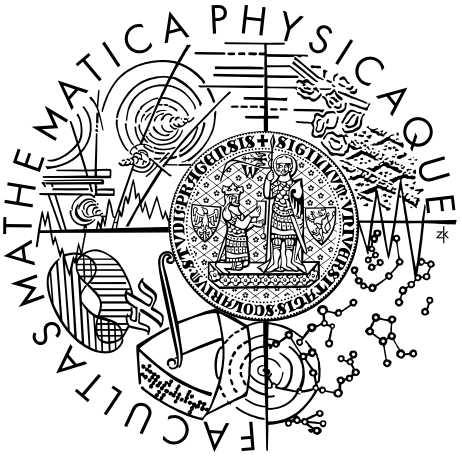
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Lukáš Folwarczný

On the Hardness of General Caching

Computer Science Institute of Charles University

Supervisor of the bachelor thesis: prof. RNDr. Jiří Sgall, DrSc.
Study programme: Computer Science
Study branch: General Computer Science

Prague 2015

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague, July 28, 2015 Lukáš Folwarczný

Název práce: O těžkosti obecného cachování

Autor: Lukáš Folwarczný

Ústav: Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: prof. RNDr. Jiří Sgall, DrSc., Informatický ústav Univerzity Karlovy

Abstrakt: *Cachování* (také známo jako *stránkování*) je klasický problém modelující obsluhu dvouúrovňových paměťových systémů. *Obecné cachování* je varianta se stránkami různých velikostí a cen. V práci se zabýváme zpřesněním charakterizace výpočetní složitosti obecného cachování v offline případě.

Nedávno bylo dokázáno, že obecné cachování v offline případě je silně NP-těžké, ovšem v důkazu byly zapotřebí instance cachování se stránkami většími nežli polovina velikosti cache. Náš hlavní výsledek se vyrovnává s tímto problémem: Dokazujeme, že obecné cachování je silně těžké již tehdy, když jsou velikosti stránek omezeny na $\{1, 2, 3\}$. Ve strukturální části práce pak představujeme nový jednodušší důkaz úplné charakterizace work functions pomocí struktury layers v případě klasického cachování, důkaz je následně rozšířen na cachování s proměnlivou velikostí cache. Na základě těchto výsledků jsme zkonstruovali dva algoritmy pro speciální případy obecného cachování.

Klíčová slova: cachování, obecné cachování, NP-těžkost, work function

Title: On the Hardness of General Caching

Author: Lukáš Folwarczný

Institute: Computer Science Institute of Charles University

Supervisor: prof. RNDr. Jiří Sgall, DrSc., Computer Science Institute of Charles University

Abstract: *Caching* (also known as *paging*) is a classical problem concerning page replacement policies in two-level memory systems. *General caching* is its variant with pages of different sizes and fault costs. We aim at a better characterization of the computational complexity of general caching in the offline version.

General caching in the offline version was recently shown to be strongly NP-hard, but the proof needed instances of caching with pages larger than half of the cache size. The primary result of this work addresses this problem as we prove: General caching is strongly NP-hard even when page sizes are limited to $\{1, 2, 3\}$. In the structural part of this work, a new simpler proof for the full characterization of work functions by layers for classical caching is given and then extended to caching with variable cache size. We invent two algorithms for restricted instances of general caching building on results around caching with variable cache size.

Keywords: caching, general caching, NP-hardness, work function

It was the 4^{th} of November 2013 when professor Sgall accidentally stole my pen I lent him at the lecture Approximation and Randomized Algorithms. When I went to the professor to get my pen back, I asked him which areas of research are worth exploring at the faculty. One thing led to another and I ended up working on a bachelor thesis under his supervision. I thank professor Sgall for bringing the topic of the thesis, a tremendous amount of time spent on discussing the problems with me, a lot of patience and a great amount of enthusiasm.

I wish to express my sincere gratitude to all the people who listened to my presentations of results from this thesis and raised valuable questions; these are participants of the Approximation and Online Algorithms Seminar, the SVOČ competition, the MAPSP workshop and the EURO Summer Institute on Online Optimization. I thank Sahar Bsaybes and anonymous reviewers for reading parts of the text and providing me with a useful feedback. My thanks also go to professor Chrobak for explaining the concept of layers to me. Last but not least, I thank my family and friends for their precious support during my studies.

Contents

Introduction

Caching (also known as *uniform caching* or *paging*) is a classical problem in the area of online algorithms and has been studied since 1960s. It models a two-level memory system: There is the fast memory of size k (the *cache*) and a slow but large main memory where all data reside. The problem instance comprises a sequence of requests, each demanding a page from the main memory. No cost is incurred if the requested page is present in the cache (a *cache hit*). If the requested page is not present in the cache (a *cache fault*), the page must be loaded at the fault cost of one; some page must be evicted to free space for the new one when there are already k pages in the cache. The natural objective is to evict pages in such a way that the total fault cost is minimized.

In 1990s, with the advent of World Wide Web, a generalized variant called *file caching* or simply *general caching* was studied [15, 23]. In this setting, each page p has its $\textsc{size}(p)$ and $\textsc{cost}(p)$. It costs $\textsc{cost}(p)$ to load this page into the cache and the page occupies $\textsc{size}(p)$ units of memory there. Uniform caching is the special case satisfying $\textsc{size}(p) = \textsc{cost}(p) = 1$ for every page p. Other important cases of this general model are

- the *cost model* (*weighted caching*): $\textsc{size}(p) = 1$ for every page p;
- the *bit model*: $\textsc{cost}(p) = \textsc{size}(p)$ for every page p;
- the *fault model*: $\textsc{cost}(p) = 1$ for every page p.

Caching, as described so far, requires the service to load the requested page when a fault occurs, which is known as caching under the *forced policy*. Allowing the service to pay the fault cost without actually loading the requested page to the cache gives another useful and studied variant of caching, the *optional policy*.

Offline version. The whole request sequence is known in advance in the offline version. In this case, uniform caching is solvable in polynomial time with a natural algorithm known as Belady's rule [8]. Caching in the cost model is a special case of the *k-server problem* and is also solvable in polynomial time [11]. In late 1990s, the questions about the complexity status of general caching were raised. This is the summary of the situation in the end of 1990s:

> “The hardness results for caching problems are very inconclusive. The NP-hardness result for the Bit model uses a reduction from PARTITION, which has pseudopolynomial algorithms. Thus a similar algorithm may well exist for the Bit model. We do not know whether computing the optimum in the Fault model is NP-hard.”
>
> — *Concluding remark of Albers et al. [3]*

There was no improvement until a breakthrough in 2010 when Chrobak et al. [13] showed that general caching is strongly NP-hard, already in the case of the fault model as well as in the case of the bit model. General caching is usually studied under the assumption that the largest page size is very small in comparison with the total cache size, as is for example the case of the aforementioned

article by Albers et al. [3]. Instances of caching with pages larger than half of the cache size (so called obstacles) are required in the proof given by Chrobak et al. Therefore, this hardness result is in fact still quite inconclusive.

Contributions I. We give a novel proof of strong NP-hardness for general caching which gives the first hardness result restricted to small pages: General caching is strongly NP-hard even in the case when the page sizes are limited to $\{1, 2, 3\}$, for both the fault model and the bit model, and under both the forced policy and the optional policy.

The proof of the result for general costs (and sizes $\{1, 2, 3\}$) is rather simple, in particular significantly simpler than the one given by Chrobak et al. [13]. The reductions for the result in the fault and bit models are significantly more involved and require a non-trivial potential-function-like argument. This part is covered in Chapter 1. We note that the aforementioned simpler proof (working only for general caching, not for the bit model or the fault model) was invented by the advisor prof. Sgall without the contribution of the primary author.

Online version. In the online version, a caching algorithm has to serve each of the requests before receiving the future requests. It is impossible to always produce an optimal sevice in this version. To measure the performance of an online algorithm, we consider the *competitive analysis*. A deterministic algorithm A is said to be *c-competitive* if there is a constant b such for each request sequence ρ

$$\mathrm{A}(\rho) \leq c\mathrm{OPT}(\rho) + b,$$

where $\mathrm{A}(\rho)$ denotes the cost of the service produced by the algorithm A and $\mathrm{OPT}(\rho)$ the optimum service cost for the sequence ρ. In the case of a randomized algorithm, the expected value of the service cost is used instead of $\mathrm{A}(\rho)$. The *competitive ratio* of an algorithm is the smallest c such that the algorithm is c-competitive.

It has been proven that no deterministic algorithm for uniform caching can be better than k-competitive (k is the cache size) [22]. Natural algorithms like LRU and FIFO match this bound. A k-competitive algorithm for general caching called LandLord with the competitive ratio k was given by Young [23].

We restrict the first part of the discussion about randomized algorithms to uniform caching. Fiat et al. [14] proved a lower bound of H_k (the kth harmonic number) on the competitive ratio, designed a simple marking algorithm Mark and proved that this algorithm is $2H_k$-competitive. The competitive ratio of Mark was later determined to be $2H_k - 1$ by Achlioptas et al. [1].

The first H_k-competitive randomized algorithm for uniform caching was given by McGeoch and Sleator [19] and was named Partition. The memory requirements of Partition cannot be bounded by a function of k. Achlioptas et al. [1] addressed this problem and invented an algorithm Equitable using $\mathcal{O}(k^2 \log k)$ memory. This was later improved to $\mathcal{O}(k)$ by Bein et al. [7] with a modification of Equitable called Equitable2. Negoescu et al. proposed a modification of Partition, the algorithm Partition2, which needs to remember only $\mathcal{O}(k/\log k)$ pages except for the pages in the cache.

The running time of Equitable is $\mathcal{O}(k^2)$ per request. The fastest known H_k-competitive algorithm is OnlineMin proposed by Negoescu et al. [20, 10]

with the running time $\mathcal{O}(\log k)$ per request (or $\mathcal{O}(\log k / \log\log k)$ depending on the implementation and computational model).

The first randomized algorithms beyond uniform caching were invented by Irani [15] who gave $\mathcal{O}(\log^2 k)$-competitive algorithms for caching in the bit model and caching in the fault model. This was later improved by Bansal et al. [4, 5]: They gave $\mathcal{O}(\log k)$-competitive algorithms for each of the cost, bit and fault models and an $\mathcal{O}(\log^2 k)$-competitive algorithm for general caching; these algorithms are based on rounding the solution of a linear program. Adamaszek et al. [2] invented a better rounding method which made them able to give an $\mathcal{O}(\log k)$-competitive algorithm for general caching. As far as lower bounds are concerned, Chrobak et al. [12] proved a lower bound on the competitive ratio of $1 + e^{-1/2}$ for caching in the cost model with $k = 2$. Because $1 + e^{-1/2} \approx 1.6065 > 1.5 = H_2$, a H_k-competitive algorithm (as in the case of uniform caching) is not possible for general caching.

Contributions II. The algorithm PARTITION uses a dynamically changing partition of the request sequence which enables the online algorithm to keep track about possible moves of the optimal algorithm. Koutsoupias and Papadimitriou [17] extended this result and showed that a similar structure may be used to fully (and efficiently) characterize the whole work function associated with the request sequence; this structure is now usually called *layers*. All the algorithms EQUITABLE, EQUITABLE2 and ONLINEMIN are based on this structure. The original proof (and the only one we are aware of in the literature) uses the quasi-convexity lemma. We provide a new simpler and more direct proof. An introduction into work functions and layers as well as the new proof are given in Chapter 2.

Caching with variable cache size. Allowing the cache size to change over time is a natural generalization of caching. To give an example, the best known approximation for general caching is an 4-approximation given by Bar-Noy et al. [6]. Their approximation works for this case of variable cache size as well. The topic of caching with variable cache size was pioneered by Peserico [21].

Contributions III. In Chapter 3, we show how to generalize our new proof from the previous chapter and derive the full characterization of work functions through layers for uniform caching with variable cache size. As a consequence, we show that layers and Belady's rule work for the optional policy as well because the optional policy may be easily simulated using variable cache size.

Solvable instances of caching. Despite the NP-hardness of general caching, it is still possible to solve some restricted instances in polynomial time. To do that, we define two parameters characterizing the instance. We call a page *normal*, if both its size and fault cost are one, and *abnormal* otherwise. For an instance of caching we define the parameter η to be the number of requests on abnormal pages. The second parameter κ is defined to be the difference between the cache size and the size of the smallest abnormal page.

Contributions IV. In the final chapter, we reap the harvest of understanding uniform caching with variable cache size and give two algorithms for general caching with the time complexity bounded by a function of the total number of requests n and the parameters η and κ. We show that general caching is solvable in time $\mathcal{O}(2^{\eta} \cdot n \log n)$, i.e., it is polynomially solvable when the total number of requests on abnormal pages is bounded by $\mathcal{O}(\log n)$. The second algorithm solves general caching in time $n^{\mathcal{O}(\kappa)}$, i.e., general caching is solvable in polynomial time if there is a constant c such that each page is either normal or has size at least $k - c$.

1. Strong NP-hardness

In this chapter we prove the following: General caching is strongly NP-hard even in the case when the page sizes are limited to $\{1, 2, 3\}$, both for the fault model and the bit model, and under both the forced policy and the optional policy.

The main part of our work – a polynomial-time reduction from independent set to caching in the fault model under the optional policy with page sizes restricted to {1, 2, 3} is explained in Subchapter 1.1 and its validity is proven in Subchapter 1.2. In Subchapter 1.3, we show how to modify the reduction so it works for the bit model as well. In Subchapter 1.4, we show how to obtain the hardness results also for the forced policy. Finally, we give a self-contained presentation of the simple proof of strong NP-hardness for general costs (in fact, only two different and polynomial costs are needed) in Subchapter 1.5.

1.1 Reduction

The decision problem IndependentSet is well-known to be NP-complete. By 3Caching(forced) and 3Caching(optional) we denote the decision versions of caching under each policy with page sizes restricted to $\{1, 2, 3\}$.

Problem:	IndependentSet
Instance:	A graph G and a number K.
Question:	Is there an independent set of cardinality K in G?

Problem:	3Caching(*policy*)
Instance:	A universe of pages, a sequence of page requests, numbers k and L. For each page p it holds size$(p) \in \{1, 2, 3\}$.
Question:	Is there a service under the policy *policy* of the request sequence using the cache of size k with a total fault cost of at most L?

We define 3Caching(fault,*policy*) to be the problem 3Caching(*policy*) with the additional requirement that page costs adhere to the fault model. The problem 3Caching(bit,*policy*) is defined analogously.

In this subchapter, we describe a polynomial-time reduction from IndependentSet to 3Caching(fault,optional). Informally, a set of pages of size two and three is associated with each edge and a page of size one is associated with each vertex. Each vertex-page is requested only twice while there are many requests on pages associated with edges. The request sequence is designed in such a way that the number of vertex-pages that are cached between the two requests in the optimal service is equal to the the size of the maximum independent set.

We now show the request sequence of caching corresponding to the graph given in IndependentSet with a parameter H. In the next subchapter, we prove that it is possible to set a proper value of H and a proper fault cost limit L such that the reduction becomes a valid polynomial-time reduction.

Reduction 1.1. *Let $G = (V, E)$ be the instance of* IndependentSet. *The graph G has n vertices and m edges and there is an arbitrary fixed order of edges*

$e_1, \ldots, e_m$. *Let H be a parameter bounded by a polynomial function of n.*

A corresponding instance $\mathcal{I}_G$ of 3Caching(fault,optional) *is an instance with the cache size $k = 2mH + 1$ and the total of $6mH + n$ pages. The structure of the pages and the requests sequence is described below.*

Pages. For each vertex v, we have a vertex-page p_v of size one. For each edge e, there are $6H$ edge-pages associated with it that are divided into H groups. The ith group consists of six pages $\bar{a}_i^e, \alpha_i^e, a_i^e, b_i^e, \beta_i^e, \bar{b}_i^e$ where pages α_i^e and β_i^e have size three and the remaining four pages have size two.

For a fixed edge e, let $\bar{a}^e$-pages be all pages $\bar{a}_i^e$ for $i = 1, \ldots, H$. Let also $\bar{a}$-pages be all $\bar{a}^e$-pages for $e = e_1, \ldots, e_m$. The remaining collections of pages (α^e-pages, α-pages, ...) are defined in a similar fashion.

Request sequence. The request sequence of $\mathcal{I}_G$ is organized in phases and blocks. There is one phase for each vertex $v \in V$, we call such a phase the v-phase. There are exactly two requests on each vertex-page p_v, one just before the beginning of the v-phase and one just after the end of the v-phase; these requests do not belong to any phase. The order of phases is arbitrary. In each v-phase, there are $2H$ adjacent blocks associated with every edge e incident with v; the blocks for different incident edges are ordered arbitrarily. In addition, there is one initial block I before all phases and one final block F after all phases. Altogether, there are $d = 4mH + 2$ blocks.

Let $e = \{u, v\}$ be an edge, let us assume that the u-phase precedes the v-phase. The blocks associated with e in the u-phase are denoted by $B^e_{1,1}$, $B^e_{1,2}, \ldots, B^e_{i,1}$, $B^e_{i,2}, \ldots, B^e_{H,1}$, $B^e_{H,2}$, in this order, and the blocks in the v-phase are denoted by $B^e_{1,3}$, $B^e_{1,4}, \ldots, B^e_{i,3}$, $B^e_{i,4}, \ldots, B^e_{H,3}$, $B^e_{H,4}$, in this order. An example is given in Figure 1.1.

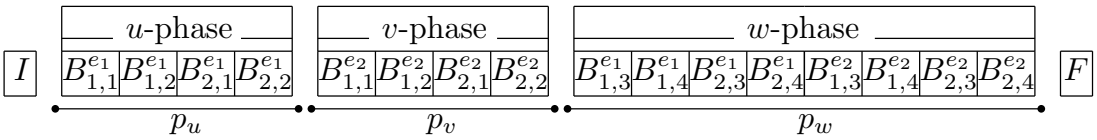


Figure 1.1: An example of phases, blocks and requests on vertex-pages for a graph with three vertices u, v, w and two edges $e_1 = \{u, w\}$, $e_2 = \{v, w\}$ when $H = 2$

Even though each block is associated with some fixed edge, it contains one or more requests to the associated pages for every edge e. In each block, we process the edges in the order $e_1, \ldots, e_m$ that was fixed above. Pages associated with the edge e are requested in two rounds. In each round, we process groups $1, \ldots, H$ in this order. When processing the ith group of the edge e, we request one or more pages of this group, depending on the block we are in. Table 1.1 determines which pages are requested.

Reduction 1.1 is now complete. An example of requests on edge-pages associated with one edge e is depicted in Figure 1.2. Notice that the order of the pages associated with e is the same in all blocks; more precisely, in each block the requests on the pages associated with e form a subsequence of

$$\bar{a}_1^e\, \alpha_1^e\, a_1^e\, \ldots\, \bar{a}_i^e\, \alpha_i^e\, a_i^e\, \ldots\, \bar{a}_H^e\, \alpha_H^e\, a_H^e\, b_1^e\, \beta_1^e\, \bar{b}_1^e\, \ldots\, b_i^e\, \beta_i^e\, \bar{b}_i^e\, \ldots\, b_H^e\, \beta_H^e\, \bar{b}_H^e. \tag{1.1}$$

Table 1.1: Requests associated with an edge e

Block	First round	•	Second round
before $B^e_{i,1}$	$\bar{a}^e_i$	•	
$B^e_{i,1}$	$\bar{a}^e_i$, α^e_i	•	b^e_i
$B^e_{i,2}$	α^e_i, a^e_i	•	b^e_i
between $B^e_{i,2}$ and $B^e_{i,3}$	a^e_i	•	b^e_i
$B^e_{i,3}$	a^e_i	•	b^e_i, β^e_i
$B^e_{i,4}$	a^e_i	•	β^e_i, $\bar{b}^e_i$
after $B^e_{i,4}$		•	$\bar{b}^e_i$

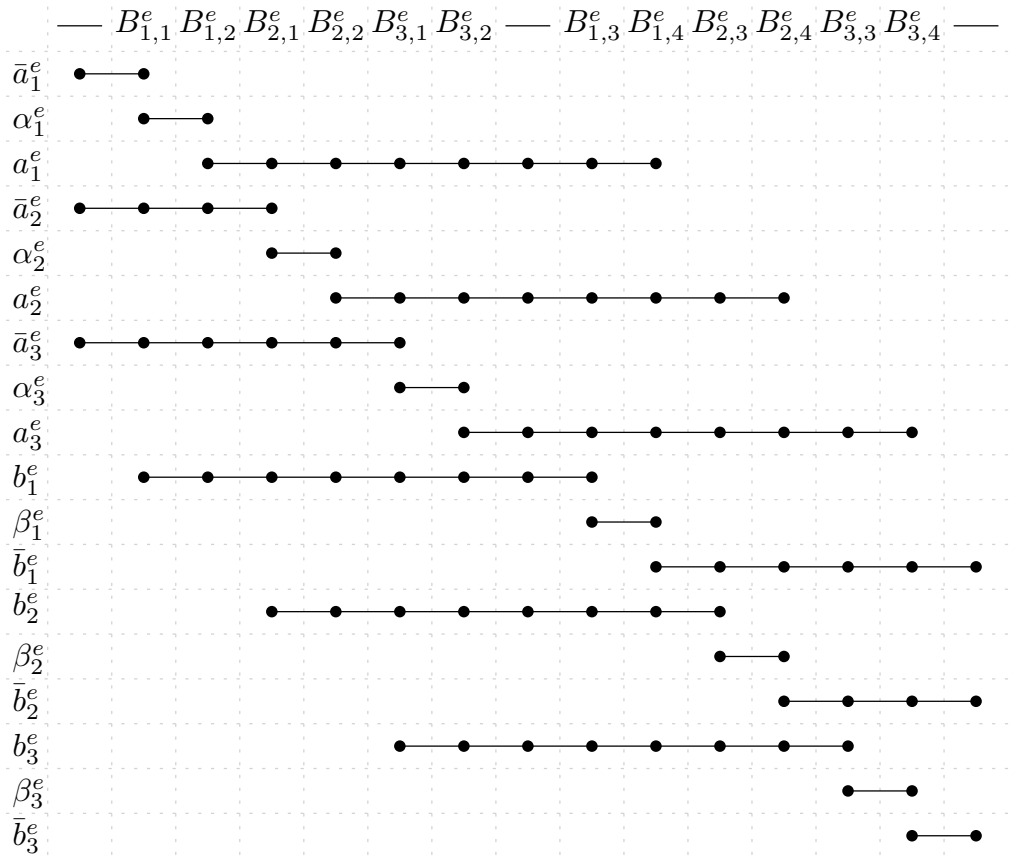


Figure 1.2: Requests on all pages associated with the edge e when $H = 3$. Each column represents some block(s). The labelled columns represent the blocks in the heading, the first column represents every block before $B^e_{1,1}$, the middle column represents every block between $B^e_{3,2}$ and $B^e_{1,3}$, and the last column represents every block after $B^e_{3,4}$. The requests in one column are ordered from top to bottom.

Preliminaries for the proof. Instead of minimizing the service cost, we maximize the savings compared to the service which does not use the cache at all. This is clearly equivalent when considering the decision versions of the problems.

Without loss of generality, we assume that any page is brought into the cache only immediately before some request to that page and removed from the cache only after some (possibly different) request to that page; furthermore, the cache is empty at the beginning and at the end. That is, a page may be in the cache only between two consecutive requests to this page, and either it is in the cache for the whole interval or not at all.

Each page of size three is requested only twice in two consecutive blocks, and these blocks are distinct for all pages of size three. Thus, a service of edge-pages is valid if and only if at each time, at most mH edge-pages are in the cache. It is convenient to think of the cache as of mH *slots* for edge-pages.

As each vertex-page is requested twice, the savings on the n vertex-pages are at most n. Furthermore, a vertex-page can be cached if and only if during the phase it never happens that at the same time all slots for edge-pages are full and a page of size three is cached.

Let S_B denote the set of all edge-pages cached at the beginning of the block B and let S_B^e be the set of pages in S_B associated with the edge e. We use $s_B = |S_B|$ and $s_B^e = |S_B^e|$ for the sizes of the sets. Each edge-page is requested only in a contiguous segment of blocks, once in each block. It follows that the total savings on edge-pages are equal to $\sum_B s_B$ where the sum is over all blocks. In particular, the maximal possible savings on the edge-pages are $(d-1)mH$, using the fact that S_I is empty. We shall show that the maximum savings are $(d-1)mH + K$ where K is the size of the maximum independent set in G.

Almost-fault model. To understand the reduction, we consider what happens if we relax the requirements of the fault model and set the cost of each vertex-page to $1/(n+1)$ instead of 1 as required by the fault model.

In this scenario, the total savings on vertex-pages are $n/(n+1) < 1$ which is less than savings incurred by one edge-page. Therefore, edge-pages must be served optimally in the optimal service of the whole request sequence.

In this case, the reduction works already for $H = 1$. This leads to a quite short proof of the strong NP-hardness for general caching and we give this proof in Subchapter 1.5. Here, we show just the main ideas that are important also for the design of our caching instance in the fault and bit models.

We first prove that for each edge e and each block $B \neq I$ we have $s_B^e = 1$ (see Subchapter 1.5). Using this we show below that for each edge e, at least one of the pages α_1^e and β_1^e is cached between its two requests. This implies that the set of all vertices v such that p_v is cached between its two requests is independent.

For a contradiction, let us assume that for some edge e, neither of the pages α_1^e and β_1^e is cached between its two requests. Because pages α_1^e and β_1^e are forbidden, there is b_1^e in $S_{B_{1,2}^e}$ and a_1^e in $S_{B_{1,3}^e}$. Somewhere between these two blocks $B_{1,2}^e$ and $B_{1,3}^e$, we must switch from caching b_1^e to caching a_1^e. However, this is impossible, because the order of requests implies that we would have to cache both b_1^e and a_1^e at some moment (see Figure 1.3). However, there is no place in the cache for such an operation, as $s_B^{e'} = 1$ for every e' and $B \neq I$.

In the fault model, the corresponding claim $s_B^e = H$ does not hold. Instead, we prove that the value of s_B^e cannot change much during the service and when

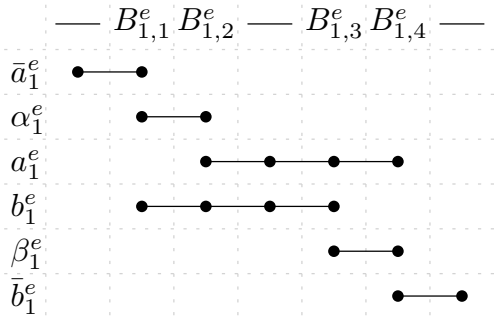


Figure 1.3: Pages associated with one edge when $H = 1$

we use H large enough, we still get a working reduction.

1.2 Proof of Correctness

In this subchapter, we show that the reduction described in the previous subchapter is indeed a reduction from IndependentSet set to 3Caching(fault,optional). We prove that there is an independent set of cardinality K in G if and only if there is a service of the caching instance $\mathcal{I}_G$ with the total savings of at least $(d-1)mH + K$. First the easy direction, which holds for any value of the parameter H.

Lemma 1.2. *Let G be a graph and $\mathcal{I}_G$ the corresponding caching instance from Reduction 1.1. Suppose that there is an independent set W of cardinality K in G. Then there exists a service of $\mathcal{I}_G$ with the total savings of at least $(d-1)mH + K$.*

Proof. For any edge e, denote $e = \{u, v\}$ so that the u-phase precedes the v-phase. If $u \in W$, we keep all $\bar{a}^e$-pages, b^e-pages, β^e-pages and $\bar{b}^e$-pages in the cache from the first to the last request on each page, but we do not cache a^e-pages and α^e-pages at any time. Otherwise, we cache all $\bar{a}^e$-pages, α^e-pages, a^e-pages and $\bar{b}^e$-pages, but do not cache b^e-pages and β^e-pages at any time. Figure 1.4 shows these two cases for the first group of pages. In both cases, at each time at most one page associated with each group of each edge is in the cache and the savings on those pages are $(d-1)mH$. We know that the pages fit in the cache because of the observations made in Subchapter 1.1.

For any $v \in W$, we cache p_v between its two requests. To check that this is a valid service, observe that if $v \in W$, then during the corresponding phase no page of size three is cached. Thus, the page p_v always fits in the cache together with at most mH pages of size two. □

We prove the converse in a sequence of lemmata. In Subchapter 1.3, we will show how to reuse the proof for the bit model. To be able to do that, we list explicitly all the assumptions about the caching instance that are used in the following proofs.

Properties 1.3. *Let $\mathcal{T}_G$ be an instance of general caching which corresponds to a graph $G = (V, E)$ with n vertices, m edges $e_1, \ldots, e_m$, the same cache size and the same universe of pages as in Reduction 1.1. The request sequence is again*

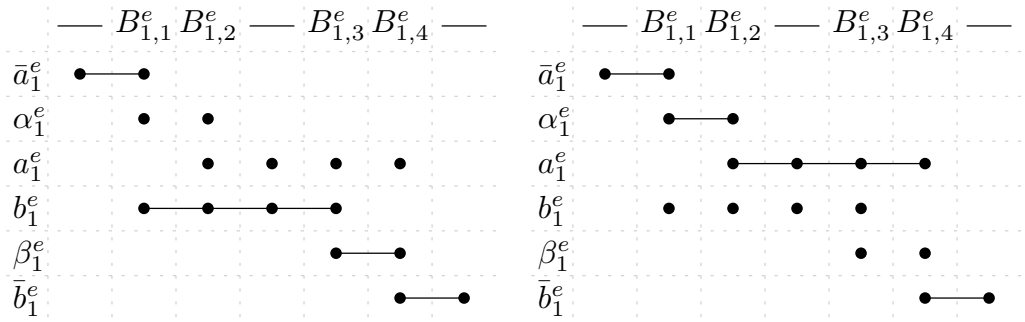


Figure 1.4: The two ways of caching in Lemma 1.2

split into phases, one phase for each vertex. Each phase is again partitioned into blocks, there is one initial block I before all phases and one final block F after all phases. There is the total of d blocks.

The instance $\mathcal{T}_G$ is required to fulfill the following list of properties:

(a) Each vertex page p_v is requested exactly twice, right before the v-phase and right after the v-phase.

(b) The total savings incurred on edge-pages are equal to $\sum s_B$ (summing over all blocks).

(c) For each edge e, there are exactly H pages associated with e requested in I, all the $\bar{a}^e$-pages, and exactly H pages associated with e requested in F, all the $\bar{b}^e$-pages.

(d) In each block, pages associated with e_1 are requested first, then pages associated with e_2 are requested and so on up to e_m.

(e) For each block B and each edge e, all requests on a^e-pages and $\bar{b}^e$-pages in B precede all requests on $\bar{a}^e$-pages and b^e-pages in B.

(f) Let $e = \{u, v\}$ be an edge and p an α^e-page or β^e-page. Let B be the first block and $\overline{B}$ the last block where p is requested. Then B and $\overline{B}$ are either both in the u-phase or both in the v-phase. Furthermore, no other page of size three associated with e is requested in B, $\overline{B}$, or any block between them.

Lemma 1.4. *The instance from Reduction 1.1 satisfies Properties 1.3.*

Proof. All properties (a), (b), (c), (d), (f) follow directly from Reduction 1.1 and the subsequent observations. To prove (e), recall that the pages associated with an edge e requested in a particular block always follow the ordering (1.1). We need to verify that when the page a^e_i is requested, no page $\bar{a}^e_j$ for $j \leq i$ is requested and that when the page $\bar{b}^e_i$ is requested, no $\bar{a}^e$-page and no page b^e_j for $j \leq i$ is requested. This can be seen easily when we explicitly write down the request sequences for each kind of block, see Table 1.2. □

For the following claims, let $\mathcal{T}_G$ be an instance fulfilling Properties 1.3. We fix a service of $\mathcal{T}_G$ with the total savings of at least $(d-1)mH$.

Let $\mathcal{B}$ be the set of all blocks and $\overline{\mathcal{B}}$ the set of all blocks except for the initial and final one. For a block B, we denote the block immediately following it by B'.

Table 1.2: Request sequences on all pages associated with an edge e

Block	First round • Second round
before $B^e_{1,1}$	$\bar{a}^e_1 \dots \bar{a}^e_H$ •
$B^e_{i,1}$	$a^e_1 \dots a^e_{i-1}\ \bar{a}^e_i\ \alpha^e_i\ \bar{a}^e_{i+1}\bar{a}^e_{i+2} \dots \bar{a}^e_H$ • $b^e_1 \dots b^e_i$
$B^e_{i,2}$	$a^e_1 \dots a^e_{i-1}\ \alpha^e_i\ a^e_i\ \bar{a}^e_{i+1}\bar{a}^e_{i+2} \dots \bar{a}^e_H$ • $b^e_1 \dots b^e_i$
between $B^e_{H,2}$ and $B^e_{1,3}$	$a^e_1 \dots a^e_H$ • $b^e_1 \dots b^e_H$
$B^e_{i,3}$	$a^e_i \dots a^e_H$ • $\bar{b}^e_1 \dots \bar{b}^e_{i-1}\ b^e_i\ \beta^e_i\ b^e_{i+1}b^e_{i+2} \dots b^e_H$
$B^e_{i,4}$	$a^e_i \dots a^e_H$ • $\bar{b}^e_1 \dots \bar{b}^e_{i-1}\ \beta^e_i\ \bar{b}^e_i\ b^e_{i+1}b^e_{i+2} \dots b^e_H$
after $B^e_{H,4}$	• $\bar{b}^e_1 \dots \bar{b}^e_H$

We define two useful values characterizing the service for the block B: $\delta_B = mH - s_B$ (the number of free slots for edge-pages at the start of the service of the block) and $\gamma^e_B = |s^e_{B'} - s^e_B|$ (the change of the number of slots occupied by pages associated with e after requests from this block are served).

The first easy lemma says that only a small number of blocks can start with some free slots in the cache.

Lemma 1.5. *When summing over all blocks except for the initial one*

$$\sum_{B \in \mathcal{B} \setminus \{I\}} \delta_B \leq n.$$

Proof. Using the property (b) and $s_I = 0$, the savings on edge-pages are

$$\sum_{B \in \mathcal{B} \setminus \{I\}} s_B = (d-1)mH - \sum_{B \in \mathcal{B} \setminus \{I\}} \delta_B.$$

The total savings are assumed to be at least $(d-1)mH$. Due to the property (a), the savings on vertex-pages are at most n. Claim of the lemma follows. □

The second lemma states that the number of slots occupied by pages associated with a given edge does not change much during the whole service.

Lemma 1.6. *For each edge $e \in E$,*

$$\sum_{B \in \overline{\mathcal{B}}} \gamma^e_B \leq 6n.$$

Proof. Let us use the notation $S^{\leq k}_{\overline{B}} = S^{e_1}_B \cup \dots \cup S^{e_k}_B$ and $s^{\leq k}_{\overline{B}} = \left|S^{\leq k}_{\overline{B}}\right|$. First, we shall prove for each $k \leq m$

$$\sum_{B \in \overline{\mathcal{B}}} \left| s^{\leq k}_{\overline{B'}} - s^{\leq k}_{\overline{B}} \right| \leq 3n. \tag{1.2}$$

Let $\mathcal{P}$ denote the set of all blocks B from $\overline{\mathcal{B}}$ satisfying $s_{B'}^{\leq k} - s_B^{\leq k} \geq 0$ and let $\mathcal{N}$ denote the set of all the remaining blocks from $\overline{\mathcal{B}}$.

As a consequence of the property (c), we get $s_{I'}^{\leq k} \in [kH - \delta_{I'}, kH]$ and $s_F^{\leq k} \in [kH - \delta_F, kH]$. So we obtain the inequality

$$s_F^{\leq k} - s_{I'}^{\leq k} \geq -\delta_F. \tag{1.3}$$

We claim $s_{B'}^{\leq k} - s_B^{\leq k} \leq \delta_B$ for each $B \in \overline{\mathcal{B}}$. We assume for a contradiction $s_{B'}^{\leq k} - s_B^{\leq k} > \delta_B$ for some block B. We use the property (d). Then after processing the edge e_k in B, the number of edge-pages in the cache is $(s_B - s_B^{\leq k}) + s_{B'}^{\leq k} > s_B + \delta_B = mH$. But more than mH edge-pages in the cache means a contradiction.

The summation over all blocks from $\mathcal{P}$ gives us the first bound

$$\sum_{B \in \mathcal{P}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) \leq \sum_{B \in \mathcal{P}} \delta_B \leq n. \tag{1.4}$$

Using the fact $\mathcal{P} \dot{\cup} \mathcal{N} = \overline{\mathcal{B}}$ and (1.3), we have

$$\sum_{B \in \mathcal{P}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) + \sum_{B \in \mathcal{N}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) = s_F^{\leq k} - s_{I'}^{\leq k} \geq -\delta_F;$$

together with (1.4), we obtain the second bound

$$-\sum_{B \in \mathcal{N}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) \leq \sum_{B \in \mathcal{P}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) + \delta_F \leq 2n. \tag{1.5}$$

Combining the bounds (1.4) and (1.5), we prove (1.2)

$$\sum_{B \in \overline{\mathcal{B}}} \left| s_{B'}^{\leq k} - s_B^{\leq k} \right| = \sum_{B \in \mathcal{P}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) - \sum_{B \in \mathcal{N}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) \leq n + 2n = 3n.$$

For the edge e_1, the claim of this lemma is weaker than (1.2) because $\gamma_B^{e_1} = |s_{B'}^{e_1} - s_B^{e_1}|$. Proving our lemma for e_k when $k > 1$ is just a matter of using (1.2) for $k-1$ and k together with the formula $|x - y| \leq |x| + |y|$:

$$\sum_{B \in \overline{\mathcal{B}}} \gamma_B^{e_k} \leq \sum_{B \in \overline{\mathcal{B}}} \left| s_{B'}^{\leq k} - s_B^{\leq k} \right| + \sum_{B \in \overline{\mathcal{B}}} \left| s_{B'}^{\leq k-1} - s_B^{\leq k-1} \right| \leq 3n + 3n = 6n$$

□

For the rest of the proof, we set $H = 6mn + 3n + 1$. This enables us to show that the fixed service must cache some of the pages of size three.

Lemma 1.7. *For each edge $e \in E$, there is a block B such that some α^e-page or β^e-page is in S_B and $\delta_B = 0$.*

Proof. Fix an edge $e = e_k$. For each block B, we define

$$\varepsilon_B = \text{number of } \alpha^e\text{-pages and } \beta^e\text{-pages in } S_B.$$

Observe that due to the property (f), ε_B is always one or zero. We use a potential function

$$\Phi_B = \text{number of } a^e\text{-pages and } \bar{b}^e\text{-pages in } S_B.$$

Because there are only $\bar{a}$-pages in the initial block and only $\bar{b}$-pages in the final block (property (c)), we know

$$\Phi_{I'} = 0 \quad \text{and} \quad \Phi_F \geq H - \delta_F. \tag{1.6}$$

Now we bound the increase of the potential function as

$$\Phi_{B'} - \Phi_B \leq \delta_B + \sum_{\ell=1}^{k-1} \gamma_B^{e_\ell} + \varepsilon_B. \tag{1.7}$$

To justify this bound, we fix a block B and look at the cache state after requests on edges $e_1, \ldots, e_{k-1}$ are processed. How many free slots there can be in the cache? There are initial δ_B free slots in the beginning of the block B, and the number of free slots can be further increased when the number of pages in the cache associated with $e_1, \ldots, e_{k-1}$ decreases. This increase can be naturally bounded by $\sum_{\ell=1}^{k-1} \gamma_B^{e_\ell}$. Therefore, the number of free slots in the cache is at most $\delta_B + \sum_{\ell=1}^{k-1} \gamma_B^{e_\ell}$.

Because of the property (e), the number of cached a^e-pages and $\bar{b}^e$-pages can only increase by using the free cache space or caching new pages instead of α^e-pages and β^e-pages. We already bounded the number of free slots and ε_B is a natural bound for the increase gained on α^e-pages and β^e-pages. Thus, the bound (1.7) is correct.

Summing (1.7) over all $B \in \overline{\mathcal{B}}$, we have

$$\Phi_F - \Phi_{I'} = \sum_{B \in \overline{\mathcal{B}}} (\Phi_{B'} - \Phi_B) \leq \sum_{B \in \overline{\mathcal{B}}} \left(\delta_B + \sum_{\ell=1}^{k-1} \gamma_B^{e_\ell} + \varepsilon_B \right)$$

which we combine with (1.6) into

$$H - \delta_F \leq \sum_{B \in \overline{\mathcal{B}}} \left(\delta_B + \sum_{\ell=1}^{k-1} \gamma_B^{e_\ell} + \varepsilon_B \right),$$

and use Lemmata 1.5 and 1.6 to bound $\sum \varepsilon_B$ as

$$\begin{aligned} \sum_{B \in \overline{\mathcal{B}}} \varepsilon_B &\geq H - \delta_F - \sum_{B \in \overline{\mathcal{B}}} \left(\delta_B + \sum_{\ell=1}^{k-1} \gamma_B^{e_\ell} \right) \\ &\geq H - n - n - (k-1)6n \\ &\geq H - 6mn - 2n = n + 1. \end{aligned}$$

As there is at most one page of size three requested in each block (property (f)), the inequality $\sum \varepsilon_B \geq n+1$ implies that there are at least $n+1$ blocks where an α^e-page or a β^e-page is cached. At most n blocks have δ_B non-zero (Lemma 1.5); we are done. □

We are ready to complete the proof of the harder direction.

Lemma 1.8. *Suppose that there exists a service of $\mathcal{T}_G$ with the total savings of at least $(d-1)mH + K$. Then the graph G has an independent set W of cardinality K.*

Proof. Let W be a set of K vertices such that the corresponding page p_v is cached between its two requests. (There are at least K of them because the maximal savings on edge-pages are $(d-1)mH$.)

Consider an arbitrary edge $e = \{u, v\}$. Due to Lemma 1.7, there exists a block B such that $\delta_B = 0$ and some α^e-page or β^e-page is cached in the beginning of the block. This block B is either in the u-phase or in the v-phase, because of the statement of the property (f). This means that at least one of the two pages p_u and p_v is not cached between its two requests, because the cache is full. As a consequence, the set W is indeed independent. □

The value of H was set to $6mn + 3n + 1$, therefore Reduction 1.1 is indeed polynomial. Lemmata 1.2, 1.4 and 1.8 together imply that there is an independent set of cardinality K in G if and only if there is a service of the instance $\mathcal{I}_G$ with the total savings of at least $(d-1)mH + K$. We showed that the problem 3Caching(fault,optional) is indeed strongly NP-hard.

1.3 Bit Model

In this subchapter, we show how to modify the proof for the fault model from the previous subchapters so that it works as a proof for the bit model as well.

Reduction 1.9. *Let G be a graph and $\mathcal{I}_G$ the corresponding instance of the problem* 3Caching(fault,optional) *from Reduction 1.1. Then the modified instance $\widetilde{\mathcal{I}}_G$ is an instance of* 3Caching(bit,optional) *with the same cache size and the same set of pages with the same sizes.*

The structure of phases and requests on vertex-pages is also preserved. The blocks from $\mathcal{I}_G$ are also used, but between each pair of consecutive blocks there are five new blocks inserted. Let B and B' be two consecutive blocks. Between B and B' we insert five new blocks $B_{(1)}, \ldots, B_{(5)}$ with the following requests

- *$B_{(1)}$: do not request anything;*
- *$B_{(2)}$: request all pages of size two that are requested both in B and B';*
- *$B_{(3)}$: request a page (there is either one or none) of size three that is requested both in B and B';*
- *$B_{(4)}$: request all pages of size two that are requested both in B and B';*
- *$B_{(5)}$: do not request anything.*

See Figure 1.5 for an example. In each new block, the order of chosen requests is the same as in B (which is the same as in B', as both follow the same ordering of edges (1.1)). The new instance has the total of $\tilde{d} = d + 5(d-1) = 6d - 5$ blocks. This time we prove that the maximal total savings are $(\tilde{d}-1)mH + K$ where K is the cardinality of the maximum independent set in G.

Lemma 1.10. *Suppose that the graph G has an independent set W of cardinality K. Then there exists a service of the modified instance $\widetilde{\mathcal{I}}_G$ with the total savings of at least $(\tilde{d}-1)mH + K$.*

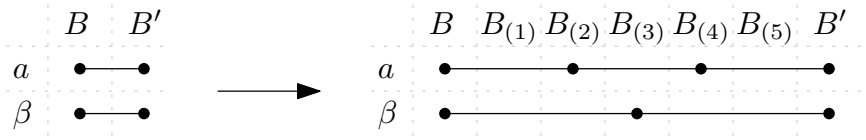


Figure 1.5: The modification of the instance for a page a of size two and a page β of size three

Proof. We consider the service of the original instance $\mathcal{I}_G$ described in the proof of Lemma 1.2 and modify it so it becomes a service of the modified instance.

In the new service, vertex-pages are served the same way as in the original service. The savings on vertex-pages are thus again K.

For each pair of consecutive blocks B and B', each page kept in the cache between B and B' in the original service is kept in the cache in the new service for the whole time between B and B' (it spans over seven blocks now). For a page of size two, savings of two are incurred three times. For a page of size three, savings of three are incurred twice. On each page in the new service we save six instead of one. Therefore, the total savings on edge-pages are $6(d-1)mH = (\tilde{d}-1)mH$.

The total savings are $(\tilde{d}-1)mH + K$. □

Lemma 1.11. *Suppose that there exists a service of the modified instance $\widetilde{\mathcal{I}}_G$ with the total savings of at least $(\tilde{d}-1)mH+K$. Then the graph G has an independent set W of cardinality K.*

Proof. This lemma is the same as Lemma 1.8. We just need to verify that the modified instance fulfills Properties 1.3.

To prove that the property (b) is preserved, we observe that each two consecutive requests on a page of size two are separated by exactly one block where the page is not requested. Consequently, when there are savings of two on a request on the page of size two, we assign savings of one to the block where the savings were incurred and savings of one to the previous block. Similarly, each pair of consecutive requests on a page of size three is separated by exactly two blocks where the page is not requested. When there are savings of three on a request on the page of size three, we assign savings of one to the block where the savings were incurred and savings of one to each of the two previous blocks. As a consequence, the total savings gained on edge-pages may indeed be computed as $\sum s_B$.

The property (a) is preserved because the requests on vertex-pages are the same in both instances. The property (c) is preserved because the initial and final blocks are the same in both instances.

Each sequence of requests in a block of the modified instance $\widetilde{\mathcal{I}}_G$ is either the same or a subsequence of the sequence in a block in the original instance. Therefore, the properties (d), (e) and (f) are preserved as well. □

Lemmata 1.10 and 1.11 imply that we have a valid polynomial-time reduction and so the problem 3Caching(bit,optional) is strongly NP-hard.

1.4 Forced Policy

Theorem 1.12. *Both the problem* 3Caching(fault,forced) *and the problem* 3Caching(bit,forced) *are strongly* NP*-hard.*

Proof. For both the fault model and the bit model, we show a polynomial-time reduction from caching with optional policy to the corresponding variant of caching with the forced policy. Let us have an instance of caching with the optional policy with the cache size k and the request sequence $\rho = r_1 \dots r_n$; let M be the maximal size of a page in ρ (in our previous reductions, $M = 3$).

We create an instance of caching with the forced policy. The cache size is $k' = k + M$. The request sequence is $\rho' = r_1 q_1 r_2 q_2 \dots r_n q_n$ where $q_1, \dots, q_n$ are requests to n different pages that do not appear in ρ and have size M. The costs of the new pages are one in the fault model and M in the bit model.

We claim that there is a service of the optional instance with savings S if and only if there is a service of the forced instance with savings S.

$\Rightarrow$ We serve the requests on original pages the same way as in the optional instance. The cache is larger by M which is the size of the largest page. Thus, pages that were not loaded into the cache because of the optional policy fit in there; we can load them and immediately evict them. New pages fit into the cache as well and we also load them and immediately evict them. This way we have the same savings as in the optional instance.

$\Leftarrow$ We construct a service for the optional instance: For each i, when serving r_i we consider the evictions done when serving r_i and q_i of the forced instance. If a page requested before r_i is evicted, we evict it as well. If a page requested by r_i is evicted, we do not cache it at all. Because the page requested by q_i has size M, the original pages occupy at most k slots in the cache when q_i is served. This way we obtain a service of the optional instance with the same savings.

Using the strong NP-hardness of the problems 3Caching(fault,optional) and 3Caching(bit,optional) proven in Subchapters 1.1 and 1.2 and the observation that the reduction preserves the maximal size of a page, we obtain the strong NP-hardness of the problems 3Caching(fault,forced) and 3Caching(bit,forced). □

1.5 Simple Proof

In this subchapter, we present a simple variant of the proof for the almost-fault model with two distinct costs. This completes the sketch of the proof presented at the end of Subchapter 1.1. We present it with a complete description of the simplified reduction, so that it can be read independently of the rest of the thesis. This subchapter can therefore serve as a short proof of the hardness of general caching.

Theorem 1.13. *General caching is strongly* NP*-hard, even in the case when page sizes are limited to* $\{1, 2, 3\}$ *and there are only two distinct fault costs.*

We prove the theorem for the optional policy. It is easy to obtain the theorem also for the forced policy the same way as in the proof of Theorem 1.12.

The Reduction

The reduction described here will be equivalent to Reduction 1.1 with $H = 1$ and the fault cost of each vertex-page set to $1/(n + 1)$.

Suppose we have a graph $G = (V, E)$ with n nodes and m edges. We construct an instance of general caching whose optimal solution encodes a maximum independent set in G. Fix an arbitrary numbering of edges $e_1, \ldots, e_m$.

The cache size is $k = 2m + 1$. For each vertex v, we have a vertex-page p_v with size one and cost $1/(n + 1)$. For each edge e, we have six associated edge-pages $a^e, \bar{a}^e, \alpha^e, b^e, \bar{b}^e, \beta^e$; all have cost one, pages α^e, β^e have size three and the remaining pages have size two.

The request sequence is organized in phases and blocks. There is one phase for each vertex. In each phase, there are two adjacent blocks associated with every edge e incident with v; the incident edges are processed in an arbitrary order. In addition, there is one initial block I before all phases and one final block F after all phases. Altogether, there are $d = 4m + 2$ blocks. There are four blocks associated with each edge e; denote them B_1^e, B_2^e, B_3^e, B_4^e, in the order as they appear in the request sequence.

For each $v \in V$, the associated page p_v is requested exactly twice, right before the beginning of the v-phase and right after the end of the v-phase; these requests do not belong to any phase. An example of the structure of phases and blocks is given in Figure 1.6.

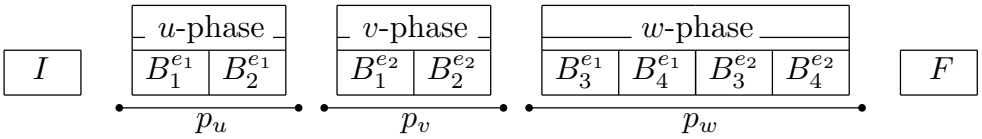


Figure 1.6: An example of phases, blocks and requests on vertex-pages for a graph with three vertices u, v, w and two edges $e_1 = \{u, w\}$, $e_2 = \{v, w\}$ when $H = 2$

Even though each block is associated with some fixed edge, it contains one or more requests to the associated pages for every edge e. In each block, we process the edges $e_1, \ldots, e_m$ in this order. For each edge e, we make one or more requests to the associated pages as follows. If the current block is:

- before B_1^e: request $\bar{a}^e$;
- B_1^e: request $\bar{a}^e$, α^e, and b^e;
- B_2^e: request α^e, a^e, and b^e;
- after B_2^e and before B_3^e: request a^e and b^e;
- B_3^e: request a^e, b^e, and β^e;
- B_4^e: request a^e, β^e, and $\bar{b}^e$;
- after B_4^e: request $\bar{b}^e$.

Figure 1.7 shows an example of the requests on edge-pages associated with one particular edge.

Proof of Correctness

Instead of minimizing the service cost, we maximize the savings compared to the service which does not use the cache at all. This is clearly equivalent when considering the decision version of the problem.

Without loss of generality, we assume that any page is brought into the cache only immediately before a request to that page and removed from the cache only immediately after a request to that page; furthermore, at the beginning and at

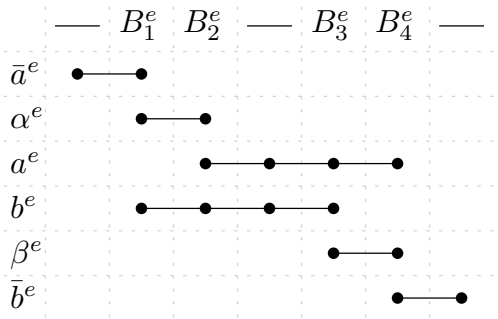


Figure 1.7: Requests on all pages associated with the edge e. Each column represents some block(s). The four labelled columns represent the blocks in the heading, the first column represents every block before B_1^e, the middle column represents every block between B_3^e and B_4^e, and the last column represents every block after B_4^e. The requests in one column are ordered from top to bottom.

the end the cache is empty. I.e., a page may be in the cache only between two consecutive requests to this page, and either it is in the cache for the whole interval or not at all.

Each page of size three is requested only twice in two consecutive blocks, and these blocks are distinct for all pages of size three. Thus, a service of edge-pages is valid if and only if at each time, at most m edge-pages are in the cache. It is thus convenient to think of the cache as of m *slots* for edge-pages.

Each vertex-page is requested twice. Thus, the savings on the n vertex-pages are at most $n/(n+1) < 1$. Since all edge-pages have cost one, the optimal service must serve them optimally. Furthermore, a vertex-page can be cached if and only if during the phase it never happens that at the same time all slots for edge-pages are full and a page of size three is cached.

Let S_B denote the set of all edge-pages cached at the beginning of the block B and let $s_B = |S_B|$. Now observe that each edge-page is requested only in a contiguous segment of blocks, once in each block. It follows that the total savings on edge-pages are equal to $\sum_B s_B$ where the sum is over all blocks. In particular, the maximal possible savings on the edge-pages are $(d-1)m$, using the fact that S_I is empty.

We prove that there is a service with the total savings of at least $(d-1)m + K/(n+1)$ if and only if there is an independent set of size K in G. First the easy direction.

Lemma 1.14. *Suppose that G has an independent set W of size K. Then there exists a service with the total savings of $(d-1)m + K/(n+1)$.*

Proof. For any e, denote $e = uv$ so that u precedes v in the ordering of phases. If $u \in W$, we keep $\bar{a}^e, b^e, \bar{b}^e$ and β^e in the cache from the first to the last request on each page, and we do not cache a^e and α^e at any time. Otherwise we cache $\bar{b}^e, a^e, \bar{a}^e$ and α^e, and do not cache b^e and β^e at any time. In both cases, at each time at most one page associated with e is in the cache and the savings on those pages is $(d-1)m$. See Figure 1.8 for an illustration.

For any $v \in W$, we cache p_v between its two requests. To check that this is a valid service, observe that if $v \in W$, then during the corresponding phase no

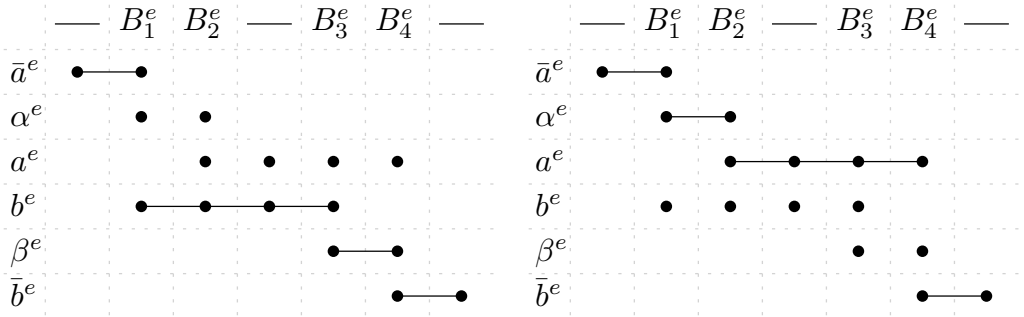


Figure 1.8: The two ways of caching in Lemma 1.14

page of size three is cached. Thus the page p_v always fits in the cache together with at most m pages of size two. □

Now we prove the converse in a sequence of claims. Fix a valid service with savings at least $(d-1)m$. For a block B, let B' denote the following block.

Claim 1.15. *For any block B, with the exception of $B = I$, we have $s_B = m$.*

Proof. For each $B \neq I$ we have $s_B \leq m$. Because $s_I = 0$, the total savings on edge-pages are $\sum_B s_B \leq (d-1)m$. We need an equality. □

We now prove that each edge occupies exactly one slot during the service.

Claim 1.16. *For any block $B \neq I$ and for any e, S_B contains exactly one page associated with e.*

Proof. Let us use the notation $S_B^{\leq k} = S_B^{e_1} \cup \cdots \cup S_B^{e_k}$ and $s_B^{\leq k} = \left|S_B^{\leq k}\right|$. First, we shall prove for each $k \leq m$

$$s_B^{\leq k} = k. \tag{1.8}$$

This is true for $B = F$, as only the m edge-pages $\bar{b}^e$ can be cached there, and by the previous claim all of them are indeed cached. Similarly for $B = I'$ (i.e., immediately following the initial block).

If (1.8) is not true, then for some k and $B \notin \{I, F\}$ we have $s_B^{\leq k} < s_{B'}^{\leq k}$. Then after processing the edge e_k in the block B we have in the cache all the pages in $(S_B \setminus S_B^{\leq k}) \cup S_{B'}^{\leq k}$. Their number is $(m - s_B^{\leq k}) + s_{B'}^{\leq k} > m$, a contradiction.

The statement of the claim is an immediate consequence of (1.8). □

Claim 1.17. *For any edge e, at least one of the pages α^e and β^e is cached between its two requests.*

Proof. Assume that none of the two pages is cached. It follows from the previous claim that $b^e \in S_{B_2^e}$, as at this point α^e and b^e are the only pages associated with e that can be cached. Similarly, $a^e \in S_{B_4^e}$.

It follows that there exists a block B between B_1^e and B_4^e such that S_B contains the page b^e and $S_{B'}$ contains the page a^e. However, in B, the page a^e is requested before the page b^e. Thus at the point between the two requests, the cache contains two pages associated with e, plus one page associated with every other edge, the total of $m+1$ pages, a contradiction. □

Now we are ready to complete this direction.

Lemma 1.18. *Suppose that there exists a valid service with the total savings of* $(d-1)m + K/(n+1)$. *Then* G *has an independent set* W *of size* K.

Proof. Let W be the set of all v such that p_v is cached between its two requests. The total savings imply that $|W| = K$.

Now we claim that W is independent. Suppose not, let $e = uv$ be an edge with $u, v \in W$. Then p_u and p_v are cached in the corresponding phases. Thus neither α^e nor β^e can be cached, since together with other $m-1$ requests of size 2 associated with the remaining edges, the cache size needed would be $2m+2$. However, this contradicts the last claim. □

Lemmata 1.14 and 1.18 together show that we constructed a valid polynomial-time reduction from the problem of independent set to general caching. Therefore, Theorem 1.13 is proven.

2. Layers and Work Functions

Only uniform caching under the forced policy is considered in this chapter. When designing the first H_k-competitive randomized algorithm (the ratio H_k is optimal with respect to the competitive analysis), McGeoch and Sleator [19] invented a dynamically changing partition of the request sequence which enables an online algorithm to keep track of the possible moves of the optimal algorithm. Subsequently, Koutsoupias and Papadimitriou [17] extended the result and showed that such a partition can be used to fully (and efficiently) characterize the associated work function. These concepts are introduced in the first subchapter.

Koutsoupias and Papadimitriou used quasi-convexity lemma to prove the result. In the second subchapter, we give a new and simpler proof that layers do characterize the associated work function. We will show that a direct extension of this proof to the case with variable cache size is possible in the next chapter.

2.1 Definitions and Basics

Work functions. Work functions originally come from the *k-server problem* proposed by Manasse et al. [18]. A metric space is given in this problem and k servers occupy points from this space. The input sequence consists of requests on points of the space; if the requested point is not occupied by any of the servers, the algorithm must move one of the servers there. The service cost is the total distance travelled by the servers. This problem is equivalent to uniform caching when the distance between each two points of the metric space is one.

In the aforementioned pioneering work by Manasse et al. [18], the famous *k-server conjecture* is postulated: There is a k-competitive deterministic algorithm for the k-server problem. The conjecture remains unresolved. To stress the importance of work functions, we note that the best known deterministic algorithm is the Work Function Algorithm and it was proven to be $(2k-1)$-competitive by Koutsoupias and Papadimitriou [16]. The actual competitive ratio is unknown and it is conjectured that this algorithm is k-competitive.

Before giving the definition of work function, we set up the notation and conventions. The empty sequence is denoted by (). A set of pages is called a *configuration*. If a configuration contains exactly ℓ pages, then it is called an ℓ-configuration. For two configurations X and Y, $c(X \to Y)$ is the transition cost of changing the cache configuration from X to Y. To do that, all pages $X \setminus Y$ must be evicted and all pages $Y \setminus X$ must be loaded. Therefore, $c(X \to Y) = |Y \setminus X|$. If $\mathcal{S}$ is a set of configurations, then $c(\mathcal{S} \to Y) = \min_{X \in \mathcal{S}} c(X \to Y)$.

Without loss of generality, we can assume that there are always exactly k pages in the cache in the beginning as we can put new pages there that are never requested instead of leaving some free space in the cache. We can also assume that pages are only evicted when necessary. Therefore, we can safely restrict ourselves to k-configurations. We write $\mathcal{P}$ for the universe of all pages, $\mathcal{C} = \binom{\mathcal{P}}{k}$ for the set of all k-configurations and C_0 for the initial cache k-configuration.

Definition 2.1 (Work function)**.** *Let σ be a request sequence. The work function associated with σ is a function $\omega_\sigma \colon \mathcal{C} \to \mathbb{N}_0$. For a k-configuration C, $\omega_\sigma(C)$*

determines the minimum cost of a service of the sequence σ ending in the configuration C.

Another page replacements are allowed after the last request is served and therefore work function is also defined for configurations not containing the last requested page.

> Intuitively, the importance of work functions stems from the almost obvious fact that they encapsulate all the useful information about the past; what an on-line algorithm needs to remember is ω_σ, not σ, because any other algorithm can be transformed to one with this property without deteriorating its competitiveness.
>
> — *Koutsoupias and Papadimitriou [16]*

We use $\min \omega$ as a shortcut for $\min_{X \in \mathcal{C}} \omega(X)$. Instead of $X \cup \{p\}$ and $X \setminus \{p\}$ we write $X + p$ and $X - p$. We will be using the following fact about work functions. The proof is simple and may be found in the literature, e.g. [16].

Fact 2.2. *The work function may be computed recurrently as*

$$\omega_{()}(C) = c(C_0 \to C);$$
$$\omega_{\sigma p}(C) = \begin{cases} \omega_\sigma(C) & \text{if } p \in C, \\ 1 + \min\limits_{x \in C} \omega_\sigma(C + p - x) & \text{otherwise.} \end{cases}$$

Layers. Koutsoupias and Papadimitriou [17] proved that the work function is coned-up from the set of minimal configurations, i.e. for each $C \in \mathcal{C}$ there is $X \in \mathcal{C}$ such that $\omega_\sigma(X) = \min \omega_\sigma$ and $\omega_\sigma(C) = \min \omega_\sigma + c(X \to C)$. They also proved that the set of minimal configurations can be represented by a specific structure; when designing the algorithm Equitable, Achlioptas et al. [1] changed the notation for this structure and called it layers. We use a slight modification of layers used by Negoescu et al. [20, 10] where the order of layer sets is inversed and there is an additional layer L_0. This simplifies, in our opinion, the analysis. We stress that this characterization does not hold for the general k-server problem.

Definition 2.3 (Layer representation)**.** *Let the sets of pages $L_0, \ldots, L_\ell$ be nonempty and pairwise disjoint. We say that the set of ℓ-configurations $\mathcal{S}$ is represented by layers $L_0, \ldots, L_\ell$, if for each ℓ-configuration X it holds*

$$X \in \mathcal{S} \Leftrightarrow (\forall i \in \{0, \ldots, \ell - 1\})\, |X \cap L_{\le i}| \le i,$$

where $L_{\le i}$ is $\bigcup_{j=0}^{i} L_j$. As a shortcut we use $\mathcal{S} = (L_0 \mid \ldots \mid L_\ell)$.

The following lemma on computing the transition cost from a set of layers to a given set of pages is simple, but we are not aware of any similar result.

Lemma 2.4. *Let the set of ℓ-configurations $\mathcal{S}$ be represented by layers $L_0, \ldots, L_\ell$. For a configuration P the transition cost from the set $\mathcal{S}$ may be computed as*

$$c(\mathcal{S} \to P) = \max_{i=0}^{\ell-1} \{|P \cap L_{\le i}| - i\}.$$

Moreover, for each $p \in P \cap L_{\ge 1}$ there is a configuration $X \in \mathcal{S}$ such that $p \in X$ and $c(X \to P) = c(\mathcal{S} \to P)$.

Proof. Let us write m for $\max_{i=0}^{\ell-1}\{|P \cap L_{\leq i}| - i\}$. It is easy to see that $c(\mathcal{S} \to P) \geq m$, because each ℓ-configuration $X \in \mathcal{S}$ fulfills $|X \cap L_{\leq i}| - i \leq 0$ for each i due to Definition 2.3.

We construct a configuration $X = \{p_1, \ldots, p_\ell\} \in \mathcal{S}$ such that $c(X \to P) = m$. For each $j = 1, \ldots, \ell - m$ we pick p_j as the page from $L_{\geq j} \cap P \setminus \{p_1, \ldots, p_{j-1}\}$ with the lowest layer number. If such a selection is possible, then clearly it holds $(\forall i)|\{p_1, \ldots, p_{\ell-m}\} \cap L_{\leq i}| \leq i$ and it is possible to choose $p_{\ell-m+1}, \ldots, p_\ell$ such that $X \in \mathcal{S}$. Then clearly $c(X \to P) \leq m$ and we are done.

It remains to prove that the selection process for $p_1, \ldots, p_{\ell-m}$ is valid. It is enough to show for each $j \leq \ell - m$ the inequality

$$|L_{\geq j} \cap P| \geq (\ell - m) - (j - 1).$$

Because $|P| = \ell$, it holds $|L_{\geq j} \cap P| = \ell - |L_{\leq j-1} \cap P|$. We obtain inequality $\ell - |L_{\leq j-1} \cap P| \geq (\ell - m) - (j - 1)$, after a simplification $|L_{\leq j-1} \cap P| - (j - 1) \leq m$ which holds because of the very definition of m.

To prove the second part of the lemma, we observe that if a page $p \in P \cap L_{\geq 1}$ is not in X, then the configuration $Y = X + p - p_\ell$ is the desired configuration fulfilling $Y \in \mathcal{S}$, $c(Y \to P) = c(\mathcal{S} \to P)$ and $p \in Y$. □

2.2 New Proof

The structure of the proof is as follows: We begin with "guessing" the right set of configurations on which the work function attains its minimum, then show how this set is representable by layers. Finally, we show that these configurations are indeed minimal and moreover that they cone-up the whole work function.

Definition 2.5 (The set of minimal configurations)**.** *For a request sequence σ, the set $\mathcal{M}_\sigma$ is defined recurrently as follows.*

$$\mathcal{M}_() = \{C_0\};$$
$$\mathcal{M}_{\sigma p} = \begin{cases} \{C \mid C \in \mathcal{M}_\sigma \wedge p \in C\} & \text{if } (\exists C \in \mathcal{M}_\sigma) p \in C, \\ \{C + p - x \mid C \in \mathcal{M}_\sigma \wedge x \in C\} & \text{otherwise.} \end{cases}$$

Lemma 2.6 (Updating layers)**.** *Let us denote $C_0 = \{p_1, \ldots, p_k\}$, then*

$$\mathcal{M}_() = (\mathcal{P} \setminus C_0 \mid \{p_1\} \mid \ldots \mid \{p_k\}).$$

Let us assume $\mathcal{M}_\sigma = (L_0 \mid \ldots \mid L_k)$ and $p \in L_i$, then

$$\mathcal{M}_{\sigma p} = \begin{cases} (L_0 \mid \ldots \mid L_{i-2} \mid L_{i-1} \cup L_i - p \mid L_{i+1} \mid \ldots \mid L_k \mid \{p\}) & \text{if } i > 0, \\ (L_0 - p \mid \ldots \mid L_{k-2} \mid L_{k-1} \cup L_k \mid \{p\}) & \text{if } i = 0. \end{cases}$$

Proof. We consider the case of an empty sequence first. By Definition 2.5 we have $\mathcal{M}_() = \{C_0\}$ and layers $(\mathcal{P} \setminus C_0 \mid \{p_1\} \mid \ldots \mid \{p_k\})$ represent a single configuration $\{p_1, \ldots, p_k\} = C_0$.

Assume $\mathcal{M}_\sigma = (L_0 \mid \ldots \mid L_k)$ and $p \in L_i$. We first consider the case $i > 0$. We shall show that the layers

$$(L'_0 \mid \ldots \mid L'_k) = (L_0 \mid \ldots \mid L_{i-2} \mid L_{i-1} \cup L_i - p \mid L_{i+1} \mid \ldots \mid L_k \mid \{p\})$$

represent exactly those configurations represented by $(L_0 \mid \ldots \mid L_k)$ that contain p. This becomes clear when we explicitly write what $L'_{\leq j}$ looks like,

$$L'_{\leq j} = \begin{cases} L_{\leq j} & \text{if } j \in \{0, \ldots, i-1\}, \\ L_{\leq j+1} - p & \text{if } j \in \{i, \ldots, k-1\}. \end{cases}$$

In the case of $i = 0$, we shall show that the updated layer representation

$$(L'_0 \mid \ldots \mid L'_k) = (L_0 \mid \ldots \mid L_{k-2} \mid L_{k-1} \cup L_k \mid \{p\})$$

represents exactly those configurations that are created from configurations represented by $(L_0 \mid \ldots \mid L_k)$ by exchanging one page for p. This again becomes clear when we explicitly write what $L'_{\leq j}$ looks like,

$$L'_{\leq j} = \begin{cases} L_{\leq j} - p & \text{if } j \in \{0, \ldots, k-2\}, \\ L_{\leq k} - p & \text{if } j = k-1. \end{cases}$$

□

Theorem 2.7. *Let σ be a request sequence. Then the corresponding work function ω_σ may be computed for each configuration $C \in \mathcal{C}$ as*

$$\omega_\sigma(C) = \min \omega_\sigma + c(\mathcal{M}_\sigma \to C).$$

Proof. We prove the theorem by induction on the request sequence.

The base case of an empty request sequence is clear, because we have $\mathcal{M}_() = \{C_0\}$ and $\omega_()(C) = c(C_0 \to C)$ for every configuration C.

Let us assume that the theorem holds for the request sequence σ and prove the theorem for the sequence σp. We denote the layers representing $\mathcal{M}_\sigma$ by $(L_0 \mid \ldots \mid L_k)$. We shall show for every configuration $C \in \mathcal{C}$ that $\omega_{\sigma p}(C) = \min \omega_{\sigma p} + c(\mathcal{M}_{\sigma p} \to C)$.

Case 1: If $p \in C$, then due to Fact 2.2, we have $\omega_{\sigma p}(C) = \omega_\sigma(C)$ and it remains to prove

$$\min \omega_{\sigma p} + c(\mathcal{M}_{\sigma p} \to C) = \min \omega_\sigma + c(\mathcal{M}_\sigma \to C). \tag{2.1}$$

Case 1a: If $p \in L_{\geq 1}$, then there is a configuration $D \in \mathcal{M}_\sigma$ such that $p \in D$ and $c(D \to C) = c(\mathcal{M}_\sigma \to C)$ due to Lemma 2.4. For this configuration $\omega_{\sigma p}(D) = \omega_\sigma(D) = \min \omega_\sigma + c(D \to \mathcal{M}_\sigma) = \min \omega_\sigma$ and therefore $\min \omega_{\sigma p} = \min \omega_\sigma$ as the work function is non-decreasing. And finally $c(\mathcal{M}_{\sigma p} \to C) = c(\mathcal{M}_\sigma \to C)$ because $D \in \mathcal{M}_{\sigma p}$ (Definition 2.5) and $\mathcal{M}_{\sigma p} \subseteq \mathcal{M}_\sigma$. The summation of $\min \omega_{\sigma p} = \min \omega_\sigma$ and $c(\mathcal{M}_{\sigma p} \to C) = c(\mathcal{M}_\sigma \to C)$ gives (2.1).

Case 1b: If $p \in L_0$, then $\min \omega_{\sigma p} = \min \omega_\sigma + 1$ as there is no configuration containing p in $\mathcal{M}_\sigma$. Because $\mathcal{M}_{\sigma p}$ is the set of all configurations created by exchanging one page with p in a configuration from $\mathcal{M}_\sigma$, we obtain

$$c(\mathcal{M}_{\sigma p} \to C) = c(\mathcal{M}_\sigma \to C) - 1.$$

Summing the two results we again obtain (2.1).

Case 2: If $p \notin C$, we use Fact 2.2 to get

$$\omega_{\sigma p}(C) = 1 + \min_{x \in C} \omega_\sigma(C + p - x) = 1 + \min_{x \in C} \omega_{\sigma p}(C + p - x).$$

Because $p \in C + p - x$ and we already solved Case 1, we get

$$\omega_{\sigma p}(C) = 1 + \min \omega_{\sigma p} + \min_{x \in C} c(\mathcal{M}_{\sigma p} \to C + p - x).$$

Because each configuration in $\mathcal{M}_{\sigma p}$ contains p and $p \notin C$, we have

$$\min_{x \in C} c(\mathcal{M}_{\sigma p} \to C + p - x) = c(\mathcal{M}_{\sigma p} \to C) - 1.$$

Therefore, $\omega_{\sigma p}(C) = \min \omega_{\sigma p} + c(\mathcal{M}_{\sigma p} \to C)$. □

3. Variable Cache Size and Layers

Allowing the cache size to vary over time is a natural generalization of caching. To give an example, the best known approximation result for general caching in the offline version is a 4-approximation due to Bar-Noy et al. [6] and this approximation implicitly works for caching with variable cache size as well. The world of uniform caching (under the fault policy) with variable cache size was pioneered by Peserico [21].

We introduce the concept of caching with variable cache size in the first subchapter and we give an extension of layers to the case with variable cache size in the second subchapter. In the third subchapter, an extension of layers to caching under the optional policy through simulating the optional policy by changing the cache size.

3.1 Introduction

We set up a notation to deal with uniform caching with variable cache size. As in Chapter 2, the universe of pages is denoted by $\mathcal{P}$. We again use $c(X \to Y) = |Y \setminus X|$ as the cost of the transition from the configuration X to the configuration Y. It is no longer possible to restrict ourselves to k-configurations and therefore this time $\mathcal{C} = 2^{\mathcal{P}}$. We also use the notation $\binom{A}{\leq n} = \{X \mid X \subseteq A \wedge |X| \leq n\}$.

A request sequence σ is a sequence of requests $r_1, \ldots, r_n$ where either $r_i = \updownarrow k_i$ meaning a request on changing the cache size to k_i or $r_i = [p_i]$ meaning a request on the page p_i. We denote the up-to-date cache size by k_σ. It is $k_() = 0$, $k_{\sigma[p]} = k_\sigma$ and $k_{\sigma\updownarrow k} = k$. When a page is requested, the cache size must be greater than zero.

As mentioned in Introduction, uniform caching is solvable with a natural algorithm known as Belady's rule. The proof that Belady's rule works for the case with variable cache size as well was given by Peserico [21]. The proof is in fact implicit as it is enough to notice that classical proofs (such as the one in [9]) do not depend on the assumption that the cache size is constant.

Theorem 3.1 (Belady's rule for variable cache size)**.** *Let $\rho = r_1, \ldots, r_n$ be a request sequence of uniform caching with variable cache size.*

A service with the optimal cost may be constructed using the following rule: If the eviction of a page is necessary (either a new page is requested and it would not fit into the cache, or the cache size is decreased), evict the page whose further request is the furthest in the future.

It is possible to directly implement Belady's rule in time $\mathcal{O}(n \log n)$.

Definition 3.2 (Generalized work function)**.** *For a request sequence σ the (generalized) work function is a function $\omega_\sigma : \mathcal{C} \to \mathbb{N}_0$ such that $\omega_\sigma(X)$ gives the optimal cost of serving the request sequence while ending in the configuration X.*

The cache size limits must be respected during the whole service, but after serving the last request it is possible to load and evict arbitrary number of pages

from the cache no matter what the cache size is. This ensures that the work function is well-defined for all configurations.

We prove the following analogy of Fact 2.2.

Lemma 3.3. *The work function may be computed recurrently as*

$$\begin{aligned}
\omega_{()}(C) &= c(\emptyset \to C) = |C|;\\
\omega_{\sigma\downarrow k}(C) &= \min_{X \in \binom{C}{\leq k}} \left(\omega_\sigma(X) + c(X \to C)\right);\\
\omega_{\sigma[p]}(C) &= \begin{cases} \omega_\sigma(C) & \text{if } |C| \leq k_\sigma \wedge p \in C,\\ \min\limits_{X \in \binom{C-p}{\leq k_\sigma - 1}} \left(\omega_\sigma(X+p) + c(X+p \to C)\right) & \text{otherwise.} \end{cases}
\end{aligned}$$

Proof. The case of an empty sequence is clear as there is nothing in the cache in the beginning and we must move to the desired configuration.

To justify the remaining three equations, we realize that it is always necessary to end up in some configuration which fits into the cache (and contains p if necessary) and then, after the service is done, move to the desired configuration. Because evicting pages is for free, we can safely assume that all pages outside $C + p$ are evicted before loading the new pages. □

3.2 Layers for Variable Cache Size

This time, we do not guess the set of minimal configurations as in Definition 2.5, but only the set of minimal configurations that are maximal on inclusion; we call these configurations optimal.

Definition 3.4 (The set of optimal configurations)**.** *For a request sequence σ, the set $\mathcal{O}_\sigma$ and the parameter ℓ_σ are defined recurrently as follows,*

$$\begin{aligned}
\ell_{()} &= 0;\\
\mathcal{O}_{()} &= \{\emptyset\};\\
\ell_{\sigma\downarrow k} &= \min\{k, \ell_\sigma\};\\
\mathcal{O}_{\sigma\downarrow k} &= \begin{cases} \{C \mid |C| = k \wedge (\exists X \in \mathcal{O}_\sigma) C \subseteq X\} & \text{if } k < \ell_\sigma,\\ \mathcal{O}_\sigma & \text{otherwise;} \end{cases}\\
\ell_{\sigma[p]} &= \begin{cases} \ell_\sigma + 1 & \text{if } \ell_\sigma < k_\sigma \text{ and } (\forall C \in \mathcal{O}_\sigma) p \notin C,\\ \ell_\sigma & \text{otherwise;} \end{cases}\\
\mathcal{O}_{\sigma[p]} &= \begin{cases} \{C \mid C \in \mathcal{O}_\sigma \wedge p \in C\} & \text{if } (\exists C \in \mathcal{O}_\sigma) p \in C,\\ \{C + p \mid C \in \mathcal{O}_\sigma\} & \text{else if } \ell_\sigma < k_\sigma,\\ \{C + p - x \mid C \in \mathcal{O}_\sigma \wedge x \in C\} & \text{otherwise.} \end{cases}
\end{aligned}$$

It is easy to see that each configuration in $\mathcal{O}_\sigma$ contains ℓ_σ pages. Now we show to represent the set of optimal configurations by layers (Definition 2.3).

Lemma 3.5 (Updating layers)**.** *Let σ be a request sequence. The set $\mathcal{O}_\sigma$ may be represented by layers with $\ell_\sigma + 1$ layer sets. The corresponding layers may be computed recurrently as follows.*

For the empty sequence, $\mathcal{O}_{()}$ is represented by trivial layers, $\mathcal{O}_{()} = (\mathcal{P})$.

Let us assume $\mathcal{O}_\sigma = (L_0 \mid \ldots \mid L_{\ell_\sigma})$ *and* $p \in L_i$*, then*

$$\mathcal{O}_{\sigma\updownarrow k} = \begin{cases} (L_0 \mid \ldots \mid L_{\ell_\sigma}) & \text{if } \ell_\sigma \le k, \\ (L_0 \mid \ldots \mid L_{k-1} \mid L_k \cup \cdots \cup L_{\ell_\sigma}) & \text{otherwise;} \end{cases}$$

$$\mathcal{O}_{\sigma[p]} = \begin{cases} (L_0 \mid \ldots \mid L_{i-2} \mid L_{i-1} \cup L_i - p \mid \ldots \mid L_{\ell_\sigma} \mid \{p\}) & \text{if } i > 0, \\ (L_0 - p \mid \ldots \mid L_{\ell_\sigma - 1} \mid L_{\ell_\sigma} \mid \{p\}) & \text{if } i = 0 \text{ and } \ell_{\sigma[p]} = \ell_\sigma + 1, \\ (L_0 - p \mid \ldots \mid L_{\ell_\sigma - 2} \mid L_{\ell_\sigma - 1} \cup L_{\ell_\sigma} \mid \{p\}) & \text{if } i = 0 \text{ and } \ell_{\sigma[p]} = \ell_\sigma. \end{cases}$$

Proof. The case of an empty sequence is clear as $(\mathcal{P})$ represents only the empty configuration and $\mathcal{O}_() = \{\emptyset\}$.

Assume $\mathcal{O}_\sigma = (L_0 \mid \ldots \mid L_{\ell_\sigma})$ and $p \in L_i$. We consider the case of a request on changing the cache size to k. If $\ell_\sigma \le k$, there is nothing to prove. If $\ell_\sigma > k$, then it is easy to see that a k-configuration C is represented by $(L_0 \mid \ldots \mid L_{k-1} \mid L_k \cup \cdots \cup L_{\ell_\sigma})$ if and only if it is a subset of a configuration represented by $(L_0 \mid \ldots \mid L_{\ell_\sigma})$.

Now we assume that the next request demands the page p. The cases $i > 0$ and $i = 0 \wedge \ell_\sigma = k_\sigma$ are the same as in Lemma 2.6. In the case $i > 0 \wedge \ell_{\sigma[p]} = \ell_\sigma + 1$, we are supposed to prove that layers

$$\left(L'_0 \mid \ldots \mid L'_{\ell_\sigma + 1}\right) = (L_0 - p \mid \ldots \mid L_{\ell_\sigma} \mid \{p\})$$

represent exactly those configurations that are created from configurations represented by $(L_0 \mid \ldots \mid L_{\ell_\sigma})$ by adding the page p. Because $L'_{\ell_\sigma + 1} = \{p\}$, each configuration represented by $\left(L'_0 \mid \ldots \mid L'_{\ell_\sigma + 1}\right)$ contains p. For the other ℓ_σ pages $P = \{p_1, \ldots, p_{\ell_\sigma}\}$ different from p, conditions $|L_{\le i} \cap P| \le i$ and $\left|L'_{\le i} \cap P\right| \le i$ for $i = 0, \ldots, \ell_\sigma - 1$ are equivalent. □

Theorem 3.6. *Let* σ *be a request sequence, then the corresponding generalized work function* ω_σ *may be computed for each configuration* $C \in \mathcal{C}$ *as*

$$\omega_\sigma(C) = \min \omega_\sigma + c(\mathcal{O}_\sigma \to C).$$

Proof. We prove the theorem by induction on the request sequence.

The base case of an empty request sequence is clear, because we have $\mathcal{O}_() = \{\emptyset\}$ and $\omega_()(C) = c(\emptyset \to C)$ for every configuration C.

Let us assume that the theorem holds for the request sequence σ and prove the theorem for the sequence σr. We use the notation $\mathcal{O} = \mathcal{O}_\sigma$, $\mathcal{O}' = \mathcal{O}_{\sigma r}$, $\omega = \omega_\sigma$, $\omega' = \omega_{\sigma r}$, $\ell = \ell_\sigma$, $\ell' = \ell_{\sigma r}$, $k = k_\sigma$, $k' = k_{\sigma r}$. We denote the layers representing $\mathcal{O}$ by $(L_0 \mid \ldots \mid L_k)$. We shall show for every configuration $C \in \mathcal{C}$ that $\omega'(C) = \min \omega' + c(\mathcal{O}' \to C)$.

Case 1: $r = \updownarrow k'$

In this case $\min \omega = \min \omega'$ ($\omega(\emptyset) = \min \omega$ due to the induction hypothesis and $\omega'(\emptyset) = \omega(\emptyset)$ due to Lemma 3.3). According to Lemma 3.3 it holds

$$\omega'(C) = \min_{X \in \binom{C}{\le k'}} (\omega(X) + c(X \to C)),$$

rewriting using the induction hypothesis

$$\omega'(C) = \min \omega + \min_{X \in \binom{C}{\le k'}} (c(\mathcal{O} \to X) + c(X \to C)).$$

It remains to prove $\min_{X \in \binom{C}{\leq k'}} (c(\mathcal{O} \to X) + c(X \to C)) = c(\mathcal{O}' \to C)$.

We observe

$$\min_{X \in \binom{C}{\leq k'}} (c(\mathcal{O} \to X) + c(X \to C)) = \min_{\substack{Y : |Y| \leq k' \\ Y \subseteq Z : Z \in \mathcal{O}}} c(Y \to C).$$

And clearly

$$\min_{\substack{Y : |Y| \leq k' \\ Y \subseteq Z : Z \in \mathcal{O}}} c(Y \to C) = c(\mathcal{O}' \to C).$$

Case 2: $r = [p]$ and $p \in C$

Case 2a: $(\exists X \in \mathcal{O}) p \in X$

In this case $\min \omega = \min \omega'$ because $\omega(X) = \min \omega$ due to the induction hypothesis and $\omega'(X) = \omega(X)$ due to Lemma 3.3. For the same reason, $(\forall Y \in \mathcal{O}')\omega'(Y) = \omega(Y)$.

Due to Lemma 2.4, there is $Y \in \mathcal{O}$ such that $p \in Y \wedge c(\mathcal{O} \to C) = c(Y \to C)$. $Y \in \mathcal{O}'$ and therefore $c(\mathcal{O}' \to C) = c(\mathcal{O} \to C)$. As a consequence $\omega'(C) \leq \omega(C)$. We are done.

Case 2b: $p \notin L_{\geq 1}$ and $\ell_\sigma < k_\sigma$

In this case $\min \omega' = \min \omega + 1$ as no configuration in $\mathcal{O}_\sigma$ contains p. It is easy to see $c(\mathcal{O}' \to C) = c(\mathcal{O} \to C) - 1$. In this case also $\omega(C) = \omega'(C)$ due to Lemma 3.3.

Case 2c: $p \notin L_{\geq 1}$ and $\ell_\sigma = k_\sigma$

Again, $\min \omega' = \min \omega$.

If $c(\mathcal{O} \to C) > 1$, then $\omega'(C) = \omega(C)$ and $c(\mathcal{O}' \to C) = c(\mathcal{O} \to C) - 1$.

If $c(\mathcal{O} \to C) = 0$, then $c(\mathcal{O}' \to C) = 0$ and $\omega'(C) = \omega(C) + 1$.

Case 3: $r = [p]$ and $p \notin C$

According to Lemma 3.3,

$$\begin{aligned} \omega'(C) &= \min_{X \in \binom{C-p}{\leq k_\sigma - 1}} (\omega(X + p) + c(X + p \to C)) \\ &= \min_{X \in \binom{C-p}{\leq k_\sigma - 1}} (\omega'(X + p) + c(X + p \to C)) . \end{aligned}$$

We make use of having the theorem proven for the case $p \in C$:

$$\begin{aligned} \omega'(C) &= \min_{X \in \binom{C-p}{\leq k_\sigma - 1}} (\min \omega' + c(\mathcal{O}' \to X + p) + c(X + p \to C)) \\ &= \min \omega' + \min_{X \in \binom{C-p}{\leq k_\sigma - 1}} (c(\mathcal{O}' \to X + p) + c(X + p \to C)) . \end{aligned}$$

To finish the proof we observe

$$\min_{X \in \binom{C}{\leq k_\sigma - 1}} (c(\mathcal{O}' \to X + p) + c(X + p \to C)) = c(\mathcal{O}' \to C).$$

□

3.3 Optional Policy

In this subchapter, we show that results around uniform caching under the forced policy holds in a very similar form also under the optional policy.

Theorem 3.7. *Let σ be a request sequence for uniform caching under the optional policy with variable cache size. Then the corresponding set of optimal configurations $\mathcal{O}_\sigma$ may be represented by layers.*

For $\sigma = ()$, $\mathcal{O}_{()} = (\mathcal{P})$. Assuming $\mathcal{O}_\sigma = (L_0 \mid \ldots \mid L_\ell)$ and $p \in L_i$, then

$$\mathcal{O}_{\sigma\updownarrow k} = \begin{cases} (L_0 \mid \ldots \mid L_\ell) & \text{if } \ell \le k_\sigma, \\ (L_0 \mid \ldots \mid L_{k-1} \mid L_k \cup \cdots \cup L_\ell) & \text{otherwise;} \end{cases}$$

$$\mathcal{O}_{\sigma[p]} = \begin{cases} (L_0 \mid \ldots \mid L_{i-2} \mid L_{i-1} \cup L_i - p \mid \ldots \mid L_\ell \mid \{p\}) & \text{if } i > 0, \\ (L_0 - p \mid \ldots \mid L_{\ell-1} \mid L_\ell \mid \{p\}) & \text{if } i = 0 \text{ and } \ell < k_\sigma, \\ (L_0 - p \mid \ldots \mid L_{\ell-1} \mid L_\ell + p) & \text{if } i = 0 \text{ and } \ell = k_\sigma. \end{cases}$$

Proof. We show how to mimic the optional policy by the forced policy. Let σ be the request sequence for the optional policy. We create the sequence σ' for the forced policy: We replace each request $[p]$ by three requests $\updownarrow(k+1), [p], \updownarrow k$ where k is the cache size at the time of the request [p].

The proof that a service for each of the request sequences may be transformed to a service of the other sequence is the same as in Theorem 1.12.

It remains to show that the layers are indeed updated as in the statement of this theorem. There is nothing to show in the case $i > 0$. In the case of $i = 0$, the request $\updownarrow(k+1)$ ensures that after processing the request $[p]$, the page p is put into the new layer $L_{\ell+1} = \{p\}$. When processing the request $\updownarrow k$, nothing changes if $\ell < k$, or the the layer $L_{\ell+1}$ is united with L_ℓ if $\ell = k$. □

A simple consequence of such a simulation of the optional policy through changing the cache size is also the following modification of Belady's rule for the optional policy:

Theorem 3.8 (Belady's rule for the optional policy)**.** *Let $\rho = r_1, \ldots, r_n$ be a request sequence of uniform caching with variable cache size under the optional policy.*

A service with the optimal cost may be constructed using the following rule: If a page request comes and a fault occurs, add the page into the cache first. If the number of pages exceeds the cache size, evict the page whose further request is the furthest in the future. Similarly, if a request on changing the cache size arrives and the cache size is exceeded, repeatedly evict the page whose furthest request is the furthest in the future until the pages fit into the cache.

4. Algorithms for General Caching

In this chapter, we give two algorithms solving general caching. The time complexity of these algorithms depends on parameters of the caching instance and the complexity becomes polynomial when the parameters are suitably bounded. Therefore, these algorithms may be considered as polynomial algorithms for special instances of general caching.

We remind the terminology from Introduction which is crucial for this chapter: A page is called *normal* if both its size and fault cost are one; otherwise it is called *abnormal*.

4.1 Algorithm 1

The parameter η is equal to the total number of requests on abnormal pages in the instance of caching. We give an algorithm that is polynomial when the number of requests on abnormal pages is bounded by a logarithmic function of the request sequence length.

Theorem 4.1. *There is an algorithm for general caching with the running time $\mathcal{O}(2^\eta \cdot n \log n)$, this holds for both the forced and the optional policy.*

Proof. We consider the service of the abnormal pages first. As in Chapter 1, we can without loss of generality assume that each page is either evicted right after processing the request on it (or even never loaded in the case of the optional policy) or is kept in the cache until another request on it arrives. This means that the total number of services of the abnormal pages is at most 2^η.

When a service of the abnormal pages is fixed, we are able to compute what is the free cache space that can be used for the normal pages. This means that the problem essentially boils down to caching with variable cache size. Due to Theorem 3.1 (the forced policy) and Theorem 3.8 (the optional policy), it is solvable using Belady's rule and, as was mentioned in the previous chapter, it can be implemented with running time $\mathcal{O}(n \log n)$. For the total of at most 2^η services we obtain the time complexity $\mathcal{O}(2^\eta \cdot n \log n)$. □

4.2 Algorithm 2

The parameter κ is defined to be the difference between the cache size and the size of the smallest abnormal page. We give an algorithm which is effective when each page is either normal or very large.

Theorem 4.2. *There is an algorithm for general caching with the running time $n^{\mathcal{O}(\kappa)}$, this holds for both the forced and the optional policy.*

Proof. We distinguish two cases. If $\kappa \geq k/2$, then the total cache size is $\leq 2\kappa$. In this case it is possible to solve the problem with a simple dynamic programming in time $n^{\mathcal{O}(\kappa)}$ as there are at most $n^{\mathcal{O}(\kappa)}$ cache configurations.

Now we proceed to the case $\kappa < k/2$. In this case, only one abnormal page fits into the cache and we use the more intuitive name *large pages* for the abnormal pages. To simplify the analysis, we add one additional request r_{n+1} on a new page of size k and fault cost 0 at the end of the request sequence.

As in the proof of Theorem 4.1, we again assume without loss of generality that each large page is either evicted right after the request (or not even loaded into the cache) or it remains in the cache until the next request on this page. And we also again use the fact that when the service of large pages is fixed, the problem becomes caching with variable cache size. We describe the algorithm for the optional policy first, then we explain the small modifications for the forced policy. We use Theorem 3.7 stating that the work function of caching with variable cache size may be characterized by layers in the case of the optional policy.

For each request r_i on a large page we wish to compute the set D_{r_i} of all pairs $(\mathcal{L}, c)$ such that layers $\mathcal{L}$ represent a work function corresponding to a sequence of caching with variable cache size which is gained by fixing a service of large pages from $r_1, \ldots, r_i$, which loads the page requested by r_i into the cache, and then updating layers with variable cache size according to Theorem 3.7. The cost c is the sum of the minimum of the work function and the cost of the fixed service of the large pages. There might me more ways of reaching a work function represented by layers $\mathcal{L}$; it is naturally enough to consider the one with the minimum value of c.

We show how to compute all the sets D_r using dynamic programming. Let r_i be a request on a large page P and assume that for each preceding request r_j on a large page, the set D_{r_j} is already computed.

We distinguish two kinds of services of all the previous requests and treat them separately. In the first kind, the page P is already present in the cache when the request r_i arrives. Let r_j be the previous request on the page P. To compute the part of D_{r_i} corresponding to this kind of services, we consider each $(\mathcal{L}, c) \in D_{r_j}$. The service of large pages among $r_{i+1}, \ldots, r_{j-1}$ is determined – these large pages are not even loaded into the cache and so we add to c the total fault cost of pages of all these requests. As far as requests on normal pages among $r_{i+1}, \ldots, r_{j-1}$ are concerned, we update the layers and add to c the increase in the minimum of the work function (the cache size remains set to $k - \textsc{size}(P)$ for the whole time).

In the second kind of services we need to deal with, the page P is not present in the cache when r_i arrives and must be loaded right now. We must consider all possibilities which was the last large page present in the cache before r_i arrived. It can also be the case that r_i is the fist large page that was ever loaded into the cache. In this case, we begin with the trivial layers $(\mathcal{P})$ which are the layers with the only layer L_0 containing all pages from the universe, raise the cache size to k, process all requests $r_1, \ldots, r_{i-1}$ as in the previous case and decrease the cache size to $k - \textsc{size}(P)$. Finally, we add to c the cost $\textsc{cost}(P)$.

Let us fix a request r_j on a large page Q preceding r_i $(j < i)$. To cover all services which have the page Q as the last page in the cache before r_i and Q is lastly evicted after r_j, we consider each $(\mathcal{L}, c) \in D_{r_j}$ and we update $\mathcal{L}$ by setting the cache size to k, processing all requests $r_{j+1}, \ldots, r_{i-1}$ as in the previous cases and decreasing the cache size to $k - \textsc{size}(P)$. Finally, we add to c the increase in the minimum of the work function and $\textsc{cost}(P)$.

This finishes the description of the computation of all D_r. We can without

loss of generality assume that in each optimal service the page r_{n+1} is loaded into the cache, because it does not cost anything. In the set $D_{r_{n+1}}$, there is only one element in the form $((\mathcal{P}), c)$ as the page requested by r_{n+1} fills the entire cache and this c does naturally determine the optimum cost of a service for the caching instance.

Let us bound the time complexity of the algorithm. The size of each set D_r is at most $n^{\kappa+1}$ as there are always at most $\kappa+1$ layers (we use the simplest bound considering all assignments of pages into those $\kappa+1$ layers). The transitions between various layer representations according to the rules from Theorem 3.7 are clearly computable in polynomial time and the complexity $n^{\mathcal{O}(\kappa)}$ follows.

It remains to explain how the algorithm looks like under the forced policy; this case is in fact a bit simpler. This time we update the layers according to Lemma 3.5 and Theorem 3.6. As the pages must be always loaded in the cache, it is enough to consider the previous request on a large page and distinguish two cases: Either the previous requested page remains in the cache (this can happen only when both requests demand the very same page), or the page is evicted after the previous request. □

Conclusion

We list the set of open questions following the results presented in this work.

In the first chapter, we proved that general caching restricted to pages of sizes $\{1, 2, 3\}$ is strongly NP-hard, even in the case of the bit model and fault model. Complexity status of general caching with page sizes restricted to $\{1, 2\}$ remains an interesting open problem.

Question 1. *Is general caching also (strongly)* NP*-hard when page sizes are limited to $\{1, 2\}$? Can caching with page sizes $\{1, 2\}$ be solved in polynomial time, at least in the bit or fault model?*

In Chapters 2 and 3, the structure of layers on which H_k-competitive algorithms for uniform caching are based was discussed. The only known $\mathcal{O}(\log k)$-competitive algorithms beyond uniform caching are based on rounding the solution of a linear program. It is still open whether there is a combinatorial algorithm for the problem.

Question 2. *Is there a randomized $\mathcal{O}(\log k)$-competitive algorithm for general caching (or at least for one of the cost, bit or fault model) which is combinatorial, for example similar to the algorithms for uniform caching?*

The first algorithm given in the fourth chapter shows that caching is solvable in polynomial time if the number of requests on abnormal pages is logarithmic. The natural question arrives whether it is also solvable when there is only a bound on the total number of different abnormal pages, not on the number of requests on them.

Question 3. *Is general caching solvable in polynomial time if the total number of different requested abnormal pages is bounded by $\mathcal{O}(\log n)$ or a constant?*

The second given algorithm is polynomial when there is a constant c such that each abnormal page has size at least $k - c$. In the proof for the case $c < k/2$, it is very useful that only one abnormal page fits into the cache. Therefore, we ask whether this property is sufficient for obtaining a polynomial algorithm.

Question 4. *Is general caching solvable in polynomial time if each abnormal page has size strictly greater than half of the cache, at least in the bit or fault model?*

Bibliography

[1] D. Achlioptas, M. Chrobak, and J. Noga, *Competitive analysis of randomized paging algorithms*, Theoretical Computer Science, 234 (2000), pp. 203–218. A preliminary version appeared at ESA 1996.

[2] A. Adamaszek, A. Czumaj, M. Englert, and H. Räcke, *An O(log k)-competitive algorithm for generalized caching*, in Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2012, pp. 1681–1689.

[3] S. Albers, S. Arora, and S. Khanna, *Page replacement for general caching problems*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 31–40.

[4] N. Bansal, N. Buchbinder, and J. Naor, *A primal-dual randomized algorithm for weighted paging*, Journal of the ACM, 59 (2012), p. 19. A preliminary version appeared at FOCS 2007.

[5] ——, *Randomized competitive algorithms for generalized caching*, SIAM Journal on Computing, 41 (2012), pp. 391–414. A preliminary version appeared at STOC 2008.

[6] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, *A unified approach to approximating resource allocation and scheduling*, Journal of the ACM, 48 (2001), pp. 1069–1090. A preliminary version appeared at STOC 2000.

[7] W. W. Bein, L. L. Larmore, and J. Noga, *Equitable revisited*, in Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings, L. Arge, M. Hoffmann, and E. Welzl, eds., vol. 4698 of Lecture Notes in Computer Science, Springer, 2007, pp. 419–426.

[8] L. A. Belady, *A study of replacement algorithms for a virtual-storage computer*, IBM Systems Journal, 5 (1966), pp. 78–101.

[9] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[10] G. S. Brodal, G. Moruz, and A. Negoescu, *Onlinemin: A fast strongly competitive randomized paging algorithm*, Theory of Computing Systems, 56 (2015), pp. 22–40. A preliminary version appeared at WAOA 2011.

[11] M. Chrobak, H. J. Karloff, T. H. Payne, and S. Vishwanathan, *New results on server problems*, SIAM Journal on Discrete Mathematics, 4 (1991), pp. 172–181. A preliminary version appeared at SODA 1990.

[12] M. Chrobak, L. L. Larmore, C. Lund, and N. Reingold, *A better lower bound on the competitive ratio of the randomized 2-server problem*, Information Processing Letters, 63 (1997), pp. 79–83.

[13] M. Chrobak, G. J. Woeginger, K. Makino, and H. Xu, *Caching is hard – even in the fault model*, Algorithmica, 63 (2012), pp. 781–794. A preliminary version appeared at ESA 2010.

[14] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, *Competitive paging algorithms*, Journal of Algorithms, 12 (1991), pp. 685–699. A preliminary version appeared in 1988.

[15] S. Irani, *Page replacement with multi-size pages and applications to web caching*, Algorithmica, 33 (2002), pp. 384–409. A preliminary version appeared at STOC 1997.

[16] E. Koutsoupias and C. H. Papadimitriou, *On the k-server conjecture*, Journal of the ACM, 42 (1995), pp. 971–983. A preliminary version appeared at STOC 1994.

[17] ——, *Beyond competitive analysis*, SIAM Journal on Computing, 30 (2000), pp. 300–317. A preliminary version appeared at FOCS 1994.

[18] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for server problems*, Journal of Algorithms, 11 (1990), pp. 208–230. A preliminary version appeared at STOC 1988.

[19] L. A. McGeoch and D. D. Sleator, *A strongly competitive randomized paging algorithm*, Algorithmica, 6 (1991), pp. 816–825. A preliminary version appeared in 1989.

[20] A. Negoescu, *Design of Competitive Paging Algorithms with Good Behaviour in Practice*, PhD thesis, Goethe University Frankfurt, 2013.

[21] E. Peserico, *Paging with dynamic memory capacity*, CoRR, abs/1304.6007 (2013).

[22] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM, 28 (1985), pp. 202–208.

[23] N. E. Young, *On-line file caching*, Algorithmica, 33 (2002), pp. 371–383. A preliminary version appeared at SODA 1998.