea of the reduction. There are n items $z_1, \ldots, z_n$ and each item z_i has weight w_i and value y_i. The knapsack has a capacity of W. The task is to pack the knapsack and maximize the value of the set S of items that are in the knapsack that fit in it, i.e., to maximize $\sum_{z_i \in S} y_i$ subject to $\sum_{z_i \in S} w_i \leq W$.

We will mimic the knapsack problem in our graph. Each node v that we remove will add value (bad words), but use up space (removes too many good words compared to the number of bad words, i.e., $\frac{g(v)}{b(v)} > t$). The size of the knapsack is mimicked by a single node v_0 that has $b(v_0) = W$ and $g(v_0) = 0$.



nodes	v_0	v_1	v_2	v_3	v_4
$g(v_i)$	0	5	3	7	4
$b(v_i)$	5	3	1	4	2
node removal problem					

items		z_1	z_2	z_3	z_4
w_i		2	2	3	2
y_i		3	1	4	2
knapsack problem					

Figure 4.3: Instance of knapsack problem with $W = 5$ (and therefore $b(v_0) = 5$) and the transformed instance of the node removal problem. We set $t = 1$. Thus, the number of bad words $b(v_i)$ is the value of the item z_i, and the number of good words $g(v_i)$ is the sum of the value and the weight of the item z_i. The optimal solution S of the knapsack problem is z_2 and z_3 (total value is 5). The optimal solution R of the node removal problem is v_0, v_2, v_3. The number of bad words that are removed is $5 + 2 + 3$. The ratio is $\frac{\sum_{v_i \in R} g(v_i)}{\sum_{v_i \in R} b(v_i)} = \frac{0+3+7}{5+2+3} \leq 1 = t$.

Hence, maximizing the number of bad words that are removed while staying below the threshold is equivalent to maximizing the value of items in the knapsack while staying below the capacity of the knapsack.

Given an instance I of the knapsack problem with $I = (W, (z_1, \ldots, z_n))$ with each item $z_i = (w_i, y_i)$ and size of the knapsack W, we create a rooted tree $G = (V, E)$ with $n + 2$ nodes and set the threshold t to any positive number. The root node has $n + 1$ children, $v_0, \ldots, v_n$. Note that this tree has depth 1. Node v_0 has $b_0 = W$ and $g_0 = 0$. Every node v_i with $1 \leq i \leq n$ has $b(v_i) = y_i$ and $g(v_i) = w_i t + y_i t$. Hence, there is a bijection between the items $z_1, \ldots, z_n$ from the knapsack instance and the nodes $v_1, \ldots, v_n$ of our graph. A small example of this construction is depicted in Figure 4.3.

We now claim that there exists a set of items S such that $\sum_{z_i \in S} y_i \geq Y$ and $\sum_{z_i \in S} w_i \leq W$ if and only if there exists a set of nodes R in the node removal problem where we remove at least $W + Y$ many bad words from the tree while

preserving $\frac{\sum_{v \in R} g(v)}{\sum_{v \in R} b(v)} \leq t$.

Let S be such a set that has value $Y' \geq Y$. We claim that if we remove $R := S \cup \{v_0\}$ from the tree, then we have $\frac{\sum_{v \in R} g(v)}{\sum_{v \in R} b(v)} \leq t$ and we remove at least $W + Y$ many bad words. Since $\sum_{v_i \in R} b(v_i) = W + Y' \geq W + Y$ by construction, we now look at the ratio. We obtain

$$\frac{\sum_{v_i \in R} g(v_i)}{\sum_{v_i \in R} b(v_i)} = \frac{\sum_{v_i \in R} (w_i t + y_i t)}{W + \sum_{v_i \in R \setminus \{v_0\}} y_i} \leq \frac{Wt + Y't}{W + Y'} = t,$$

which establishes the claim.

Let R be the set, which, if removed, removes $W + Y' \geq W + Y$ many bad words from G. It is easy to see that v_0 must be part of any optimal solution. We know that $\sum_{v_i \in R} b(v_i) \geq W + Y$ and we know

$$\begin{aligned} \frac{\sum_{v_i \in R} g(v_i)}{\sum_{v_i \in R} b(v_i)} &= \frac{\sum_{v_i \in R} (w_i t + y_i t)}{W + \sum_{v_i \in R \setminus \{v_0\}} y_i} \\ &= \frac{\sum_{v_i \in R} w_i t + Y't}{W + Y'} \leq t \end{aligned}$$

i.e., $\sum_{v_i \in R \setminus \{v_0\}} y_i = Y'$. From $\frac{\sum_{v_i \in R} w_i t + Y't}{W + Y'} \leq t$ we can deduce that $t \sum_{v_i \in R} w_i \leq tW$ and thus those items fit in the knapsack.

This establishes the claim. □

Hence, the problem is $\mathcal{NP}$-hard. Even though there exists a factor 2 approximation and even FPTAS for the knapsack problem [113], we chose not to use either of these algorithms. We do so for two reasons. None of these algorithms takes the tree structure of the web page into account. Furthermore, even the simple ones require the nodes to be sorted according to their ratio of good to bad words. This takes $\mathcal{O}(n \log n)$ time. Hence, we opt for a simpler algorithm that runs in linear time (in the number of nodes) and exploits the tree structure to ensure a smooth surfing experience.

4.5 Concept

Since the initial problem is $\mathcal{NP}$-hard, we decided to use a heuristic to select which nodes to remove. This heuristic is based on the following assumptions:

- A node v' that is close to a node v that contains bad words has a higher chance of revealing unwanted content than nodes further away (close in the number of hops in the tree).
- Every node v that contains bad words must be removed.

The reason for the first assumption is based on the fact that these nodes are also close on the web page as it is shown to the user. Hence, the content of these nodes tends to be closely related and thus node v' should be considered

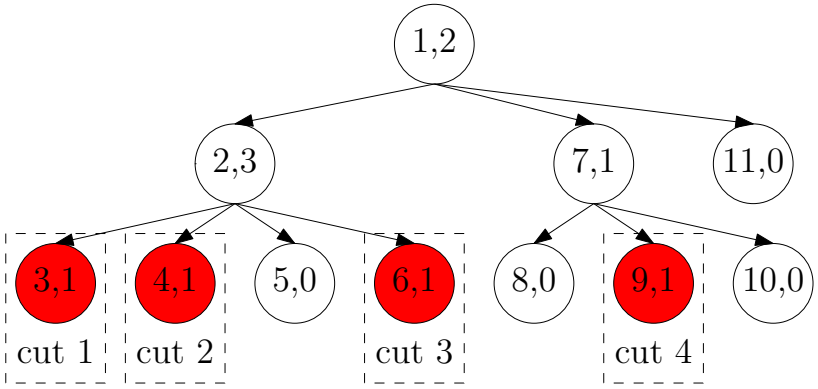


Figure 4.4: An example showing the number of children that contain bad words $c(v)$ for each node in addition to its Id (notation: Id,$c(v)$).

for removal. The second assumption ensures that we remove every user defined keyword. Our algorithm traverses the graph three times; each time starting from the root.

The values $\beta(v)$ and $\gamma(v)$ at every node v are not known to us in the beginning and stored these during the first traversal of the tree. During the second traversal, we consider each node v with $\beta(v) > 0$ and its children. The bad words must originate from either the node v itself or one of its descendants. Since every child v' of v has a value $\beta(v')$, we know which children contribute to $\beta(v)$. Furthermore, the number of children that contribute to $\beta(v)$ are a lower bound on the number of cuts that we need to perform if we do not remove v. We denote this value with $c(v)$. An example of this is shown in Figure 4.4.

Armed with this information, we traverse the tree again – starting from the root. At every node v we consider the ratio $r(v) = \frac{c(v)}{\gamma(v)}$, i.e., the minimal number of cuts we have to perform if we do not remove v divided by the number of good words in v and all its descendants. As soon as the ratio of a node v exceeds a predefined threshold T, we remove it (and thereby all its children) from the tree. To understand the intuition behind this, let us first consider $c(v) = 1$. The ratio is high once there are not many good words left. For larger values of $c(v)$, we implicitly assume that the good words are spread out over its children. Thus, in order to avoid too many cuts, we increase the chance to remove node v – keeping in mind that it is better to err on this side.

4.6 Evaluation

As a proof of concept, we implemented a Firefox extension that uses our filtering mechanism. The extension was developed and tested with Firefox versions 34.0 to 36.01. In the following, we discuss the performance evaluation based on the Firefox extension. Our extension filtering terms in on a web page can be seen in Figure 4.5. On this page, we have defined "algorithm" as a spoiler term. Our extension correctly removes every occurrence of this term including the immediate surroundings of those occurrences. Thus, if a paper was published at "Symp. on Discrete Algorithms", then the whole entry – independent of the

title of the paper – is removed.

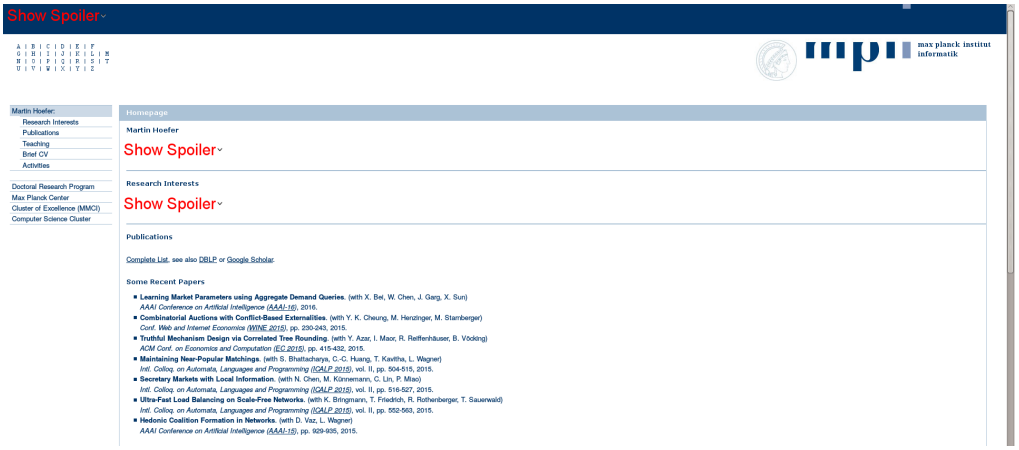


Figure 4.5: Our filter applied to `http://people.mpi-inf.mpg.de/~mhoefer/` using "algorithm" as a spoiler term. It removes three sections on the upper part of the web page.

Website	Section	Keyword	# Nodes (bad / all)
20min.ch	International	IS-	188 / 1753
	Finance	Bank	388 / 2333
	Switzerland	Ecopop	121 / 2135
	Sport	Sieg	200 / 2529
blick.ch	International	IS-	62 / 1203
	Economy	Bank	50 / 1159
	Politics	Ecopop	90 / 801
	Sport	Sieg	58 / 1909
nzz.ch	International	IS-	186 / 4565
	Finance	Bank	530 / 4125
	Switzerland	Ecopop	67 / 2904
	Sport	Sieg	215 / 1913

Table 4.6: News sites including sections, keywords, and page size in number of nodes used for evaluation.

Our test set contains three different news sites as shown in Table 4.6 for which we downloaded the overview page of the sections International, Sports, Economy/Finance and Switzerland/Politics. For each section we defined a keyword that matches part of the content to filter with.

This set up provides us with twelve different pages to process and allows us to explore the trade-off by filtering too much or too little by varying the threshold T. All sites are tested against all spoiler terms which means that we also test sites that do not contain any content that matches the spoiler term. We downloaded the sites on 5th of November 2014 and manually annotated the parts

of the web pages that contain spoilers before the experiments were performed. Note that our approach is also able to remove pictures using the alt attribute.

Depending on the user preference, the threshold T can be set in the preferences of our extension. Decreasing the value of T will lead to lower false negative rates and hence minimize the probability of spoilers being shown. However, smaller values of T also lead to higher false positive rates which means that the likelihood of unrelated content being filtered grows.

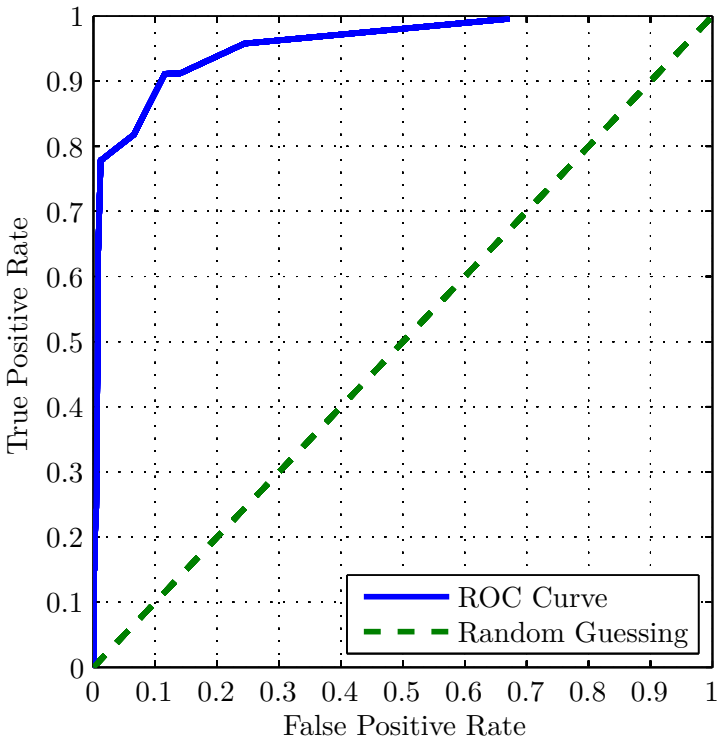


Figure 4.7: ROC curve plot for varying values of T

Figure 4.7 shows the performance trade-off for varying values for T. An ideal system would reach the top left corner and maximize the true positive rate while maintaining low false positive rates. The figure shows, for example, that our system can reach a true positive rate of more than 90% with reasonable false positive rates of less than 15%. The ideal value for T can vary for different web pages because of the different structures in the DOM Tree. However, our test set indicates that a global threshold still delivers good results.

4.7 Conclusion

We have looked at personalized web filtering in this chapter. Since calculating an exact solution is intractable, we resort to using a heuristic. It allows us to remove most of the undesired content in practice. Since we exploit the DOM tree of a web page, we are even able to remove this content while preserving the overall layout of the web page. Thus, we allow the user to surf the web safely

and nearly undisturbed without the fear of seeing the score of her favorite sport team or the latest plot twist from her favorite TV series – even when faced with adversarial input.

5

Clairvoyant Mechanisms

5.1 Introduction

We now analyze adversarial input in a market setting, more concretely in an online auction. We decouple the two issues of such an auction – always needing to have a solution and not knowing the future.

Traditional auctions have a rich theory but only make sense in the presence of at least two bidders. In reality, however, many auctions have a rather low demand, and bidders do not compete concurrently. Instead, bidders appear *online*, one after the other.

A familiar example is booking a seat in an airplane. Prices for a flight fluctuate over time, a known pattern is that seats become more expensive as a flight fills up, because the airline starts to learn that there is demand for the flight. Selling seats in an airplane is not a traditional auction since customers are not bidding against each other. Rather, potential customers check the price well in advance of a flight. If the price is right, they book a seat, sealing the deal with the airline. Airlines generally try to marginally overbook flights, i.e., they sell more tickets than available, assuming that not all customers will actually show up at the gate. Sometimes there are more customers than seats, and the airline must get some customers off the plane. This is usually achieved by having them fly later and giving them cash as compensation. We believe that such compensations are easily covered by the high premium of late customers.[1]

In this chapter we analyze these online auctions. Our bidders come in an online fashion and name their price for a good. The seller can choose to sell the

[1] In reality, airlines do not implement online auctions in the clean form described in this chapter. Airlines do not seem to maximize their profits with this mechanism, probably for psychological reasons. As such, on web pages, flights still can be sold out, instead of just asking for a higher and higher premium for an unexpectedly popular flight.

good for that price, or not sell the good (and hope for a better bid to come in later). Bidder and seller also establish a compensation, in case the good is sold to the customer but the deal is later canceled (in the case of a better bidder showing up, worth paying the compensation). These online auctions need two ingredients: First, a good with a price that may fluctuate over time. Second, customers which want to receive the good (or a reservation for the good) quickly. In particular, the time between the arrivals of two customers should generally be larger than the time a customer is willing to wait for the outcome of her bid. In this case online auctions seem to be a better suitable model than traditional auctions. We believe that such online auctions happen often in practice. Booking flights is the running example in this chapter, but there are plenty of other examples. Selling ad slots on web pages is a popular one. Since the number of page views is not known beforehand, some sold slots might not be served and thus those slots need to be bought back. More examples are real estate sales, selling network services with quality of service guarantees, or concert tickets.

A simple example will show that online auctions become academically interesting only if reasonable compensations are present: Let us assume that a first customer offers a low price but a prohibitively high compensation. If the seller accepts the deal, a next customer offering a much higher price will show up. But the good cannot be sold to the second bidder due to the high compensation of the first bidder. On the other hand, if the seller does not accept the offer from the first bidder, no other customer will show up. No matter how the seller decides regarding the first customer, the mistake could be devastating.

The starting point for our analysis is what we call the *clairvoyant model*, a hybrid online/offline model. In the clairvoyant model, a sequence of all potential customers (their bids and compensations) is known in advance to the seller, but the seller does not know when the sequence stops, i.e., who the last customer of the sequence is. No matter who the last customer is, the seller wants to do a good job, i.e., the seller wants to sell the good to a customer with a high bid *and* keep compensations that accumulated so far low. It turns out that the clairvoyant model is a stepping stone for a deeper understanding of online auctions, sitting nicely between the pure online and offline models. It introduces a novel technique for analyzing online auctions from a theoretical point of view.

Our contributions are as follows: After we have introduced the clairvoyant model, we present an optimal mechanism for it in the case of a single good. The result of that mechanism is a factor Δ worse than an offline mechanism (that knows when the sequence stops, and can simply sell the good to the customer with the highest bid, without having to pay any compensations). In other words, the parameter Δ tells us how nasty the compensations are. It directly tells us the difficulty of an input sequence. If compensations are minimal (just return the money to canceled customers), then we have by definition $\Delta = 1$. We also show an optimal clairvoyant mechanism if there are multiple goods to be sold. If the number of goods is unbounded, however, we prove that the clairvoyant model becomes $\mathcal{NP}$-hard. Based on the results in the clairvoyant model, we study the pure online problem (where the sequence is unknown to the mechanism) in a deterministic setting. If Δ is known, we show that there is a tight gap of $\Theta(\Delta^5)$

between the online and the offline model.

5.2 Related Work

There has been a lot of research of traditional ("offline") auctions, inspired by the seminal papers of Vickrey, Clarke, and Groves ("VCG") [36, 60, 114]. They introduce the notion of truthfulness, which means that no bidder has an advantage if she is not telling the truth about her valuation. There is a large amount of work on traditional auctions, for an overview see, e.g., Nisan, Roughgarden, Tardos, and Vazirani [99].

Due to the sheer amount of work about auctions, we will only touch the surface in this section and thus we restrict ourselves to mentioning general areas. In the single-unit case, where only a single good is sold, it is easy to calculate the allocation and the price since the highest bidder simply gets the good for the second highest price. This quickly becomes more complicated for so-called combinatorial auctions [38,97], where bidders are interested in specific combinatorial subsets of goods. In some settings, the so-called single minded cases, a bidder only has a positive utility if she is allocated all the goods she desires [14,89]. Even though it is still possible to assign the prices and allocate the goods according to a VCG auction, this is no longer computationally feasible [70, 109, 111, 119]. Thus, several approximations exists [5, 81, 82]. Some mechanisms work without money to facilitate trade. Those include mechanisms for political decisions, organ donations, and many more [33, 40, 98, 107]. A detailed overview on auctions can be found in the book by Nisan, Roughgarden, Tardos, and Vazirani [99].

Online mechanisms have been introduced in [53, 84]. In those online mechanisms, the bidders have an arrival and departure time and a valuation for the good. It is assumed that the good expires after a certain period of time, and that a replacement becomes available. In this setting, it was shown that an auction similar to VCG style second price auction is still a viable allocation strategy. The initial motivation behind these kind of online auctions is the WiFi at Starbucks [53]. Customers arrive and then depart some time later with each customer having a valuation for the WiFi. Many papers on online mechanisms mainly focus on truthfulness or other incentive compatible solution concepts, e.g., [64, 85, 102, 103]. An overview of online auctions can be found in [99].

Somewhat related to our online auctions are not even auctions, but the secretary problem [90]. In the classic setting one employer interviews n secretaries, with the goal to hire the best secretary. The employer has to decide right after an interview whether to hire or discard a secretary. Unlike our model, previous decisions cannot be recalled. If secretaries are interviewed in random order, it has been shown that the optimal strategy is to first interview n/e secretaries, and then simply hire the first secretary that is better than all previously interviewed secretaries [90]. It has also been shown that if the input is adversarial (as in our work), the situation is hopeless; the best strategy is to just hire a random secretary, without any interview process [56]. This setting has been adapted to the online auctions in [65]. Instead of secretaries, there are buyers and instead of a job there is a single indivisible good. They present a mechanism that is, if the

buyers appear in random order – as in the original problem – $e+o(1)$-competitive for efficiency and $e^2+o(1)$ competitive for revenue. Since we have the possibility to cancel previous decisions with financial compensations, our model allows more freedom.

The work closest to ours considers online auctions with buyback, introduced independently by Babaioff et al. and Constantin et al. [13, 37]. Both limit the preemption price (paid to reacquire the good) to a constant fraction of the valuation v of a bidder and this fraction is independent of the individual bidder. Lower and upper bounds for deterministic and randomized algorithms depending on the fraction of the preemption price are presented in their work. Our work allows arbitrary values for the preemption price (that can depend on the specific customer) and we analyze how to deal with this very heterogeneous set of customers. This kind of auction is not truthful since a buyer can overstate her preemption price and thus gain if her good is bought back [37]. In [8] the goods cannot be allocated to any subset of bidders, but bidders form a matroid, This is extended to an intersection of matroids in [7], while still limiting the buyback factor. The concept of buyback has also been applied to the knapsack problem [13, 67, 76] where the goods appear in an online fashion and can be removed later on from the knapsack. Buyback is also used in scheduling with eviction [54].

Online algorithms often face two different types of problems: First, they do not know the future, and second, they have to deal with past mistakes. Hartline and Sharp [68, 69] formalized the two types of problems. When problems are analyzed in this framework, they are called incremental problems. This approach has been applied to various problems, e.g., to maximum flow, online median, facility location, and clustering [34, 42, 93, 104]. Our setting is different as we can potentially fix past mistakes with compensations. Nevertheless, our clairvoyant analysis is a relative of incremental problems.

5.3 Model

We consider an online auction. There are r indivisible and identical *goods*. Each bidder b_i is willing to buy exactly one good, and has a *valuation* v_i for being allocated a good. The bidders arrive one after another; whether to allocate a good to a bidder must be decided immediately. Bidders that are not allocated a good cannot be recalled, but bidders that are allocated a good can be recalled. A recalled bidder b_i is willing to return her good if she receives adequate compensation. We call the value *preemption price*, which is paid if the good is *bought back*. The preemption price of bidder b_i is denoted by π_i. In summary, bidder b_i is fully specified by $b_i = (v_i, \pi_i)$. Neither v_i nor π_i are bounded, any value in $\mathbb{R}^+$ is allowed. We assume that the input sequence of bidders $b_1, \ldots, b_n$ is created in advance by an adversary who knows the mechanism that is used to allocate the goods. As described above, if the good of a bidder b_i is bought back, the mechanism has to pay the preemption price. For now, we assume that the mechanism retains the initial valuation v_i of the bidder. We denote this the *retaining* model. In this model we assume that $v_i \leq \pi_i$ for every bidder b_i. We

will show later that this is not necessary and in fact use the model when the value is not retained, which is called the *non-retaining* model.

Let us concentrate on the case of a single good ($r = 1$); we will adapt the definitions for multiple goods in Section 5.5. Let offline(ℓ) denote the highest valuation of the first ℓ bidders, i.e., offline(ℓ) $= \max_{1 \le i \le \ell} v_i$. Since the offline mechanism knows the whole input sequence and when it stops, it can sell the good just to one single bidder, the bidder with the highest valuation.

As discussed in the introduction (and formally proved in Section 5.4.2), the online mechanism cannot be competitive with the offline model. Essentially, an online mechanism has to deal with two different issues: First, it does not know the future, and second, it needs to offer a solution at all times. We will now introduce the *clairvoyant* model, a model between pure online and offline. The clairvoyant model knows the whole sequence $b_1, \ldots, b_n$ of future potential bidders, but does not know when the sequence stops, i.e., who the last bidder of the sequence is. Because of this, a clairvoyant mechanism must offer a solution at all times.

Both pure online and clairvoyant mechanisms may need to accept more than one bidder (and hence buy the good back). Let S be the set of all bidders that have been accepted during the course of a mechanism and let $[\ell]$ denote the set of the first ℓ bidders, i.e., $\{b_1, \ldots, b_\ell\}$. We define $\text{gain}(S, \ell) = \sum_{b_i \in S \cap [\ell]} (v_i - \pi_i) + \max_{b_i \in S \cap [\ell]} \pi_i$. It is the sum of valuations of bidders in S up to bidder b_ℓ, minus the preemption prices for the bidders whose good were bought back.

Since the mechanism does not know when the input sequence stops and it thus can stop anytime, we evaluate any mechanism in its worst round. Specifically, given S, the *gain competitiveness* is defined to be $\max_{1 \le \ell \le n} \frac{\text{offline}(\ell)}{\text{gain}(S,\ell)}$. If we now minimize this over all sets S of accepted bidders, we get the optimal gain competitiveness

$$\Delta = \min_S \max_{1 \le \ell \le n} \frac{\text{offline}(\ell)}{\text{gain}(S, \ell)}.$$

This can be interpreted as the difficulty of the input sequence. In other words, our mechanisms are evaluated in their worst round, i.e., the round in which it has the highest competitive ratio compared to the offline mechanism. This forces our mechanisms into accepting bidders early, and possibly repeatedly, thus paying preemption prices repeatedly. The task is to design mechanisms that choose a set S and thereby allocate the goods to the bidders minimizing gain competitiveness.

We will clarify the terms defined above by presenting a simple example. Let the input sequence be $(1, 2), (4, 100), (50, 60)$. An offline mechanism will accept $b_3 = (50, 60)$ since this is the bidder with the highest valuation. A clairvoyant mechanism must always accept the first bidder since it could also be the last one. Assume that it also accepts the third bidder. We now calculate the gain competitiveness for this set as

$$\max \left\{ \frac{\text{offline}(1)}{v_1} = \frac{1}{1}, \frac{\text{offline}(2)}{v_1} = \frac{4}{1}, \frac{\text{offline}(3)}{v_1 + v_3 - \pi_1} = \frac{50}{1 + 50 - 2} \right\} = 4.$$

Note that this is also optimal since accepting bidder b_2 prevents the mechanism from choosing b_3, hence $\Delta = 4$. This gives us a theoretical insight on the input sequence. No online mechanism could have done better.

As explained, the clairvoyant model sits between offline and online models. It turns out that it is comparable to both models, even though the models are not comparable to each other.

5.4 Auctioning Off a Single Good

We start our analysis by considering the special case of just a single good being sold, i.e., $r = 1$.

5.4.1 Clairvoyant Mechanism

We now present a mechanism that optimally solves the clairvoyant model, giving us insights into what is possible for an online mechanism.

Theorem 5.1. *There exists a clairvoyant mechanism that calculates the set of bidders that should be accepted to solve the online auction for one good optimally, i.e., it calculates Δ. If the inputs are integers, its runtime is polynomial; otherwise it is a FPTAS.*

Proof. We show this via constructive proof. We present an algorithm that determines Δ for an arbitrary input sequence. The algorithm consists of two parts. Let $\delta \in \mathbb{R}^+$ be any fixed value. We first check whether there is a set S of bidders such that $\Delta \leq \delta$, or equivalently, $\delta \cdot \text{gain}(S, k) \geq \text{offline}(k)$ for all k. Afterwards, we perform a binary search to find the optimal value for δ.

We use a dynamic programming approach to calculate the set S of bidders. After bidder b_i has been processed, we want to find a set S of bidders for which $\text{gain}(S, i)$ has always been at least $(1/\delta)$ times the offline solution. We call such a set a *surviving set* up to bidder b_i.

Given a set S of bidders, we define

$$\text{net}(S) = \sum_{b_i \in S} (v_i - \pi_i).$$

Let $m(i) = \max_S\{\text{net}(S) \mid \delta \cdot \text{gain}(S, k) \geq \text{offline}(k) \text{ for } 1 \leq k \leq i\}$ be the maximum net value achieved by a surviving strategy up to bidder b_i. Let $m(0) = 0$. We compute $m(i)$ using the following recurrence relation: $m(i) = \max_{j \in Q_i}(m(j-1)+v_j-\pi_j)$, where $Q_i = \{j \in [1, i] \mid \delta \cdot (m(j-1)+v_j) \geq \text{offline}(i)\}$. That is, the optimal strategy up to bidder b_i is composed of the optimal strategy up to bidder b_{j-1} and then buying back of the good from the latest bidder in this strategy and allocating the good to bidder b_j. The set Q_i is used to filter out the strategies that do not survive from bidder b_j to bidder b_i. If there exists i such that Q_i is an empty set, then there is no solution, because it implies that there are no surviving strategies. Thus, there is no set S that maintains δ-competitiveness.

We now proceed to show by induction that the recurrence correctly computes $m(i)$. First, by the definition of the recurrence, we must have $m(1) = v_1 - \pi_1$. This corresponds to the fact that we must take the first bidder in any case. Otherwise, the competitiveness is infinity.

Our induction hypothesis is that $m(j)$ is correctly computed for $1 \leq j < i$. We will first show that there exists a set S of bidders such that $\delta \cdot \text{gain}(S, k) \geq \text{offline}(k)$ for $1 \leq k \leq i$ and $m(i) = \text{net}(S)$. Let $j = \arg\max_{j' \in Q_i}(m(j'-1) + v_{j'} - \pi_{j'})$. By the definition of $m(j-1)$ and the induction hypothesis, there exists S' such that $\delta \cdot \text{gain}(S', k) \geq \text{offline}(k)$ for $1 \leq k \leq j-1$ and $m(j-1) = \text{net}(S')$. Let $S = S' \cup \{j\}$. We now have $\text{net}(S) = m(j-1) + v_j - \pi_j = m(i)$. Furthermore, for any $j \leq k \leq i$, $\delta \cdot \text{gain}(S, k) = \delta \cdot (m(j-1) + v_j) \geq \text{offline}(i) \geq \text{offline}(k)$. While for any $1 \leq k < j$, we have $\delta \cdot \text{gain}(S, k) = \delta \cdot \text{gain}(S', k) \geq \text{offline}(k)$.

Moreover, we show that S is the set of bidders with maximum net value under the condition that $\delta \cdot \text{gain}(S, k) \geq \text{offline}(k)$ for $1 \leq k \leq i$. Suppose instead that S^* is a set of bidders with the maximum net value and $\delta \cdot \text{gain}(S^*, k) \geq \text{offline}(k)$ for $1 \leq k \leq i$, we will show that $\text{net}(S) = \text{net}(S^*)$. Let $j^* = \max_{b_k \in S^*} k$.

First we show that $j^* \in Q_i$. By the definition of S^*, for all $1 \leq k \leq j^* - 1$, we have

$$\delta \cdot \text{gain}(S^* \setminus \{j^*\}, k) = \delta \cdot \text{gain}(S^*, k) \geq \text{offline}(k).$$

Thus, $S^* \setminus \{j^*\}$ is a valid set until $j^* - 1$. By the optimality of $m(j^* - 1)$, we have $m(j^* - 1) \geq \text{net}(S^* \setminus \{j^*\})$. Thus,

$$\delta \cdot (m(j^* - 1) + v_{j^*}) \geq \delta \cdot (\text{net}(S^* \setminus \{j^*\}) + v_{j^*}) = \delta \cdot \text{gain}(S^*, i) \geq \text{offline}(i)$$

and so $j^* \in Q_i$. Therefore,

$$\begin{aligned} \text{net}(S^*) &\leq m(j^* - 1) + v_{j^*} - \pi_{j^*} && \text{by the optimality of } m(j^* - 1) \\ &\leq m(j-1) + v_j - \pi_j \\ &\qquad j^* \in Q_i \text{ and } j = \arg\max_{j' \in Q_i} m(j'-1) + v_{j'} - \pi_{j'} \\ &= \text{net}(S). \end{aligned}$$

Since S^* is an optimal choice, we have $\text{net}(S) = \text{net}(S^*)$. A surviving strategy can now be constructed by repeatedly taking

$$i' = \arg\max_{i' \in Q_i}(m(i'-1) + v_{i'} - \pi_{i'})$$

then setting $i = i'$ and initializing $i = n$.

It remains to explain how the binary search is performed. Note that Δ is bounded from below by 1 and from above by $\text{offline}(n)$. Each iteration of the binary search reduces the range by half, thus after $\mathcal{O}(\log(\text{offline}(n)/\epsilon))$ iterations of the binary search, we have reduced the range to $(1 + \epsilon)$, which implies Δ is approximated up to a $(1 + \epsilon)$ factor.

In the case where all the input values are integers, there will be at most $\mathcal{O}(\text{offline}(n)^2)$ potential candidates for Δ, since it must be in the form of i/j where $i, j \in [1, \text{offline}(n)]$. Therefore, $\mathcal{O}(\log \text{offline}(n))$ iterations will be sufficient to perform binary search on the potential candidates for Δ. □

5.4.2 About Δ and the Input Sequence

We now formalize and extend the impossibility result from the introduction.

Lemma 5.2. *(1) The value of Δ depends on the input sequence and is unbounded.*

(2) The gain competitiveness of the pure online mechanism is unbounded and independent of Δ.

(3) No randomized pure online mechanism can achieve bounded gain competitiveness if the number r of items is in $o(n)$, i.e., $r \in o(n)$.

Proof. (1) The main idea is that any decision made by a mechanism can turn out to be the wrong one. Let the input be $b_1 = (v_1, \pi_1) = (1, 1), b_2 = (v_2, \pi_2) = (x, x^2), b_3 = (v_3, \pi_3) = (x^2 - \varepsilon, x^2)$ with $\varepsilon > 0$. Any mechanism must accept the first bidder to avoid having unbounded competitiveness. If the clairvoyant mechanism accepts the second bidder, then the third bidder also appears. In this case, the gain competitiveness is approximately x. If the second bidder is not accepted, the third bidder does not appear. Thus, the clairvoyant mechanism has a gain competitiveness of approximately x and x can be arbitrarily large.

(2) This proof follows the same structure. No matter which decision an online mechanism makes, the input sequence will be such that it is the wrong one.

Let the first bidder b_1 have $(v_1 = 1, \pi_1 = 1)$, the second one $b_2 = (x, y)$, and the third bidder $b_3 = (z, z)$. We will set z a such that the pure online mechanism is always wrong. If the pure online mechanism does not accept the second bidder, the second bidder will be the one with the highest valuation and thus the gain competitiveness of the online mechanism will be x and does not depend on Δ (which is roughly 1 if the clairvoyant mechanism accepts the first and second bidder). If the online mechanism does accept the bidder, we set $z \gg x$, but $z < y$. Thus, in this case the pure online mechanism has a gain competitiveness of $\frac{z}{x}$, whereas the clairvoyant mechanism has x (if it accepts the first and third bidder). Since z can be arbitrarily large, the claim follows.

(3) Let $f(\cdot)$ denote an arbitrary, strictly monotonically increasing function. Every bidder b_i has a preemption price of $\pi_i = \infty$.

We first consider $r = 1$. Every bidder b_i has $v_i = f(i)$ (e.g., $v_i = 2^i$) up to and including bidder b_j that has a probability of at most $\frac{1}{n}$ for being allocated the good. Such a bidder b_j exists since a single indivisible good cannot be allocated with a constant probability to n bidders and buying back the good is made impossible by $\pi_i = \infty$. If $v_i = 2^i$, then this mechanism is at least a factor of two worse. If $f(\cdot)$ is the Ackermann function (or an even faster growing function), then the competitive factor becomes potentially infinitely large.

For $r \in o(n)$ we can use the same argument as above, since there must be a bidder who is not allocated the good with probability at most $\frac{r}{n} = o(1)$. □

5.4.3 Bounded Preemption Prices

The impossibility results from the introduction and the previous section exploited that the preemption price could be arbitrarily large. In this section we restrict the previously arbitrarily large preemption prices to be at most ρ times as large as the valuation, i.e., $\rho \geq \frac{\pi_i}{v_i}$ for all $1 \leq i \leq n$. Intuitively, this can either be seen as a simple, reasonable constraint for the customers. If someone values a seat on an airplane with some value v, then losing this seat should not be arbitrarily larger than v. One could also interpret this that every customer also has to buy an insurance whose compensation depends on the premium. If she loses her seat, then the insurance will pay her the preemption price. The price of the insurance is closely related to the preemption price. Hence, the value the seller has from selling a seat increases because of the high insurance fee. This interpretation also guarantees us that there is at most a factor of ρ between v_i and π_i for every bidder b_i. The following results resemble closely those in [13, 37]. The factor ρ allows us to design a mechanism that is 4ρ gain competitive. It accepts a bidder if her valuation is at least by a factor 2 larger than the preemption price of the bidder that is currently allocated the good.

We do a constructive proof by presenting a mechanism that achieves 4ρ gain competitiveness. It is shown in Algorithm 5.4.

Theorem 5.3. *There exists a mechanism that has 4ρ factor gain competitiveness.*

Algorithm 5.4 Constant Factor Mechanism (CFM)

```
π* ← 0
while there is a new bidder b_i do
    if v_i ≥ 2π* then
        buy good back and give it to bidder b_i
        π* ← π_i
    end if
end while
```

Proof. Let $b = (v^*, \pi^*)$ be the last accepted bidder. Our induction hypothesis is that the gain is at least a $\frac{v^*}{2}$ and that v^* is always larger than the sum of preemption prices. The induction base $i = 1$ holds vacuously.

Let us now consider $i \to i + 1$: We make a simple case distinction: If bidder b_{i+1} is not allocated a good, then we know that $v_{i+1} < 2\pi^* \leq 2\rho v^*$. We can combine this with the induction hypothesis, which states that at the current gain is at least $\frac{v^*}{2}$ to obtain that the gain competitiveness is $\frac{2\rho v^*}{\frac{v^*}{2}} = 4\rho$. Since we do not accept a bidder, our induction hypothesis continues to hold. If bidder b_{i+1} is allocated a good, then we know that $v_{i+1} \geq 2\pi^*$. Our previous observation states that v^* is larger than the sum of preemption prices thus far. Combining these two observations establishes the induction hypothesis. Note that we have a gain competitiveness of at least 2.

Combining the two cases yields a gain competitiveness of 4ρ. □

Corollary 5.5. *If $\rho \geq \frac{\pi_i}{v_i}$ for every bidder b_i, then $\Delta \leq 4\rho$.*

5.4.4 Online Mechanism with Δ

This raises the question whether restricting the preemption price is the only way to go. We already know that Δ contains valuable information about the input sequence. But does it contain all the necessary information for an online mechanism to be competitive? We now provide the mechanisms with this information and denote them Δ-online mechanisms. These more powerful online mechanisms can achieve a $\mathcal{O}(\Delta^5)$ factor approximation. Note that this information is not as strong as knowing that the preemption price of every bidder is at most a factor of ρ larger. The clairvoyant mechanism might accept someone whose preemption price is much larger than her valuation.

We briefly describe the mechanism that is $\mathcal{O}(\Delta^5)$ competitive. Simply put, this mechanism accepts bidders with a sufficiently small preemption price (and a high enough valuation to pay back the last bidder). Furthermore, it also accepts bidders that have such a high valuation that the clairvoyant mechanism also has to accept her.

We denote the current bidder with $b = (v, \pi)$ and the last accepted bidder $b^* = (v^*, \pi^*)$. The online mechanism accepts the first bidder for sure, so initially $b^* = (v_1, \pi_1)$. After the first bidder, the current bidder b is accepted for two different reasons: We call bidders *good* if $\pi \leq 2\Delta^2 v$; if a bidder is not good, she is *bad*. The mechanism will accept a good bidder if her valuation $v > 2\pi^*$. We call bidders *crucial* if $v > 2\Delta v^{**}$, where $v^{**} \geq v^*$ is the largest valuation seen so far. The mechanism will accept a crucial bidder if $v > \pi^*/(1 - \frac{1}{\Delta^2})$ holds. The pseudocode is shown in Algorithm 5.6.

Algorithm 5.6 A Δ-online mechanism

```
accept the first bidder and set (v*, π*) = (v_1, π_1)
v** = v_1
while there is a new bidder b_i do
    if π_i ≤ 2Δ²v_i and v_i > 2π* then
        buy good back and give it to bidder b_i
        π* ← π_i
        v* ← v_i
    else if v_i ≥ 2Δv** and v_i > π*/(1 − 1/Δ²) then
        buy good back and give it to bidder b_i
        π* ← π_i
        v* ← v_i
    end if v** = max{v**, v_i}
end while
```

Theorem 5.7. *Given the value of Δ, there exists a mechanism that has gain competitiveness $O(\Delta^5)$ compared to the offline solution.*

Proof. Note that the clairvoyant mechanism will accept every crucial bidder. Let $\bar{b}_1 = (\bar{v}_1, \bar{\pi}_1), \bar{b}_2 = (\bar{v}_2, \bar{\pi}_2), \ldots$ be the subsequence of bidders who are crucial, and

let $\bar{b}_0 = b_1$ be the very first bidder, who will also be accepted by the clairvoyant mechanism. We will prove the theorem by induction over the crucial bidders. Our induction hypothesis is that before $\bar{b}_i$ came, the gain competitiveness of the mechanism is at most $8\Delta^5$. We then prove that before $\bar{b}_{i+1}$ came, the gain competitiveness remains $8\Delta^5$. Before we can continue our proof, we need two helper lemmas.

Lemma 5.8. *If the clairvoyant mechanism accepts a bad bidder $\hat{b} = (\hat{v}, \hat{\pi})$, then the next bidder it will accept must be the first crucial bidder that comes afterward.*

Proof. Let $\bar{b} = (\bar{v}, \bar{\pi})$ be the next bidder clairvoyant mechanism accepts after $\hat{b} = (\hat{v}, \hat{\pi})$, and v^{**} be the maximum valuation of all bidders before $\bar{b}$. Then we must have $\bar{v} > \hat{\pi} > 2\Delta^2\hat{v}$. Note that $v^{**} \leq \Delta\hat{v}$, since otherwise the gain competitiveness of the clairvoyant mechanism will be larger than Δ. Thus, we have $\bar{v} > 2\Delta v^{**}$ and therefore $\bar{b}$ must be a crucial bidder. As clairvoyant mechanism needs to accept all crucial bidders, $\bar{b}$ must be the first crucial bidder after $(\hat{v}, \hat{\pi})$. □

Lemma 5.9. *If b^* is bad, then the next bidder our mechanism accepts must be the first crucial bidder $\bar{b} = (\bar{v}, \bar{\pi})$ that comes afterward. Furthermore, the gain after accepting $\bar{b}$ is at least $\frac{1}{\Delta^2}\bar{v}$.*

Proof. Let $\bar{b}$ be the next crucial bidder after b^*. If b^* is bad, then b^* must be crucial since our mechanism only accepts bad bidders that are crucial. So the clairvoyant mechanism will also accept b^* since it accepts every crucial bidder. By Lemma 5.8, the next bidder after b^* the clairvoyant mechanism will accept is $\bar{b}$. So $\bar{v} - \pi^* \geq \frac{1}{\Delta}\bar{v}$, since otherwise the gain of clairvoyant mechanism will be less than $\frac{1}{\Delta}\bar{v}$. This implies that $\bar{v} \geq \pi^* + \frac{1}{\Delta}\bar{v} > \pi^* + \frac{1}{\Delta^2}\bar{v}$ and therefore $\bar{v} > \pi^*/(1 - \frac{1}{\Delta^2})$. Thus, our mechanism will also accept $\bar{b}$. Let v^{**} be the maximum valuation before $\bar{b}$, then $v^{**} \leq \Delta v^*$. So between b^* and $\bar{b}$, our mechanism will not accept any bidder. □

By our assumption, b^* is the last bidder that our mechanism accepts before $\bar{b}_i$. Thus, if b^* is bad, then $\bar{b}_i$ must be the first crucial bidder after b^*, and our mechanism will accept $\bar{b}_i$. The gain after accepting $\bar{b}_i$ is at least $\frac{1}{\Delta^2}\bar{v}_i$, and the gain competitiveness is at most Δ^2.

If our mechanism does not accept $\bar{b}_i$, then $\bar{v}_i < \pi^* + \frac{1}{\Delta^2}\bar{v}_i$. Moreover, by Lemma 5.9, if our mechanism does not accept $\bar{b}_i$, then b^* is good, and thus $\bar{v}_i < \pi^* + \frac{1}{\Delta^2}\bar{v}_i \leq 2\Delta^2 v^* + \frac{1}{\Delta^2}\bar{v}_i$. Thus, we have $\bar{v}_i - \frac{1}{\Delta^2}\bar{v}_i < 2\Delta^2 v^*$ or equivalently $\bar{v}_i < 2\Delta^2 v^*/(1 - \frac{1}{\Delta^2}) \leq 3\Delta^2 v^*$ (w.l.o.g. assuming $\Delta > 2$, otherwise we can achieve constant factor competitiveness by treating Δ as two in the mechanism). This implies that the current gain competitiveness is at most $6\Delta^2$ using that b^* is a good bidder.

Based on the above analysis and a simple induction we conclude that if our mechanism accepts a bad bidder $b^* = (v^*, \pi^*)$, the gain is at least $\frac{1}{\Delta^2}v^*$ at this moment. It is also easy to see, if b^* is good, then the gain is at least $v^*/2$ (analogue to the proof of Theorem 5.3).

We now combine the previous observations. Let $c = \{\bar{b}_i, c_1, c_2, \cdots c_t\}$ be the sequence of bidders that arrive between $\bar{b}_i$ and $\bar{b}_{i+1}$ (excluding $\bar{b}_{i+1}$). Let $b' = (v', \pi')$ be the last bidder the clairvoyant mechanism accepts. If the clairvoyant mechanism only accepts good bidders in c, then the gain competitiveness between our online mechanism and the clairvoyant mechanism is at most $4\Delta^4$, because $v' \leq 2\pi^* \leq 4\Delta^2 v^*$ holds at all time (otherwise, our online mechanism will accept (v', π')) and the gain of our online mechanism is at least v^*/Δ^2, which implies the gain competitiveness is at most $4\Delta^4$.

Thus, we only need to consider the case when clairvoyant mechanism accepts at least one bad bidder in c (possibly $\bar{b}_i$). By the above analysis, we know that if the clairvoyant mechanism accepts some bad bidder $\hat{c} = (\hat{v}, \hat{\pi})$, then the next bidder it accepts is $\bar{b}_{i+1}$. Furthermore, $v^{**} \leq \Delta\hat{v}$, where v^{**} is maximum valuation before $\bar{b}_{i+1}$.

Before accepting $\hat{c} = (\hat{v}, \hat{\pi})$ the clairvoyant mechanism only accepts good bidders. Now suppose we are at the time right before $\hat{c}$ comes. Suppose, at this time, our online mechanism accepts $b^* = (v^*, \pi^*)$ and clairvoyant mechanism accepts (v', π'). We first consider the case when b^* is good. Then we have $v' \leq 2\pi^* \leq 4\Delta^2 v^*$. Let m be the maximum valuation before $\hat{c}$. We have $m \leq \Delta v'$, and $\hat{bv} \leq 2\Delta m$ (otherwise $\bar{b}_{i+1} = \hat{c}$). Remember that v^{**} is the maximum valuation before $\bar{b}_{i+1}$. Hence, $v^{**} \leq \Delta\hat{v} \leq 2\Delta^2 m \leq 2\Delta^3 v' \leq 8\Delta^5 v^*$.

We now conclude this proof with a simple case distinction. If $b^* = (v^*, \pi^*)$ is good, then the gain competitiveness of our mechanism will never be worse than $8\Delta^5$ after it accepts $b^* = (v^*, \pi^*)$, as the gain is at least $v^*/2$. Moreover, before $\hat{b}$ (with $\hat{v}$) came, both our mechanism and clairvoyant mechanism only accept good bidders, so the gain competitiveness of our mechanism is at most $8\Delta^5$ before this time. So the gain competitiveness of our mechanism is at most $8\Delta^5$ before $\bar{b}_{i+1}$ comes.

On the other hand, if $b^* = (v^*, \pi^*)$ is bad, which implies that $\hat{c} = b^* = \bar{b}_i$, and that the clairvoyant mechanism does not accept any bidder before $\bar{b}_{i+1}$. This implies that $v^{**} \leq \Delta v^*$, and the gain competitiveness of our mechanism in this period is at most Δ^3, since the gain is at least $\frac{1}{\Delta^2} v^*$. □

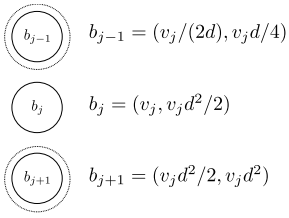
The bound from Theorem 5.7 is tight. We proceed by showing the matching lower bound for any deterministic mechanism.

Theorem 5.11. *Any deterministic Δ-online mechanism has gain competitiveness of $\Omega(\Delta^5)$ compared to the offline solution.*

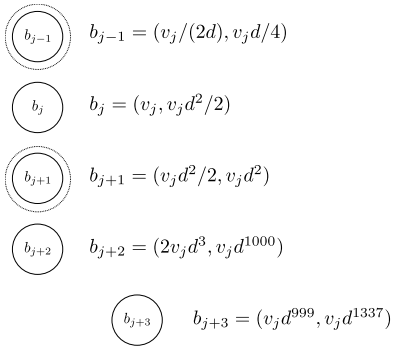
Proof. For any $d > 0$, we will present a sequence of bidders, for which the gain competitiveness between the offline mechanism and the clairvoyant mechanism is at most $2d$, but for any online mechanism, the gain competitiveness is at least $4d^5$. Given Δ, we can set $d = \Delta/2$. Thus, any online mechanism is at least $\Omega(\Delta^5)$ worse than the offline mechanism. The input sequence is depicted in Figure 5.10.

The input sequence starts with bidder b_1 with $(v_1, \pi_1) = (1, 1)$, then the adversary inserts a sequence of bidders $b_{i+1} = (v_{i+1}, \pi_{i+1})$, for $i = 1, 2 \ldots$ with

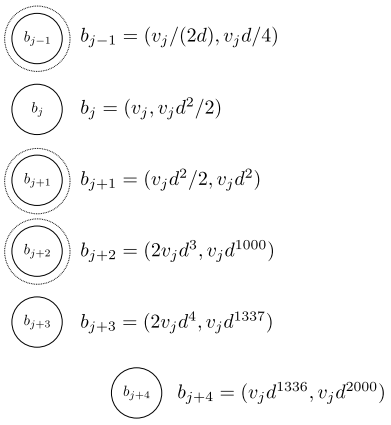
$$(v_{i+1}, \pi_{i+1}) = (2d^i, d^{i+2}).$$



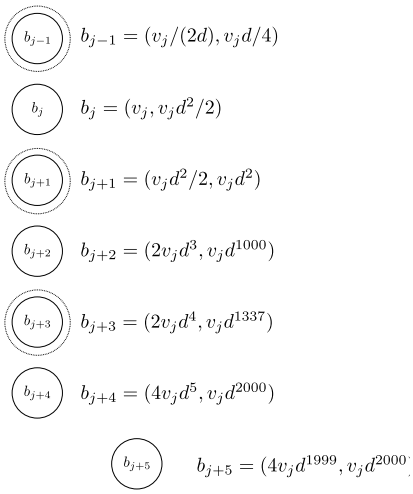
(a) The bidders b_{j-1} and b_{j+1} are accepted by the clairvoyant mechanism. The bidders b_j and b_{j+1} are accepted by the online mechanism resulting in negative gain.



(b) The bidders b_{j-1} and b_{j+1} are accepted by the clairvoyant mechanism. The bidders b_j and b_{j+2} are accepted by the online mechanism. Thus, a bidder $b_{j+3} = (v_j d^{999}, v_j d^{1337})$ would inevitably lead to a gain competitiveness of $\omega(\Delta^5)$.



(c) The bidders b_{j-1}, b_{j+1}, and b_{j+2} are accepted by the clairvoyant mechanism. The bidders b_j and b_{j+3} are accepted by the online mechanism. Thus, a bidder $b_{j+4} = (v_j d^{1336}, v_j d^{2000})$ would inevitably lead to a gain competitiveness of $\omega(\Delta^5)$.



(d) The bidders b_{j-1}, b_{j+1}, and b_{j+3} are accepted by the clairvoyant mechanism. The bidders b_j and b_{j+4} are accepted by the online mechanism. Thus, a bidder $b_{j+5} = (v_j d^{1999}, v_j d^{2000})$ would inevitably lead to a gain competitiveness of $\omega(\Delta^5)$.

Figure 5.10: The bidders accepted by the clairvoyant mechanism are marked with (thinly) dashed lines. The online mechanism accepts by definition b_j. If the online mechanism accepts the bottom left bidder, the bidder on the bottom right appears; resulting in a $\omega(\Delta^5)$ gain competitiveness.

Let b_j be the first bidder in this sequence that the online mechanism accepts. Notice that the online mechanism has to accept one, since otherwise the gain competitiveness is infinity. The clairvoyant mechanism accepts bidder b_{j-1}, but not b_j. The adversary then sets

$$(v_{j+1}, \pi_{j+1}) = ((d^2/2)v_j, d^2 v_j),$$

so that the online mechanism cannot accept this bidder because the new gain would be at most $v_j d^2/2 - v_j d^2/2 - \pi_1 < 0$. The clairvoyant mechanism accepts bidder b_{j+1} to maintain gain competitiveness $\mathcal{O}(\Delta)$.

The next bidder b_{j+2} that comes has

$$(v_{j+2}, \pi_{j+2}) = (d^3 v_j, d^{1000} v_j),$$

so the online mechanism cannot accept this bidder either, since otherwise the adversary can make the next bidder have a valuation of $d^{999} v_j$, which makes the gain competitiveness much larger than $4d^5$. The clairvoyant mechanism does not accept bidder b_{j+2} and still maintains gain competitiveness $\mathcal{O}(\Delta)$.

Bidder b_{j+3} is then

$$(v_{j+3}, \pi_{j+3}) = (2d^4 v_j, d^{1337} v_j).$$

For the same reason, the online mechanism cannot accept this one. The clairvoyant mechanism accepts bidder b_{j+3} to maintain gain competitiveness $\mathcal{O}(\Delta)$. If the online mechanism accepts this bidder, then the clairvoyant mechanism accepts bidder b_{j+2}, but not bidder b_{j+3} (see Figure 5.10).

Bidder b_{j+4} is

$$(v_{j+4}, \pi_{j+4}) = (4d^5 v_j, d^{2000} v_j),$$

and again the online mechanism cannot accept this one. The clairvoyant mechanism does not accepts bidder b_{j+4} and still maintains gain competitiveness $\mathcal{O}(\Delta)$.

At this point, the online mechanism accepted (v_j, π_j), and the gain competitiveness is $\Omega(\Delta^5)$. Thus, the claim follows. □

5.5 Auctions with Several Goods

In this section we consider auctions with r goods. The offline mechanism chooses the best r bidders and never has to pay a preemption price. We let offline(ℓ) denote the sum of the valuations of the best r bidders before bidder b_ℓ. Similar to the single good case, we can denote our strategy by a subset of bidders to whom we will allocate the goods (at least once). The observation is that given the subset S of bidders, the optimal strategy is as follows: First allocate the goods to the first r bidders in S. We ignore the bidders not in S when they arrive. When a new bidder in S arrives, we buy back the good from the bidder who has the smallest preemption price and reallocate it to the new bidder. It is not hard to see that such a greedy strategy must be optimal under the condition that each bidder in S is allocated a good at some point. Given S, let gain(S, ℓ) denote the gain achieved by this strategy after bidder b_ℓ has been processed. The gain competitiveness is defined analogously to the term in Section 5.3.

5.5.1 An Algorithm for Computing Δ with Several Goods

In this section we study computing Δ, the optimal gain competitiveness, which is defined similar to the single good case to be $\min_S \max_\ell \frac{\text{offline}(\ell)}{\text{gain}(S,\ell)}$.

We show that calculating Δ for the clairvoyant mechanism in the multiple goods problem is $\mathcal{NP}$-hard by a reduction from the partition problem, which is known to be $\mathcal{NP}$-hard [55]. Note that we can get a brute force algorithm that runs in $\tilde{\mathcal{O}}(n \cdot 2^n)$ time by simply testing all possible subsets S, the set of customers to whom we will allocate our goods to at least once, and then use the optimal strategy described at the beginning of this section. We can use a heap to store the bidders that occupy the goods, so that in each step the bidder having the minimum preemption price can be found in $\mathcal{O}(\log r)$ time. There are $\mathcal{O}(2^n)$ subsets and for each $\mathcal{O}(n \log r)$ time is required to compute the gain competitiveness. The total time is therefore $\tilde{\mathcal{O}}(n \cdot 2^n)$. On the other hand, we show that a polynomial time algorithm is obtainable if r is a constant. We now provide a dynamic programming algorithm that computes Δ in $\tilde{\mathcal{O}}(n^{r+1})$ time.

We present a dynamic programming algorithm for deciding if $\Delta \leq \delta$ or not in $O(n^{r+1})$ time in the clairvoyant model with several goods.

Theorem 5.12. *Checking whether there is a solution with gain competitiveness of δ in an online auction is r goods can be computed in $\mathcal{O}(n^{r+1})$.*

Proof. For $0 \leq i \leq n$ and $R \subseteq [1, \ldots, n]$ is a set of size at most r. Let $v(i, R)$ denote the maximum possible gain obtained after bidder b_i has arrived and the R is the set of bidders currently occupying the goods. Initially, we set $v(0, \emptyset) = 0$ and all other entries to be $-\infty$. For each state (i, R), define its successor state $\phi(i, R) = (i + 1, R')$, where

$$R' = \begin{cases} R \setminus \{\arg\min_{j \in R} \pi_j\} \cup \{i+1\} & \text{if } |R| = r \\ R \cup \{i+1\} & \text{if } |R| < r \end{cases}$$

Then we let

$$v(i+1, R) = \max \begin{cases} \max_{(i,R') \in \phi^{-1}(i+1,R), |R'|=|R|} v(i, R') + v_{i+1} - \min_{j \in R'} \pi_j \\ v(i, R \setminus \{i+1\}) + v_{i+1} \\ v(i, R) \end{cases}$$

$\phi(i, R)$ is the state where the bidder with the minimum preemption price in R is replaced by bidder b_{i+1}. In the first line of the recurrence, we enumerate all possible predecessors of the state $(i+1, R)$ and try to replace the bidder with the minimum preemption price of it by bidder b_{i+1}. The second line covers the case obtained by adding bidder b_{i+1} to $R \setminus \{i+1\}$ without replacements. The third line indicates that we do not (re)allocate a good to bidder b_{i+1}. This covers all possible cases for a potential optimal strategy up to bidder b_{i+1}, since such a strategy either excludes bidder b_{i+1} or is an optimal strategy up to b_i, and then the good allocated to the bidder with the smallest preemption price is reallocated to bidder b_{i+1}. After the updating, we check if $\delta \cdot v(i+1, R) \geq \text{offline}(i+1)$. If

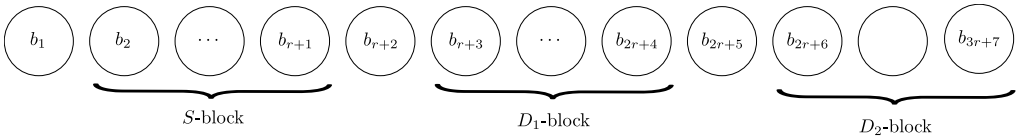


Figure 5.14: The sequence of bidders. The bidders in the S-block represent the initial input from the partition problem. The bidders b_{r+2} and b_{2r+5} both have prohibitively high preemption prices. The S-block consists of r bidders and the two D-block of $r+2$ bidders each.

not, we set $v(i+1, R) = -\infty$ to indicate that the gain competitiveness is larger than δ.

In the end, if there exists R such that $v(n, R)$ is not $-\infty$, then it means there exists a solution such that the gain competitiveness is at most δ. The total number of entries is $O(n \cdot n^r)$ and each entry has a constant number of successor. Thus, the algorithm runs in $O(n^{r+1})$ time. Similar to the single good case (Theorem 5.1), to get an $(1+\epsilon)$ approximation of the difficulty Δ, the number of iterations of the binary search on δ is at most $O(\log \text{offline}(n)/\epsilon)$. In the special case when all input values are integers, $O(\log \text{offline}(n))$ iterations will be sufficient. □

5.5.2 $\mathcal{NP}$-hardness of Calculating the Difficulty Δ

We now show that the problem is $\mathcal{NP}$-hard if r is unbounded. To facilitate the proof, we briefly show the equivalence of the retaining model and the non-retaining model.

Theorem 5.13. *We can take an input sequence from non-retaining model and transform into an equivalent input in the retaining model such that the gain for any mechanism remains unchanged in every round.*

Proof. Consider the input sequence $b_1, \ldots, b_n$. We now turn every bidder $b_i = (v_i, \pi_i)$ into a new bidder $b'_i = (v_i, \pi_i + v_i)$.

Let S be the set of accepted bidders from any mechanism $\mathcal{M}$. We show the gain competitiveness is identical via induction over bidders. For b_1 it holds vacuously.

Thus, let us consider the inductive step from $i-1 \to i$: Because of the induction hypothesis it holds vacuously that for every bidder $b_i \notin S$, the gain remains identical. Thus, let $b_i \in S$ and let b_j be the bidder who was allocated the good before. Let g denote the gain before b_i was allocated the good. In the non-retaining model the new gain is $g + v_i - v_j - \pi_j$ and the retaining model the gain is $g + v_i - (\pi_j + v_j)$. Thus, the claim holds. □

Thus, we can use the two models interchangeably. In the following, it is more convenient to use the non-retaining model.

Theorem 5.15. *Checking whether there is a solution with gain competitiveness of δ in an online auction is $\mathcal{NP}$-hard.*

Proof. In the partition problem, the input is a multiset $S = \{s_1, \ldots, s_r\}$ of positive integers. The task is to decide whether it is possible to partition the set into two sets S_1 and S_2 such that the sum of the numbers in S_1 equals to that of S_2. Let $N = \sum_{i=1}^{r} s_i$, the problem is equivalent to deciding whether there exists a subset of S that sums up to $N/2$. We reduce an instance of the problem to an instance of deciding whether $\Delta \leq \delta$ in an online auction of $r+2$ goods, where $\delta = 1 + \frac{1}{2r}$.

We do so in two steps. First, we provide any mechanism with the option to choose any of r bidders. Now, if the sum of valuations s of the bidders chosen by the mechanism is smaller than $N/2$, then the mechanism is forced to accept a bidder with infinite preemption price. We link preemption price of the first set of bidders with their valuation such that the combined preemption price of the selected bidders is $s \cdot k$ for some k. Afterwards, we present $r+2$ bidders that must be accepted by any mechanism. The linkage of the preemption price of the first set of bidders with their valuation allows us to ensure that the valuation before was not larger than $N/2$. Otherwise the sum of preemption prices would be so high that we can force the mechanism to accept a bidder with infinite preemption price. The sequence of bidders is depicted in Figure 5.14. We now present the details.

Consider the first bidder $b_1 = (v_1, \pi_1)$. We set $\pi_1 = v_1 + N = (2r+1)N$ such that buying back the good from this bidder yields negative gain (for the first $r+2$ bidders). We set v_1 such that $\frac{N+v_1}{v_1} = \delta = 1 + \frac{1}{2r}$ holds, i.e., $v_1 = \frac{N}{\delta - 1} = 2rN$. This allows the mechanism to pick any subset from the following r bidders without violating the gain competitiveness. These r bidders, denoted the *S block*, can be described as follows: $v_{i+1} = s_i$ and $\pi_{i+1} = s_i \cdot k$ with k being chosen such that buying back a good from any bidder from this block is not possible, e.g., $k = \pi_1 = (2r+1)N$. Now, bidder $b_{r+2} = (v_{r+2}, \infty)$ arrives. Her preemption price is chosen such that accepting her will be "fatal". We now show that if the mechanism has accepted bidders such that $s < N/2$, it is forced to accept this one. The number of goods is chosen such that the offline$(r+2)$ is simply the sum of the first $r+2$ bidders. The mechanism has a gain of $s + v_1$. Thus, we want $\frac{v_{r+2}+N+v_1}{N/2+v1} = \delta = 1 + \frac{1}{2r}$. Simple math yields

$$v_{r+2} = \delta \cdot (N/2 + v_1) - (N + v_1) = (1 + \frac{1}{2r}) \cdot (N/2 + 2rN) - (N + 2rN) = \frac{N}{2} + \frac{N}{4r}.$$

Now, if $s < N/2$, then it is clear that bidder b_{r+2} needs to be accepted.

The next $r+2$ bidders are part of a so called *dominant block* D_1. They are bidders whose valuation T is so high that any mechanism has to accept all of them in order to be able to maintain the gain competitiveness. To facilitate notation, we use $v(D_1)$ as a short hand form for $v(D_1) = \sum_{i=r+3}^{2r+4} v_i = (r+2)T$, i.e., the sum of all bidders in the dominant set D_1. At this point, we have offline$(2r+4) = v(D_1)$ and even though the mechanism has the same set of bidders, it has to pay the preemption prices. Thus, the gain competitiveness is now

$$\frac{v(D_1)}{v(D_1) - s \cdot k - \pi_1} = \frac{(r+2)T}{(r+2)T - (s+1)(2r+1)N} \leq \delta.$$

Straightforward math yields that this holds for any $T \geq \frac{(s+1)(2r+1)N}{\frac{r+2}{2r}}$.

We now present another "fatal" bidder, $b_{2r+5} = (v_{2r+5}, \infty)$ with an even higher valuation than the bidders from D_1. We want to force the algorithm to accept this bidder if $s > N/2$. Thus,

$$\frac{v(D_1) - T + v_{2r+5}}{v(D_1) - N/2 \cdot k - \pi_1} = \delta,$$

i.e.,

$$\begin{aligned} v_{2r+5} &= \delta \cdot (v(D_1) - N/2 \cdot k - \pi_1) - (v(D_1) - T) \\ &= (1 + \frac{1}{2r}) \cdot ((r+2)T - (N/2+1) \cdot (2r+1)N) - ((r+2)T - T). \end{aligned}$$

It follows that if $s > N/2$, then the bidder needs to be accepted in order to maintain δ gain competitiveness. To ensure that this would result in a gain competitiveness larger than δ, we need another dominant block – D_2. Thus, the last $r+2$ bidders have again a very high valuation (much higher than the previous dominant block and v_{2r+5}) and every mechanism that wants to be δ gain competitive needs to accept every bidder from this block. Thus, only when s is equal to $N/2$ does the mechanism have a gain competitiveness of δ.

For completeness' sake, we now argue why while accepting the bidders from a dominant block, the gain competitiveness does not exceed δ. Let $\text{offline}_{\text{old}}$ denote the value of the offline solution before block D_i, and gain_{old}, the gain before block D_i. Consider the bidder from each solution whose good is now allocated to a bidder from the dominant set. Let b_i be such a bidder in the offline solution and b_j be such a bidder in the set of currently accepted bidders from the mechanism. Thus, the absolute change in gain for the offline solution is $T - v_i$ and for the mechanism it is $T - v_j - \pi_j$. We can combine this to $\frac{\text{offline}_{old} + T - v_i}{\text{gain}_{old} + T - v_j - \pi_j}$. By choosing T sufficiently large, the gain competitiveness remains below δ. □

5.6 Conclusion

We have introduced the clairvoyant model, a hybrid between an online and an offline model. In it a mechanism knows the input sequence – similar to an offline mechanism. But, it always has to have a good solution in case the sequence ends – similar to an online mechanism. By separating these two issues an online mechanism faces, we made an analysis possible.

We have proved that an optimal clairvoyant mechanism exists and that we can compute the sequence of bidders that should be accepted in polynomial time. This mechanism is a factor of Δ worse than the offline solution; a factor that depends on the input sequence. There exists a mechanism that is equipped with this information that is $\mathcal{O}(\Delta^5)$ competitive to the offline solution; without it any pure online mechanism has an arbitrarily large competitive factor. Thus, using this small piece of information allows us to deal with adversarial input in such an online auction.

6

Automatable Jobs and Automatable Tasks

6.1 Introduction

In our final chapter, we analyze the job market. Computerization is considered to be one of the biggest socio-economic challenges. What is the foundation of the recent worries about many jobs being affected by automation [24, 25, 50, 52]?

Why did the last few years see dramatic technological progress regarding self-driving cars [61], board games [112], automatic language translation [9], or face recognition [86]? One reason is big data. While "intelligent algorithms" in the past were restricted to learning from data sets with a few thousand examples, we now have exabytes of data. Learning becomes even more powerful if you combine big data with a highly parallel hardware, stirred by the success of graphics processing units (GPUs). However, both of these technological advancements needed to be harvested, and they are with the advent of so-called deep learning algorithms, which have blown the competition away, starting with voice recognition [43]. As a consumer, you can already witness some of these advancements on your smartphone, and many more will happen soon. We believe that these advancements will revolutionize white collar work and (with a little help from sensors and robotics) also blue collar work. In contrast to previous waves of innovation, this time new emerging jobs might not be able to compensate for the jobs that are endangered by the new technology.

In their seminal paper, Frey and Osborne [52] quantitatively study job automation, predicting that 47% of US employment is at risk of automation. In order to calculate this number, Frey and Osborne labeled 70 of the 702 jobs

from the O*NET OnLine job database[1] manually as either "automatable" or "not automatable". Then, for the remainder of the jobs in the O*NET database, they computed the automation probability as a function of the distance to the labeled jobs.

But the results of Frey and Osborne are opaque, one either believes their "magic" computerization percentages, or one has doubts. We want anybody to be able to easily understand and argue about our results, by incorporating the unique tasks of each job. This additional depth will help laymen as well as job experts to argue about potential flaws in our methodology.

If we know that a job is 100% automatable, we also know that every task of that job must be completely automatable. But what if a job is 87% automatable? Is every task 87% automatable? Or are 87% of the tasks completely automatable, and 13% not at all? We want to forecast which tasks of a job are safe and which tasks are automatable.

In order to calculate the automation probability for a task, we first need to determine its share of a job (Section 6.4). Based on this, we are able to assign each task a probability to be automated such that the weighted average of the probabilities is equal to the probability of the corresponding job (Section 6.6.1). During our evaluation (Section 6.6.2), we discover a few suspicious results in the probabilities by Frey and Osborne, e.g., a surprisingly high automation probability of 96% for the job *compensation and benefits managers*. We conclude our paper by analyzing the correlation between various properties of a job and its probability to be automated (Section 6.7). We show for example that there is a strong negative correlation between the level of education required for a job and its probability to be automated.

Our complete results can be found at `http://jobs-study.ethz.ch`.

6.2 Related Work

The current effects of automation have been studied intensively in economics. Most studies agree that some routine tasks have already fallen victim to automation [10, 12, 59]. A task is routine if "it can be accomplished by machines following explicit programmed rules" [12]. With computers being able to do routine tasks, the demand for human labor performing these tasks has decreased. But on the other hand, the demand for college educated labor has increased over the last decades [19, 115, 116]. The effect is more pronounced in industries that are computer-intensive [11]. As a consequence of this, the employment share of the highest skill quartile has increased. In addition to more people being employed in the highest skill quartile, the real wage for this quartile has increased faster than the average real wage. Service occupations, which are non-routine, but also not well paid, have also seen an increase in employment share and in real hourly wage. Thus, both, employment share and real wage, are U-shaped with

[1] O*NET is an application that was created for the general public to provide broad access to the O*NET database of occupational information. The site is maintained by the National Center for O*NET Development, on behalf of the U.S. Department of Labor, Employment and Training Administration (USDOL/ETA); see https://www.onetonline.org/

respect to the skill level [10]. This employment pattern is a phenomenon that is called polarization. This is not unique to the US, but can also be observed, e.g., in the UK [59]. These papers make important observations about the effects that automation already has. Until now, routine tasks are the ones most affected, but more and more tasks can nowadays be performed by a computer. We focus on the future and try to predict which tasks will be automated next.

John Keynes predicted already in 1933 that there will be widespread technological unemployment "due to the means of economising the use of labour outrunning the pace at which we can find new uses for labour" [78]. Automation might be the technology, where this becomes true [24, 25, 50, 52]. "Automation of knowledge work", "Advanced robotics", and "Autonomous and near-autonomous vehicles" are considered to be 3 out of 12 potentially economically disruptive technologies [92]. Computer labor and human labor may no longer be complements, but competitors. Automation might be the cause for the current stagnation [24]. There might be too much technological progress, which causes high unemployment. A trend that could be going on for years, but was hidden by the housing boom [32].

The seminal paper by Frey and Osborne is the first to make quantitative claims about the future of jobs [52]. Together with 70 machine learning experts, Frey and Osborne first manually labeled 70 out of 702 jobs from the O*NET database as either "automatable" or "non automatable". This labeling was, as the authors admit, a subjective assignment based on "eye balling" the job descriptions from O*NET. Labels were only assigned to jobs where the whole job was considered to be (non) automatable, and to jobs where the participants of the workshop were most confident. To calculate the probability for non-labeled jobs, Frey and Osborne used a probabilistic classification algorithm. They chose 9 properties from O*NET as features for their classifier, namely "Finger Dexterity", "Manual Dexterity", "Cramped Work Space, Awkward Positions", "Originality", "Fine Arts", "Social Perceptiveness", "Negotiation", "Persuasion", and "Assisting and Caring for Others".

The results from Frey and Osborne for the US job market were adopted to other countries, e.g., Finland, Norway, and Germany [18,101]. This was done by matching each job from O*NET to the locally used standardized name. Due to differences in the economies, a different percentage of people will be affected by this change, e.g., only one third in Finland and Norway are at risk compared to 47% in the US.

6.3 Model

We are given a set of jobs $J = \{j_1, \ldots, j_n\}$. Each job j_i consists of a set of tasks $T_i = \{t_i^1, \ldots, t_i^m\}$, where every task belongs to exactly one job. We call two tasks t_i^k and $t_{i'}^{k'}$ related if and only if these tasks are similar according to O*NET. Two jobs with related tasks are also called related. An example with 3 jobs is depicted in Figure 6.1.

O*NET provides us for each task t_i^k with the information how often it is performed. This information was gathered by asking job incumbents and occu-

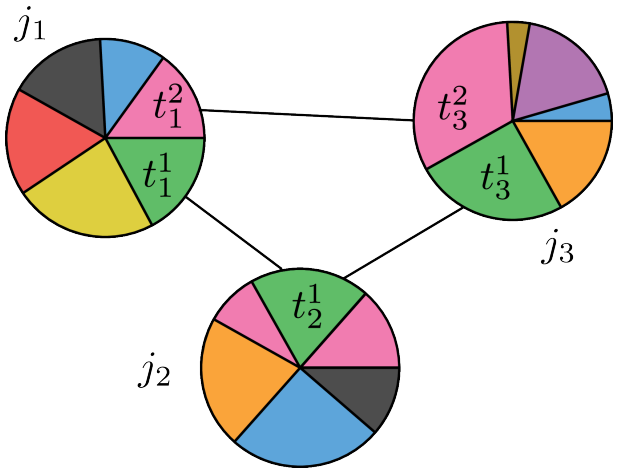


Figure 6.1: This figure shows a small example consisting of three jobs represented by circles. The share of each task of a job is shown by its sector. Two tasks are connected by a line if and only if they are related, i.e., similar according to O*NET. Task t_1^1 from job j_1 and task t_2^1 from job j_2 are related as indicated by the line connecting them. Note that this relationship is not transitive. Thus, tasks t_1^1 and t_3^1 do not need to be related.

pational experts. The options are "yearly or less", "more than yearly", "more than monthly", "more than weekly", "daily", "several times daily", and "hourly or more". O*NET provides a percentage for each of the 7 options. We denote these frequencies of task t_i^k with $f_1(t_i^k), \ldots, f_7(t_i^k)$. Since these values are percentages, for every task t_i^k they sum up to 100%, i.e., $\sum_{\ell=1}^{7} f_\ell(t_i^k) = 1$.

Each job j_i has a given probability $p(j_i)$ to be automated. We want to use $p(j_i)$ to calculate a probability to be automated for each task of this job.

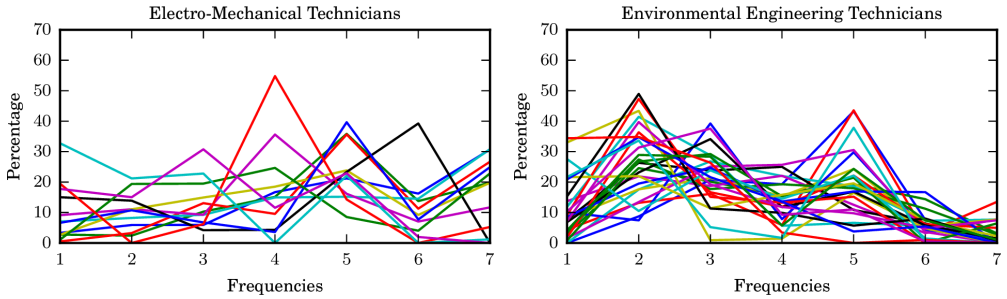
6.4 From Task Frequencies to Task Shares

We use the frequencies with which a task is performed to assign each task t_i^k its share $s(t_i^k)$. For every task t_i^k, the share denotes how much time is spent doing this task, such that $\sum_{t_i^k \in T_i} s(t_i^k) = 1$. The frequency values from O*NET do not fulfill this property; their values are very consistent for one job, but they can vary a lot between different jobs and might even seem to contradict each other. An extreme example can be seen in Figure 6.2. The seven frequency options provided by O*NET are on the x-axis and on the y-axis is the corresponding value of each option.

To make use of the high consistency within a job, we decided that the share of a task is a weighted average of its frequencies, i.e., $s(t_i^k) := \sum_{\ell=1}^{7} x_i^\ell f_\ell(t_i^k)$. We want to calculate the job specific coefficients x_i^ℓ. Let us illustrate these coefficients with a simple example. If $x_i^7 = 0.1$, then a task t_i^k that is done exclusively "hourly or more" (i.e., $f_7(t_i^k) = 1$) makes up 10% of job j_i.

We want these coefficients to satisfy a few assumptions. If O*NET states that a task is done "hourly or more", then the share of this task should be higher than the share of a task that is done "several times daily". This translates to

$x_i^\ell \leq x_i^{\ell+1}\ \forall \ell \in \{1, \ldots, 6\}$ and $0 \leq x_i^1$.



(a) Electro-Mechanical Technicians **(b)** Environmental Engineering Technicians

Figure 6.2: The number of tasks and the frequencies assigned to them can differ significantly even for related jobs.

These constraints neither use that jobs are related nor do they define the coefficients uniquely. Both issues are solved if we require the coefficients x_i^ℓ and $x_{i'}^\ell$ for two related jobs j_i and $j_{i'}$ to be similar. The intuition behind this is that the frequencies of O*NET for related jobs are not independent of each other either, but rather should be similar as well. Occupational experts who have rated the frequency of a task of a job, are likely to have rated the frequencies of related jobs.

The coefficients cannot be identical without violating the other constraints. Jobs have a different number of tasks and the frequencies are task specific. The example in Figure 6.2 highlights this. It is therefore easy to see that we cannot have the same coefficients for two related jobs and fulfill the equality $\sum_{t_i^k \in T_i} s(t_i^k) = 1$ for both jobs simply because the number of tasks can differ a lot.

Thus, we allow a bit of slack in the coefficients of related jobs. We use the variable $x_{i,i'}^\ell$ to express the difference between the coefficients x_i^ℓ and $x_{i'}^\ell$ for two related jobs $j_i, j_{i'}$. Formally, we define it as $x_{i,i'}^\ell := \max\{x_i^\ell - x_{i'}^\ell, x_{i'}^\ell - x_i^\ell\}$. This yields the following linear program, which minimizes the overall slack:

$$
\begin{aligned}
\text{minimize} \quad & \textstyle\sum x^{\ell}_{i,i'} & & \\
\text{s.t.} & & & \\
x^{\ell}_{i,i'} \geq \quad & x^{\ell}_{i} - x^{\ell}_{i'} & \forall \ell & \\
& & \forall j_i, j_{i'} \in J \text{ that are related} & \\
x^{\ell}_{i,i'} \geq \quad & x^{\ell}_{i'} - x^{\ell}_{i} & \forall \ell & \\
& & \forall j_i, j_{i'} \in J \text{ that are related} & \\
\sum_{t^k_i \in T_i} \sum_{\ell=1}^{7} x^{\ell}_{i} f_{\ell}(t^k_i) \leq \quad & 1 + \varepsilon & \forall j_i \in J & \\
\sum_{t^k_i \in T_i} \sum_{\ell=1}^{7} x^{\ell}_{i} f_{\ell}(t^k_i) \geq \quad & 1 - \varepsilon & \forall j_i \in J & \\
x^{1}_{i} \geq \quad & 0 & \forall j_i \in J & \\
x^{\ell}_{i} \geq \quad & x^{\ell-1}_{i} & \forall j_i \in J & \ell \in \{2, \ldots, 7\}
\end{aligned}
$$

We set ε to 0.01. The resulting LP has 169,372 variables in its objective function. Since there are 735 jobs,[2] this means that a job is related to approximately 32.9 other jobs on average. The value of the objective function is 24.6, i.e., for two related jobs the coefficients differ only by 0.000145 on average. For comparison, the average value of a coefficient is 0.060. Our complete results can be found online at `http://jobs-study.ethz.ch`.

6.5 From Jobs to Tasks

Knowing the shares of the tasks enables us to set up a linear program that calculates for each task the probability to be automated. We want that the weighted average of the automation probabilities $p(t^k_i)$ of the tasks of a job j_i can explain the automation probability $p(j_i)$ of the job, i.e., $\sum_{t^k_i \in T_i} p\left(t^k_i\right) \cdot s\left(t^k_i\right) \approx p\left(j_i\right)$. Furthermore, we want to assign related tasks similar automation probabilities. To do this, we define a variable $t^{k,k'}_{i,i'}$ for each pair of related tasks t^k_i and $t^{k'}_{i'}$. It denotes the probability difference that we assign to the two tasks. Formally, it is defined as $t^{k,k'}_{i,i'} := \max\{p(t^k_i) - p(t^{k'}_{i'}), p(t^k_i) - p(t^{k'}_{i'})\}$. We want to minimize the sum of these variables, i.e., the sum of the probability difference of all related tasks.

Combining these requirements with necessary conditions to have meaningful probabilities, i.e., $0 \leq p(t^k_i) \leq 1$, yields the following linear program:

[2] We consider slightly more jobs than Frey and Osborne, since we use the finest granularity available from O*NET.

$$
\begin{aligned}
\text{minimize} \quad & \sum t_{i,i'}^{k,k'} & \\
\text{s.t.} & & \\
p(t_i^k) - p(t_{i'}^{k'}) \leq \quad & t_{i,i'}^{k,k'} & \forall t_i^k, t_{i'}^{k'} \text{ that are related} \\
p(t_{i'}^{k'}) - p(t_i^k) \leq \quad & t_{i,i'}^{k,k'} & \forall t_i^k, t_{i'}^{k'} \text{ that are related} \\
\sum_k p(t_i^k) \cdot s\left(t_i^k\right) \leq \quad & p(j_i)\,(1+\varepsilon) & \forall j_i \in J \\
\sum_k p(t_i^k) \cdot s\left(t_i^k\right) \geq \quad & p(j_i)\,(1-\varepsilon) & \forall j_i \in J \\
\\
p(t_i^k) \geq \quad & 0 & \forall j_i \in J, t_i^k \in T_i \\
p(t_i^k) \leq \quad & 1 & \forall j_i \in J, t_i^k \in T_i \\
t_{i,i'}^{k,k'} \geq \quad & 0 & \forall t_{i,i'}^{k,k'} \\
t_{i,i'}^{k,k'} \leq \quad & 1 & \forall t_{i,i'}^{k,k'}
\end{aligned}
$$

We set ε to 0.01.

6.6 Linear Program Results

We now analyze the results of the linear program as described above. Later on, we will look at a small refinement to automatically detect inconsistencies in our results.

6.6.1 Task Probabilities

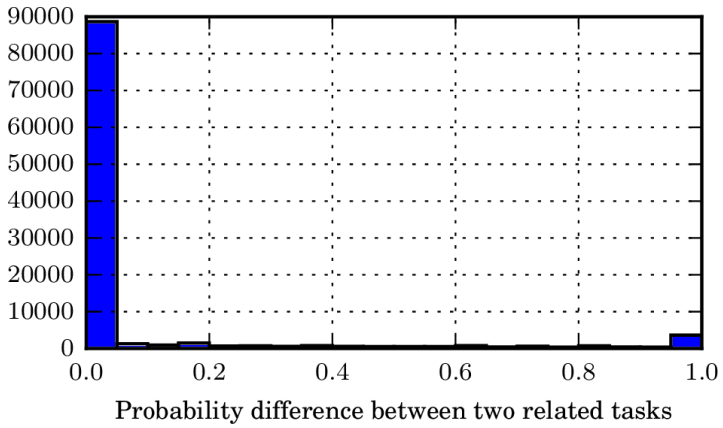


Figure 6.3: A histogram of the probability with which two related tasks differ.

The linear program as described in Section 6.5 has 105,748 variables in its objective function and it has a minimal value of 9,846. This means that two related tasks differ, on average, with regard to their probability by 9.3%. The complete results can be found online at `http://jobs-study.ethz.ch`. The

Task Description	p	Share
Write decisions on cases.	1	5.1
Instruct juries on applicable laws, direct juries to deduce the facts from the evidence presented, and hear their verdicts.	1	3.4
Monitor proceedings to ensure that all applicable rules and procedures are followed.	1	8.0
Advise attorneys, juries, litigants, and court personnel regarding conduct, issues, and proceedings.	1	6.2
Interpret and enforce rules of procedure or establish new rules in situations where there are no procedures already established by law.	1	5.4
Conduct preliminary hearings to decide issues such as whether there is reasonable and probable cause to hold defendants in felony cases.	1	3.9
Rule on admissibility of evidence and methods of conducting testimony.	0.94	5.3
Preside over hearings and listen to allegations made by plaintiffs to determine whether the evidence supports the charges.	0.46	5.9
Perform wedding ceremonies.	0.39	2.7
Read documents on pleadings and motions to ascertain facts and issues.	0	10.1
Research legal issues and write opinions on the issues.	0	6.5
Settle disputes between opposing attorneys.	0	4.6
Participate in judicial tribunals to help resolve disputes.	0	6.6
Rule on custody and access disputes, and enforce court orders regarding custody and support of children.	0	6.3
Sentence defendants in criminal cases, on conviction by jury, according to applicable government statutes.	0	4.0
Grant divorces and divide assets between spouses.	0	4.7
Award compensation for damages to litigants in civil cases in relation to findings by juries or by the court.	0	3.8
Supervise other judges, court officers, and the court's administrative staff.	0	8.5

Table 6.4: The automation probability and the share of each task of *Judges, Magistrate Judges, and Magistrates"*. The automation probability of this job is 40%.

histogram of the probability difference between two related tasks is shown in Figure 6.3. A majority of related tasks is assigned a similar probability. A small fraction of related tasks is assigned diametrically opposed probabilities, which seems startling. It can be reconciled by considering that neither the classification by Frey and Osborne nor the classification of tasks being related by O*NET are perfect.

One example that highlights this are the two jobs *computer programmer* and *software developers, applications*. These two jobs have many related tasks, but the probabilities of these jobs differ a lot (4% for *software developers, applications*, 48% for *computer programmers*). Hence, the diametrically opposed probabilities are necessary to meet the constraints of the linear program.

In the following, we present a few selected jobs to illustrate our results. The first example is *chemists*. This job has an automation probability of 10% according to Frey and Osborne. Only one task has, according to our linear program, a high probability of being automated: "Induce changes in composition of substances by introducing heat, light, energy, or chemical catalysts for quantitative or qualitative analysis." Other simple mechanical tasks have been assigned low automation probabilities. We will revisit this job in Section 6.6.2.

Next up: *judges*. Their automation probability is 40%. The tasks, their probabilities, and their shares are shown in Table 6.4. The tasks that can be automated can be grouped in two sets: preliminary hearings which includes making first assessments, and ensuring that the procedures in court are followed.

The tasks that involve sentencing (or the preparation thereof) have been assigned low automation probabilities.

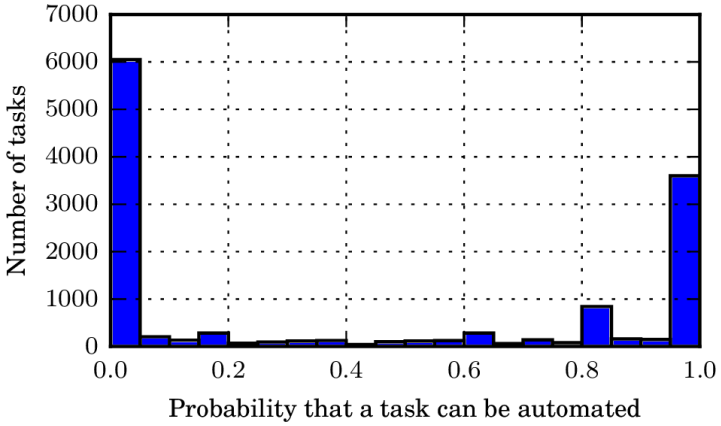


Figure 6.5: A histogram of the probabilities that tasks can be automated. Nearly all tasks are assigned either 0 or 1. This simplifies arguing whether our classification is correct.

Figure 6.5 shows the histogram of the probabilities from our linear program. The probabilities for most tasks are either very high or very low and only a few tasks have a probability in-between. This desired side effect of our linear program helps us to achieve our goal of allowing job experts (and laymen) to argue about the validity of our results. We invite the reader to have a look at other jobs at `http://jobs-study.ethz.ch`.

6.6.2 Inconsistency Detection

To evaluate our approach and check for inconsistencies, we use a variant of cross-validation. For every job j_i, we create a linear program without job j_i. This yields a probability $p_i(\cdot)$ for every task but the tasks from j_i. Afterward, we calculate the new probability $p'(j_i)$ that job j_i can be automated. We do this by setting the probability $p'(t_i^k)$ of each task t_i^k to the average of all tasks that are related to it. We denote the set of related tasks by $N(t_i^k)$. Formally, we set $p'(t_i^k) := \frac{1}{|N(t_i^k)|} \sum_{t_{i'}^{k'} \in N(t_i^k)} p_i(t_{i'}^{k'})$.

We first compare $p'(t_i^k)$ with $p(t_i^k)$. The difference between these two probabilities should be small for the majority of the tasks. This is indeed what can be seen in Figure 6.6. The histogram of $p'(t_i^k) - p(t_i^k)$ shows that nearly all tasks have similar probabilities in both approaches. The average absolute difference is less than 20%. The distribution is centered around 0. Its mean is less than 0.05%.

By combining the new probability of each task with its share, we can calculate the new probability of job j_i by using a weighted average. This allows us to compare $p'(j_i)$ with $p(j_i)$.

We have plotted this difference, i.e., $p(j_i) - p'(j_i)$, in Figure 6.7. We can see that the difference is centered around 0%; with the average absolute difference being less than 20%. For more than half of the jobs our probability differs by

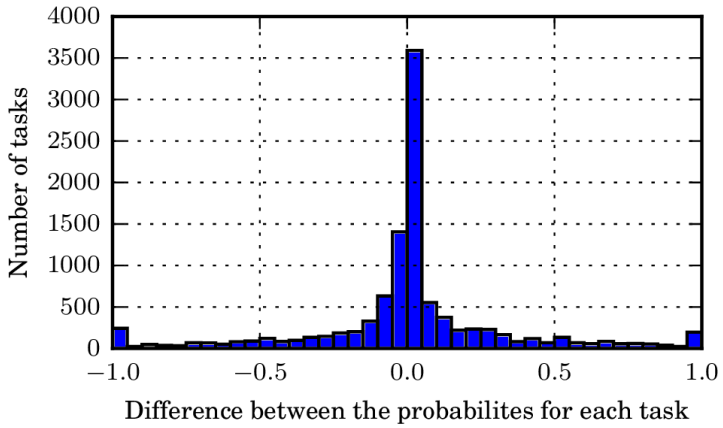


Figure 6.6: A histogram of $p(t_i^k) - p'(t_i^k)$ for all tasks. This plot shows that most tasks have a similar probability in both approaches. The values are sharply distributed around 0.

less than 20% from Frey and Osborne [52]. Most interesting are the jobs whose probability differs significantly. We now have a look at a few of them.

There are jobs where our probability is more than 80% smaller than the one by Frey and Osborne. One job is *compensation and benefits managers*. We assigned it a probability $p'(j)$ to be automated of 9.1%; compared to $p(j) = 96\%$ by Frey and Osborne. We do not claim to know the true value, but we can look at the job and compare it to the probabilities of jobs we consider similar. Notice that this is conceptually similar to what our linear program does and thus might be biased. The tasks of this job are shown in Table 6.8. If we manually compare them to related tasks, we conclude that they do not seem to be automatable in the next few decades. We do favor our result over the result of Frey and Osborne.

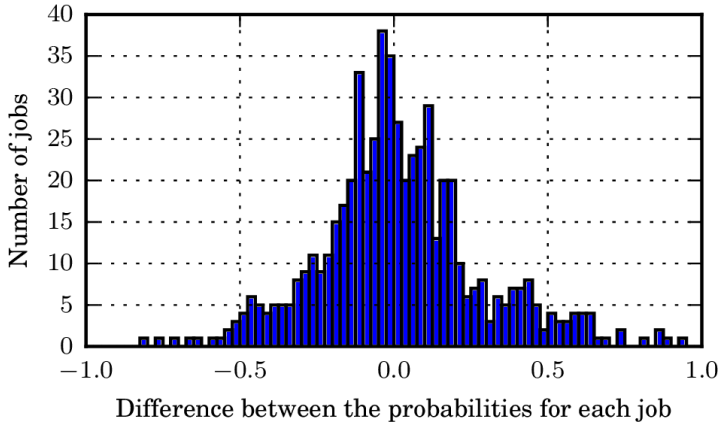


Figure 6.7: A histogram of the difference between the probability by Frey and Osborne with our probability. The distribution is centered around 0 and a majority of the jobs differs by less than 20%.

There is only one job that we assign a much higher probability than Frey and

Task Description	p	p'
Advise management on such matters as equal employment opportunity, sexual harassment and discrimination.	1	0.15
Study legislation, arbitration decisions, and collective bargaining contracts to assess industry trends.	1	0
Fulfill all reporting requirements of all relevant government rules and regulations, including the Employee Retirement Income Security Act (ERISA).	1	0.20
Investigate and report on industrial accidents for insurance carriers.	1	0.12
Represent organization at personnel-related hearings and investigations.	1	0
Analyze compensation policies, government regulations, and prevailing wage rates to develop competitive compensation plan.	1	0.5
Mediate between benefits providers and employees, such as by assisting in handling employees' benefits-related questions or taking suggestions.	1	0.42
Prepare detailed job descriptions and classification systems and define job levels and families, in partnership with other managers.	1	0
Prepare personnel forecasts to project employment needs.	1	0
Direct preparation and distribution of written and verbal information to inform employees of benefits, compensation, and personnel policies.	1	0
Manage the design and development of tools to assist employees in benefits selection, and to guide managers through compensation decisions.	1	0
Design, evaluate and modify benefits policies to ensure that programs are current, competitive and in compliance with legal requirements.	1	0
Administer, direct, and review employee benefit programs, including the integration of benefit programs following mergers and acquisitions.	1	0
Prepare budgets for personnel operations.	1	0.03
Maintain records and compile statistical reports concerning personnel-related data such as hires, transfers, performance appraisals, and absenteeism rates.	1	0
Contract with vendors to provide employee services, such as food services, transportation, or relocation service.	1	0.38
Identify and implement benefits to increase the quality of life for employees, by working with brokers and researching benefits issues.	1	0
Plan, direct, supervise, and coordinate work activities of subordinates and staff relating to employment, compensation, labor relations, and employee relations.	1	0
Negotiate bargaining agreements.	1	0.67
Plan and conduct new employee orientations to foster positive attitude toward organizational objectives.	1	0
Conduct exit interviews to identify reasons for employee termination.	1	0
Develop methods to improve employment policies, processes, and practices, and recommend changes to management.	0.51	0
Formulate policies, procedures and programs for recruitment, testing, placement, classification, orientation, benefits and compensation, and labor and industrial relations.	0.23	0.01

Table 6.8: The tasks and their corresponding probability that they will be automated for *Compensation and Benefits Managers* according to our original linear program and the cross-validation.

Task Description	p	p'
Induce changes in composition of substances by introducing heat, light, energy, or chemical catalysts for quantitative or qualitative analysis.	0.68	0.82
Analyze organic or inorganic compounds to determine chemical or physical properties, composition, structure, relationships, or reactions, using chromatography, spectroscopy, or spectrophotometry techniques.	0.18	0.82
Maintain laboratory instruments to ensure proper working order and troubleshoot malfunctions when needed.	0.16	0.78
Conduct quality control tests.	0.07	0.54
Write technical papers or reports or prepare standards and specifications for processes, facilities, products, or tests.	0.03	0.03
Study effects of various methods of processing, preserving, or packaging on composition or properties of foods.	0	0.20
Prepare test solutions, compounds, or reagents for laboratory personnel to conduct tests.	0	0.63
Purchase laboratory supplies, such as chemicals, when supplies are low or near their expiration date.	0	1
Evaluate laboratory safety procedures to ensure compliance with standards or to make improvements as needed.	0	0
Direct, coordinate, or advise personnel in test procedures for analyzing components or physical properties of materials.	0	0.01
Develop, improve, or customize products, equipment, formulas, processes, or analytical methods.	0	0
Confer with scientists or engineers to conduct analyses of research projects, interpret test results, or develop nonstandard tests.	0	0.02

Table 6.9: The automation probability and the share of each task of a *chemist*.

Osborne. The job *First-Line supervisors of production and operating workers* has been assigned a 83% automation probability by us and only 1.6% by Frey and Osborne. A close inspection of the tasks makes us believe that the true value is between these extremes. Quite a few of the tasks are clearly automatable, e.g., "Keep records of employees' attendance and hours worked." and "Observe work and monitor gauges, dials, and other indicators to ensure that operators conform to production or processing standards." Others, e.g., "Read and analyze charts, work orders, production schedules, and other records and reports to determine production requirements and to evaluate current production estimates and outputs." seem difficult to automate. The complete results for this can job be found at `http://jobs-study.ethz.ch`.

We continue by comparing the previous results with the approach described in this section. To do this, we return to the jobs that we have looked at previously. First off is the job *chemists*. As shown in Table 6.9, the automation probability of most tasks has increased. Consequently, the automation probability of this job has increased from 10% to 42%. Due to the large difference, this job should be analyzed in-depth by job experts.

The changes in the automation probability of the tasks of *judges* are much smaller. Most tasks have a similar automation probability as before and the overall probability of this job has changed marginally, i.e., increased only from 40% to 50%. Therefore, we are confident that the classification by Frey and Osborne is correct.

We conclude that our approach can also be used to detect inconsistencies in the results of Frey and Osborne. We can then manually inspect the automation probabilities of the tasks of such an outlier to determine the truth. We think our

results allow us to fine tune the results from Frey and Osborne, but not replace it, as we need their results to bootstrap our linear program.

6.7 Further Analysis

In addition to inspecting every task of every job, we consider a broader picture. We do this by looking at general properties of a job that correlate with the probability that it can be automated.

6.7.1 Tasks

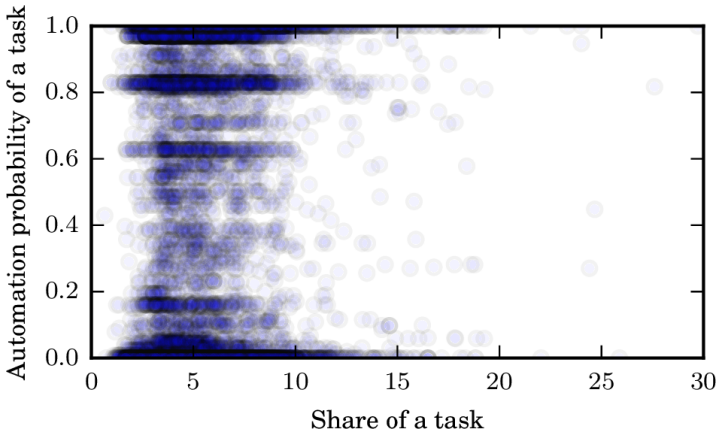


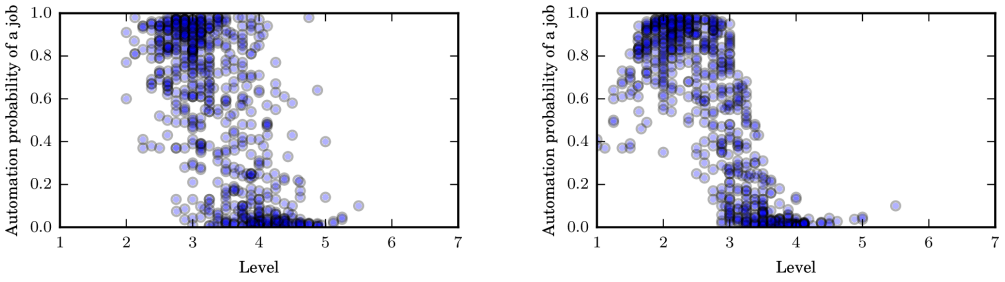
Figure 6.10: Our results show almost no correlation between the share of a task and its probability that it will be automated.

We first analyze the share of a task. The higher the share, the more often a task is performed. Hence, from a machine learning perspective this means that much more training data is available. This might lead to the conclusion that such a task is easier to automate. To disprove this claim, we plotted the share of a task over the probability that a task can be automated according to our linear program. The resulting graph is shown in Figure 6.10. Every dot represents one task, with its share on the x-axis and its probability on the y-axis. We see that there is barely any correlation between these two. We conclude that tasks that are done more frequently are not more likely to be automated.

6.7.2 Jobs

We continue our analysis by looking at the correlation between the properties that a job has, e.g., what kind of degree is necessary to do a job, and the probability that this job can be automated. Correlation does not imply causation, but nevertheless, these results reveal some interesting nuggets.

O*NET provides the level that the ability "deductive reasoning" is used in a job. The level ranges from 1 to 7, where for example level 2 means "knowing that a stalled car can coast downhill" and level 5 "deciding what factors to consider in selecting stocks". For every job, we have one value between 1 and 7. The



(a) Deductive Reasoning **(b)** Originality

Figure 6.11: The probability that a job can be automated over the level of the abilities "deductive reasoning" and "originality" used in this job. These levels are defined by O*NET and for each they provide an anchor point. Level 2 of "deductive reasoning" means "knowing that a stalled car can coast downhill" and level 5 "deciding what factors to consider in selecting stocks". Level 2 of "originality" means "using a credit card to open a locked door" and level 6 "inventing a new type of man-made fiber". Every point represents one job. A higher level of either of these two abilities correlates, as expected, with a lower probability to be automated.

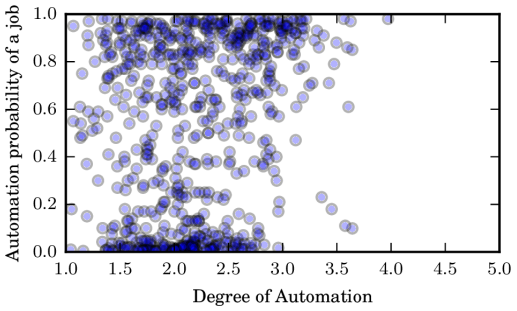
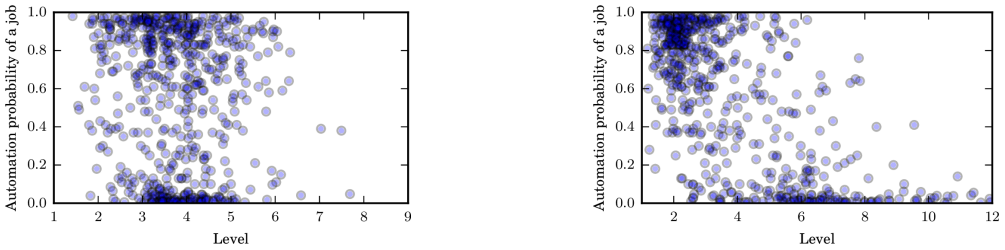


Figure 6.12: The probability that a job can be automated over the level of the current "Degree of Automation". This level ranges from 1 (Not at all automated) to 5 (Completely automated). The rather small correlation of 0.23 implies that different jobs will soon be affected.

resulting graph can be seen in Figure 6.11. Every job is represented by one dot; its x-coordinate being its level and the y-coordinate its probability. We can see that jobs that require a high level of deductive reasoning tend to have a lower probability of being automated. A similar result can be seen for "originality" (see Figure 6.11). Level 2 of "originality" means "using a credit card to open a locked door" and level 6 means "inventing a new type of man-made fiber". This confirms our expectation that these abilities will remain difficult for a computer.

O*NET even has an explicit value for the current level of the "degree of automation" for each job. This level ranges from 1 (not at all automated) to 5 (completely automated). As depicted in Figure 6.12, the already existing level of automation barely correlates with the probability that this job will be automated. This indicates that not only jobs that are already affected by automation are in danger, but also a whole new set of jobs. This is aligned with the recent worries about many new jobs soon being affected by computerization.



(a) On the Job Training **(b)** Required Level of Education

Figure 6.13: The probability that a job can be automated over the amount of "On the Job Training" (ranging from 1 (none or short demonstration) to 9 (over 10 years) and the "Required of Level of Education" (ranging from 1 (less than a high school diploma) to 12 (post-doctoral training)).

We conclude this section by looking at the effect that the level of required education for a job has on the probability to be automated. Jobs that require only very little education (level 1, i.e., less than a high school diploma) tend to have a higher probability than jobs that require an associate degree (level 5) which in turn have a higher probability than jobs that require post-doctoral training (level 12). Most jobs that require little education are in danger. It is noteworthy that the effect of training before the job is much stronger than the effect of on the job training. Jobs that require more on the job training only have a marginally smaller probability to be automated. Both plots are shown in Figure 6.13.

6.8 Conclusion

We believe that automation will cause a massive change in the job market and is one of the main challenges for society. In our opinion, the seminal work of Frey and Osborne did an excellent job of getting the discussion going. In this paper we dug a bit deeper, by looking not only at jobs – but at the tasks that make up a job. We hope that opening the Frey/Osborne black box will help the discussion. The professionals that are actually doing a job are the main experts to decide what parts of their job can or cannot be computerized. The Frey/Osborne work only tells these experts that their job is 87% automatable, but what does it actually mean? With our work, job experts can look inside the box, and understand which tasks of their job are at risk. Our hope is that the job experts have a discussion which results are believable and which are not, and why. To facilitate this discussion, we have created a web page (`http://jobs-study.ethz.ch`) that allows users to comment upon our results.

7

Conclusion

We have analyzed several problems. We proved that these problems are difficult if we allow adversarial input. Following the main thread of this thesis, we looked for shortcuts and tricks to deal with these problems.

Many heuristics exist that exploit that most users do not access their flash memory in a worst case manner. We show that if they did, then space overhead σ and write overhead ω would always have to obey $\sigma\omega \geq 1$. Our cycling algorithm performs well in case of adversarial access patterns and guarantees $\sigma\omega \leq 1$.

We proved that finding the best friends is essentially hopeless; no local algorithm can be as good as a global one and there is not even a best local algorithm. We provided some hope by proving that one can execute several algorithms in parallel and be at least half as good as the best of these algorithms. A more positive result was shown for personalized web filtering. Even though the original problem is $\mathcal{NP}$-hard, our heuristic performs very well; showing us that real world input may not be adversarial.

We also analyzed two types of markets. The first type of market is an online auction, where bidders have a valuation and a preemption price. Using our clairvoyant model, we were able to analyze this type of online auction. We have proved that an optimal and polynomial time algorithm exists to compute the difficulty Δ of the input. Last, but not least, we refined the results by Frey and Osborne by analyzing data from the O*NET database to predict which tasks of a job are susceptible to computerization.

Despite all the shortcuts we have presented for various problems, we want to point out that this is not always the way to go. Even if a simplified version of a song sounds nearly as good as the original, there is a difference. A few people will notice this and will greatly appreciate the time and effort that was spent practicing. Furthermore, the experiences had and skills gained while never giving up and continuing to work on a problem can be their own reward. A reward that

cannot be acquired without putting in the work. The side effects of the 10,000 hours of practice – incidentally also the number of hours I spent on my PhD – can be much more valuable than the initial goal.

Bibliography

[1] Abraham, D., Levavi, A., Manlove, D., O'Malley, G.: The stable roommates problem with globally-ranked pairs. In: WINE. Volume 4858 of Lecture Notes in Computer Science., Springer (December 2007) 431–444

[2] Agarwal, R., Marrow, M.: A closed-form expression for write amplification in NAND Flash. In: GLOBECOM Workshops. (2010) 1846–1850

[3] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M., Panigrahy, R.: Design Tradeoffs for SSD Performance. In: USENIX Annual Technical Conference on Annual Technical Conference. ATC (2008) 57–70

[4] Ajtai, M., Aspnes, J., Dwork, C., Waarts, O.: A theory of competitive analysis for distributed algorithms. In: FOCS, IEEE (1994) 401–411

[5] Andelman, N., Azar, Y., Sorani, M.: Truthful Approximation Mechanisms for Scheduling Selfish Related Machines. Theor. Comp. Sys. **40**(4) (June 2007) 423–436

[6] Arcaute, E., Vassilvitskii, S.: Social Networks and Stable Matchings in the Job Market. In: WINE. (2009) 220–231

[7] Ashwinkumar, B.V.: Buyback Problem - Approximate Matroid Intersection with Cancellation Costs. In: ICALP. (2011) 379–390

[8] Ashwinkumar, B.V., Kleinberg, R.: Randomized Online Algorithms for the Buyback Problem. In: WINE. (2009) 529–536

[9] Auli, M., Galley, M., Quirk, C., Zweig, G.: Joint language and translation modeling with recurrent neural networks. In: EMNLP. Volume 3. (2013) 0

[10] Autor, D.H., Dorn, D.: The growth of low skill service jobs and the polarization of the us labor market. Technical report, National Bureau of Economic Research (2009)

[11] Autor, D.H., Katz, L.F., Krueger, A.B.: Computing inequality: have computers changed the labor market? Technical report, National Bureau of Economic Research (1997)

[12] Autor, D.H., Levy, F., Murnane, R.J.: The skill content of recent technological change: An empirical exploration. Technical report, National Bureau of Economic Research (2001)

[13] Babaioff, M., Hartline, J.D., Kleinberg, R.: Selling ad campaigns: online algorithms with cancellations. In: EC. (2009) 61–70

[14] Babaioff, M., Lavi, R., Pavlov, E.: Mechanism design for single-value domains. In: AAAI. Volume 20. (2005) 241

[15] Belleflamme, P., Bloch, F.: Market Sharing Agreements and Collusive Networks. International Economic Review **45**(2) (2004) 387–411

[16] Ben-Aroya, A., Toledo, S.: Competitive Analysis of Flash Memory Algorithms. ACM Trans. Algorithms **7**(2) (March 2011) 23:1–23:37

[17] Bissig, P., Brandes, P., Wattenhofer, R., Willi, R.: Spoilers Ahead - Personalized Web Filtering. In: 4th International Workshop on Web Personalization, Recommender Systems and Social Media (WPRSM), Singapore. (December 2015)

[18] Bonin, H., Gregory, T., Zierahn, U.: Übertragung der Studie von Frey/Osborne (2013) auf Deutschland. Technical report, ZEW Kurzexpertise (2015)

[19] Bound, J., Johnson, G.E.: Changes in the structure of wages during the 1980's: An evaluation of alternative explanations. Technical report, National Bureau of Economic Research (1989)

[20] Boyd-Graber, J., Glasgow, K., Zajac, J.S.: Spoiler alert: Machine learning approaches to detect social media posts with revelatory information. In: Proceedings of the 76th ASIS&T Annual Meeting: Beyond the Cloud: Rethinking Information Boundaries. ASIST, Silver Springs, MD, USA, American Society for Information Science (2013)

[21] Brandes, P., Huang, Z., Su, H.H., Wattenhofer, R.: Clairvoyant Mechanisms for Online Auctions. In: 22nd Annual International Computing and Combinatorics Conference (COCOON), Ho Chi Minh City, Vietnam. (August 2016)

[22] Brandes, P., Wattenhofer, R.: On Finding Better Friends in Social Networks. In: 14th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Toronto, Canada. (October 2012)

[23] Brandes, P., Wattenhofer, R.: Space and Write Overhead are Inversely Proportional in Flash Memory. In: 8th ACM International Systems and Storage Conference (SYSTOR), Haifa, Israel. (May 2015)

[24] Brynjolfsson, E., McAfee, A.: Race against the machine: How the digital revolution is accelerating innovation, driving productivity, and irreversibly transforming employment and the economy. Brynjolfsson and McAfee (2012)

[25] Brynjolfsson, E., McAfee, A.: The second machine age: work, progress, and prosperity in a time of brilliant technologies. WW Norton & Company (2014)

[26] Calvó-Armengol, A., Jackson, M.O.: Networks in labor markets: Wage and employment dynamics and inequality. Journal of Economic Theory **132**(1) (2007) 27–46

[27] Canali, D., Cova, M., Vigna, G., Kruegel, C.: Prophiler: A fast filter for the large-scale detection of malicious web pages. In: Proceedings of the 20th International Conference on World Wide Web. WWW '11, New York, NY, USA, ACM (2011) 197–206

[28] Chang, L.P.: On Efficient Wear Leveling for Large-scale Flash-memory Storage Systems. In: Proceedings of the ACM Symposium on Applied Computing. SAC (2007) 1126–1130

[29] Chang, L.P., Chou, T.Y., Huang, L.C.: An Adaptive, Low-cost Wear-leveling Algorithm for Multichannel Solid-state Disks. ACM Trans. Embed. Comput. Syst. **13**(3) (December 2013) 55:1–55:26

[30] Chang, L.P., Du, C.D.: Design and Implementation of an Efficient Wear-leveling Algorithm for Solid-state-disk Microcontrollers. ACM Trans. Des. Autom. Electron. Syst. **15**(1) (December 2009) 6:1–6:36

[31] Chang, L.P., Huang, L.C.: A Low-cost Wear-leveling Algorithm for Block-mapping Solid-state Disks. In: Proceedings of the SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems. LCTES (2011) 31–40

[32] Charles, K.K., Hurst, E., Notowidigdo, M.J.: Manufacturing decline, housing booms, and non-employment. Technical report, National Bureau of Economic Research (2013)

[33] Ching, S.: Strategy-proofness and "median voters". International Journal of Game Theory **26**(4) (1997) 473–490

[34] Chrobak, M., Kenyon, C., Noga, J., Young, N.E.: Incremental medians via online bidding. Algorithmica **50**(4) (2008) 455–478

[35] Chung, T.S., Park, D.J., Park, S., Lee, D.H., Lee, S.W., Song, H.J.: System Software for Flash Memory: A Survey. In: Proceedings of the International Conference on Embedded and Ubiquitous Computing. EUC (2006) 394–404

[36] Clarke, E.H.: Multipart pricing of public goods. Public Choice **11**(1) (1971) 17–33

[37] Constantin, F., Feldman, J., Muthukrishnan, S., Pál, M.: An online mechanism for ad slot reservations with cancellations. In: SODA. (2009) 1265–1274

[38] Cramton, P., Shoham, Y., Steinberg, R.: Combinatorial auctions. (2006)

[39] Curtsinger, C., Livshits, B., Zorn, B., Seifert, C.: Zozzle: Fast and precise in-browser javascript malware detection. In: Proceedings of the 20th USENIX Conference on Security. SEC, Berkeley, CA, USA, USENIX Association (2011) 3–3

[40] D. Gale, Shapley, L.: College Admission and the Stability of Marriage. American Mathematical Monthly **69**(1) (1962) 9–15

[41] Danchev, D.: South korea to block port 25 as anti-spam countermeasure. In: http://www.zdnet.com/article/ south-korea-to-block-port-25-as-anti-spam-countermeasure/. (November 2011)

[42] Dasgupta, S., Long, P.M.: Performance Guarantees for Hierarchical Clustering. J. Comput. Syst. Sci. **70**(4) (June 2005) 555–569

[43] Deng, L., Yu, D., Hinton, G.: Deep learning for speech recognition and related applications. In: NIPS Workshop. (2009)

[44] Desnoyers, P.: Analytic Models of SSD Write Performance. Trans. Storage **10**(2) (March 2014) 8:1–8:25

[45] Diamantoudi, E., Miyagawa, E., Xue, L.: Random paths to stability in the roommate problem. Games and Economic Behavior **48**(1) (2004) 18–28

[46] Dietrich, C., Rossow, C.: Empirical research of ip blacklists. In Pohlmann, N., Reimer, H., Schneider, W., eds.: ISSE 2008 Securing Electronic Business Processes. Vieweg+Teubner (2009) 163–171

[47] Dunbar, R.: How Many Friends Does One Person Need?: Dunbar's Number and Other Evolutionary Quirks. Harvard University Press (2010)

[48] Echenique, F., Oviedo, J.: A theory of stability in many-to-many matching markets. Theoretical Economics **1**(2) (2006) 233–273

[49] Floréen, P., Kaski, P., Polishchuk, V., Suomela, J.: Almost Stable Matchings by Truncating the Gale-Shapley Algorithm. Algorithmica **58**(1) (2010) 102–118

[50] Ford, M.: Rise of the Robots: Technology and the Threat of a Jobless Future. Basic Books (2015)

[51] Frankie, T., Hughes, G., Kreutz-Delgado, K.: A Mathematical Model of the Trim Command in NAND-flash SSDs. In: Proceedings of the 50th Annual Southeast Regional Conference. ACM-SE (2012) 59–64

[52] Frey, C.B., Osborne, M.A.: The future of employment: How susceptible are jobs to computerisation? (2013)

[53] Friedman, E.J., Parkes, D.C.: Pricing WiFi at Starbucks: Issues in Online Mechanism Design. In: EC. (2003) 240–241

[54] Fung, S.P.Y.: Online Scheduling of Unit Length Jobs with Commitment and Penalties. In: TCS. (2014) 54–65

[55] Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)

[56] Gilbert, J., Mosteller, F.: Recognizing the Maximum of a Sequence. Journal of the American Statistical Association **61**(313) (1966) pp. 35–73

[57] Gladwell, M.: Outliers: The Story of Success. Little, Brown (2008)

[58] Golbeck, J.: The twitter mute button: A web filtering challenge. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI, New York, NY, USA, ACM (2012) 2755–2758

[59] Goos, M., Manning, A.: Lousy and lovely jobs: The rising polarization of work in britain. The review of economics and statistics **89**(1) (2007) 118–133

[60] Groves, T.: Incentives in teams. Econometrica: Journal of the Econometric Society (1973) 617–631

[61] Guizzo, E.: How google's self-driving car works. IEEE Spectrum Online, October **18** (2011)

[62] Guo, S., Ramakrishnan, N.: Finding the storyteller: Automatic spoiler tagging using linguistic cues. In Huang, C.R., Jurafsky, D., eds.: COLING, Tsinghua University Press (2010) 412–420

[63] Gusfield, D., Irving, R.W.: The Stable marriage problem - structure and algorithms. Foundations of computing series. MIT Press (1989)

[64] Hajiaghayi, M.T.: Online Auctions with Re-usable Goods. In: EC. (2005) 165–174

[65] Hajiaghayi, M.T., Kleinberg, R.D., Parkes, D.C.: Adaptive limited-supply online auctions. In: EC. (2004) 71–80

[66] Halldórsson, M.M., Irving, R.W., Iwama, K., Manlove, D.F., Miyazaki, S., Morita, Y., Scott, S.: Approximability results for stable marriage problems with ties. Theoretical Computer Science **306**(1-3) (2003) 431–447

[67] Han, X., Kawase, Y., Makino, K.: Online Knapsack Problem with Removal Cost. In: COCOON. (2012) 61–73

[68] Hartline, J., Sharp, A.: An Incremental Model for Combinatorial Maximization Problems. In: WEA. (2006) 36–48

[69] Hartline, J., Sharp, A.: Incremental flow. Networks **50**(1) (2007) 77–85

[70] Hastad, J.: Clique is Hard to Approximate Within $n^{1-\varepsilon}$. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science. FOCS, Washington, DC, USA, IEEE Computer Society (1996) 627–

[71] Hoefer, M.: Local Matching Dynamics in Social Networks. Automata Languages and Programming (2011) 113–124

[72] Hoefer, M., Wagner, L.: Locally stable marriage with strict preferences. In: Automata, Languages, and Programming. Springer (2013) 620–631

[73] Hu, X.Y., Eleftheriou, E., Haas, R., Iliadis, I., Pletka, R.: Write Amplification Analysis in Flash-based Solid State Drives. In: Proceedings of of SYSTOR: The Israeli Experimental Systems Conference. SYSTOR (2009) 10:1–10:9

[74] Iwama, K., Miyazaki, S., Manlove, D., Morita, Y.: Stable Marriage with Incomplete Lists and Ties. In: Proceedings of the 26th International Colloquium on Automata, Languages and Programming. ICALP, London, UK, Springer-Verlag (1999) 443–452

[75] Jackson, M.O. In: A Survey of Models of Network Formation: Stability and Efficiency. Cambridge University Press (2005) 1–62

[76] Kawase, Y., Han, X., Makino, K.: Unit Cost Buyback Problem. In: ISAAC. (2013) 435–445

[77] Kelso, A.S., Crawford, V.P.: Job Matching, Coalition Formation, and Gross Substitutes. Econometrica **50**(6) (1982) 1483–1504

[78] Keynes, J.M.: Economic possibilities for our grandchildren (1930). Essays in persuasion (1933) 358–73

[79] Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theoretical Computer Science **127** (May 1994) 255–267

[80] Kim, Y., Tauras, B., Gupta, A., Mihai, D., Urgaonkar, N.B.: FlashSim: A Simulator for NAND Flash-based Solid-State Drives (2009)

[81] Kothari, A., Parkes, D.C., Suri, S.: Approximately-strategyproof and tractable multiunit auctions. Decision Support Systems **39**(1) (2005) 105–121

[82] Kovács, A.: Fast Monotone 3-Approximation Algorithm for Scheduling Related Machines. In: ESA 2005. Volume 3669 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 616–627

[83] Kranton, R.E., Minehart, D.F.: A theory of buyer-seller networks. American Economic Review **91**(3) (2001) 485–508

[84] Lavi, R., Nisan, N.: Competitive Analysis of Incentive Compatible On-line Auctions. In: EC. (2000) 233–241

[85] Lavi, R., Nisan, N.: Online Ascending Auctions for Gradually Expiring Items. In: SODA. (2005) 1146–1155

[86] Le, Q.V.: Building high-level features using large scale unsupervised learning. In: ICASSP, IEEE (2013) 8595–8598

[87] Lee, J., Byun, E., Park, H., Choi, J., Lee, D., Noh, S.H.: CPS-SIM: Configurable and Accurate Clock Precision Solid State Drive Simulator. In: Proceedings of the ACM Symposium on Applied Computing. SAC '09 (2009) 318–325

[88] Lee, S.W., Park, D.J., Chung, T.S., Lee, D.H., Park, S., Song, H.J.: A Log Buffer-based Flash Translation Layer Using Fully-associative Sector Translation. ACM Trans. Embed. Comput. Syst. **6**(3) (July 2007)

[89] Lehmann, D., Oćallaghan, L., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. Journal of the ACM (JACM) **49**(5) (2002) 577–602

[90] Lindley, D.V.: Dynamic Programming and Decision Theory. j-APPL-STAT **10**(1) (1961) 39–51

[91] Manlove, D.F., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. Theoretical Computer Science **276**(1-2) (2002) 261–279

[92] Manyika, J., Chui, M., Bughin, J., Dobbs, R., Bisson, P., Marrs, A.: Disruptive technologies: Advances that will transform life, business, and the global economy. Volume 12. McKinsey Global Institute New York (2013)

[93] Mettu, R.R., Plaxton, C.G.: The Online Median Problem. In: FOCS. (2000) 339–348

[94] Micron: Wear-Leveling Techniques in NAND Flash Devices (2008)

[95] Micron: NAND Flash 101: An Introduction to NAND FLash and How to Design It In to Your Next Product (2010)

[96] Micron: Wear-Leveling in Micron NAND Flash Memory (2011)

[97] Milgrom, P.: Putting auction theory to work. Cambridge University Press (2004)

[98] Moulin, H.: On strategy-proofness and single peakedness. Public Choice **35**(4) (1980) 437–455

[99] Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, New York, NY, USA (2007)

[100] Ntoulas, A., Najork, M., Manasse, M., Fetterly, D.: Detecting spam web pages through content analysis. In: Proceedings of the 15th International Conference on World Wide Web. WWW, New York, NY, USA, ACM (2006) 83–92

[101] Pajarinen, M., Ekeland, A.: Computerization and the Future of Jobs in Norway. (2015)

[102] Parkes, D.C., Singh, S.P.: An MDP-Based Approach to Online Mechanism Design. In: NIPS. (2003)

[103] Parkes, D.C., Singh, S.P., Yanovsky, D.: Approximately Efficient Online Mechanism Design. In: NIPS. (2004)

[104] Plaxton, C.G.: Approximation Algorithms for Hierarchical Location Problems. In: STOC. (2003) 40–49

[105] Pomeranz, H.: A simple dns-based approach for blocking web advertising. (August 2013)

[106] Qureshi, M.K., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., Abali, B.: Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling. In: Proceedings of theAnnual IEEE/ACM International Symposium on Microarchitecture. MICRO 42 (2009) 14–23

[107] Roth, A.E., Sonmez, T., Unver, M.U.: Pairwise Kidney Exchange. Working Paper 10698, National Bureau of Economic Research (August 2004)

[108] Roth, A.E., Sotomayor, M.: Two-sided matching. Handbook of Game Theory vol 1 **78**(2) (1990) 75–95

[109] Rothkopf, M.H., Pekec, A., Harstad, R.M.: Computationally Manageable Combinatorial Auctions (1998)

[110] Sahami, M., Dumais, S., Heckerman, D., Horvitz, E.: A bayesian approach to filtering junk e-mail (1998)

[111] Sandholm, T.: Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence **135**(1) (2002) 1–54

[112] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529** (2016) 484–503

[113] Vazirani, V.V.: Approximation Algorithms. Springer (2004)

[114] Vickrey, W.: Counterspeculation, Auctions, and Competitive Sealed Tenders. The Journal of Finance **16**(1) (1961) 8–37

[115] Wood, A.: North-south trade, employment and inequality. New York (1994)

[116] Wood, A.: Globalisation and the rise in labour market inequalities. The economic journal **108**(450) (1998) 1463–1482

[117] Woodhouse, D.: JFFS : The Journalling Flash File System (2001)

[118] Xiang, L., Kurkoski, B.M.: An Improved Analytical Expression for Write Amplification in NAND Flash. CoRR (2011)

[119] Zuckerman, D.: Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. Theory of Computing **3**(6) (2007) 103–128

Curriculum Vitae

01 Aug 1986	Born in Peine, Germany
2006 – 2009	Studies for BSc in Computer Science, University of Paderborn, Germany
2009 – 2011	Studies for MSc in Computer Science, University of Paderborn, Germany
2011 – 2016	PhD student, research and teaching assistant, Distributed Computing Group, Prof. Dr. Roger Wattenhofer, ETH Zurich, Switzerland