Algorithmica manuscript No.
(will be inserted by the editor)

General Caching Is Hard: Even with Small Pages

Lukáš Folwarczný · Jiří Sgall

August 1, 2016

Abstract *Caching* (also known as *paging*) is a classical problem concerning page replacement policies in two-level memory systems. *General caching* is the variant with pages of different sizes and fault costs. The strong NP-hardness of its two important cases, the *fault model* (each page has unit fault cost) and the *bit model* (each page has the same fault cost as size) has been established, but under the assumption that there are pages as large as half of the cache size. We prove that this already holds when page sizes are bounded by a small constant: The bit and fault models are strongly NP-complete even when page sizes are limited to $\{1, 2, 3\}$.

Considering only the decision versions of the problems, general caching is equivalent to the *unsplittable flow on a path problem* and therefore our results also improve the hardness results about this problem.

Keywords General caching · Small pages · NP-hardness · Unsplittable Flow on a Path

1 Introduction

Caching (also known as *uniform caching* or *paging*) is a classical problem in the area of online algorithms and has been extensively studied since 1960s. It models a two-level memory system: There is the fast memory of size C (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

Partially supported by the Center of Excellence – ITI, project P202/12/G061 of GA ČR (J. Sgall) and by the project 14-10003S of GA ČR (L. Folwarczný).

Computer Science Institute of Charles University,
Faculty of Mathematics and Physics,
Malostranské nám. 25, CZ-11800 Praha 1,
Prague, Czech Republic.
E-mail: {folwar,sgall}@iuuk.mff.cuni.cz

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 make space for the new one when there are already C pages in the cache. The natural objective is to evict pages in such a way that the total fault cost is minimized. For a reference on classical results, see Borodin and El-Yaniv [11].

In 1990s, with the advent of World Wide Web, a generalized variant called *file caching* or simply *general caching* was studied [19,21]. 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, caching under the *optional policy.*

Previous work. In this article, we consider the problem of finding the optimal service in the *offline version* of caching when the whole request sequence is known in advance. Uniform caching is solvable in polynomial time with a natural algorithm known as Belady's rule [9]. Caching in the cost model is a special case of the *k-server problem* and is also solvable in polynomial time [13]. In late 1990s, the questions about the complexity status of general caching were raised. The situation was summed up by Albers et al. [3]: *"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."*

There was no improvement until a breakthrough in 2010 when Chrobak et al. [15] 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.

Independently, a weak NP-hardness of the fault model was proven by Darmann et al. [16]. A substantially simpler proof of the strong NP-hardness of general caching was given by Bonsma et al. [10]; only pages of sizes in $\{1, 2, 3\}$ are needed in the proof, but they have many different costs: This means that the costs are far from the fault model and in fact they are also far from the bit model.

Contribution. We give a novel proof of strong NP-hardness for general caching which gives the first hardness result restricted to small pages in the fault and bit models:

Theorem 1.1 *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 each of the forced and optional policies.*

We also give a separate self-contained version of the proof of the result for general costs (and sizes $\{1, 2, 3\}$). This version is rather simple, in particular significantly simpler than the one given by Chrobak et al. [15] and at the same time it uses only two different costs, which is a simpler structure compared to Bonsma et al. [10]. At the same time, the simplified version illustrates some of the main ideas of our reduction. The reductions for the result in the fault and bit models are significantly more involved and require a non-trivial potential-function-like argument.

Open problems. We prove that general caching with page sizes $\{1, 2, 3\}$ is strongly NP-hard while general caching with unit page sizes is easily polynomially solvable. Closing the remaining gap will definitely contribute to a better understanding of the problem:

Question 1.2 *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 a broader perspective, the complexity status of general caching is still far from being well understood as the best known result on approximation is a 4-approximation due to Bar-Noy et al. [7] and there is no result on the hardness of approximation. Therefore, better understanding of the complexity of general caching remains an important challenge.

Question 1.3 *Is there an algorithm for general caching with an approximation ratio better than 4, or even a PTAS? Is general caching* APX*-hard?*

Related work. In the decision version, offline general caching is equivalent (together with interval scheduling/packing, resource allocation and other problems) to the *unsplittable flow on a path problem* (UFPP). This equivalence does not translate to the approximation version of the problem, as the flow corresponds not to the cost of caching, but rather to the complementary quantity of the total saving of the cost compared to the trivial service that faults on each request. An important parameter in the world of UFPP is *task density*, in the language of caching it is the fault cost divided by the page size. An approximation scheme with quasi-polynomial time complexity when the ratio of the maximum density to the minimum density is quasi-polynomial was given by Bansal et al. [6]. The first non-trivial instance of UFPP for which a PTAS was invented by Batra et al. [8] is the one when the task densities are in a constant range. The bit model of caching is equivalent to the case when all task densities are equal to one. The best known approximation for UFPP is $(2+\varepsilon)$-approximation due to Anagnostopoulos et al. [4]. It is also known that UFPP is not APX-hard, unless NP $\subseteq$ DTIME$(2^{\mathrm{polylog}(n)})$ [6].

We wish to conclude the overview of results with recent advances in randomized online caching (competitive analysis against oblivious adversary is considered). Classical results state that the lower bound on the competitive ratio for uniform caching is H_C (the C-th harmonic number) [17] and H_C-competitive algorithms were designed [20,1]; it has also been known that for general caching an H_C-competitive algorithm is impossible [14]. Recently, Brodal et al. [12] developed a substantially faster H_C-competitive algorithm for uniform caching. Based on new methods for rounding of linear programs, Bansal et al. [5] recently invented an $\mathcal{O}(\log^2 C)$-competitive algorithm for general caching and Adamaszek et al. [2] improved the algorithm to be $\mathcal{O}(\log C)$-competitive.

Outline. 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 Section 2 and its validity is proven in Section 3. In Section 4, we show how to modify the reduction so that it works for the bit model as well. In Section 5, 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 Appendix A.

An extended abstract of this paper appeared in Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC 2015) [18].

2 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 C 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 C 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 section, 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 section, 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 2.1 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.

The corresponding instance $\mathcal{I}_G$ of 3CACHING(FAULT,OPTIONAL) is an instance with the cache size $C = 2mH + 1$ 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 $\bar{a}^e_i, \alpha^e_i, a^e_i, b^e_i, \beta^e_i, \bar{b}^e_i$ where pages α^e_i and β^e_i have size three and the remaining four pages have size two.

For a fixed edge e, let $\bar{a}^e$-pages be all pages $\bar{a}^e_i$ 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 Fig. 1.

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

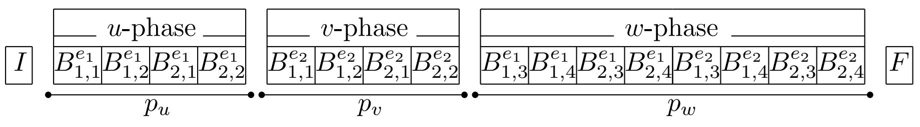
I
u-phase
$B^{e_1}_{1,1}$ $B^{e_1}_{1,2}$ $B^{e_1}_{2,1}$ $B^{e_1}_{2,2}$
v-phase
$B^{e_2}_{1,1}$ $B^{e_2}_{1,2}$ $B^{e_2}_{2,1}$ $B^{e_2}_{2,2}$
w-phase
$B^{e_1}_{1,3}$ $B^{e_1}_{1,4}$ $B^{e_1}_{2,3}$ $B^{e_1}_{2,4}$ $B^{e_2}_{1,3}$ $B^{e_2}_{1,4}$ $B^{e_2}_{2,3}$ $B^{e_2}_{2,4}$
F
p_u
p_v
p_w

Fig. 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 $H = 2$

Table 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 $B^e_{i,4}$		•	$\bar{b}^e_i$

$1, \ldots, H$ 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 determines which pages are requested. Note that the number of requests to pages of size two is between mH and $2mH$ per block; in addition, each block except for the initial and final ones contains exactly one request to a page of size three.

Reduction 2.1 is now complete. An example of requests on edge-pages associated with one edge e is depicted in Fig. 2. Note 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

$$\bar{a}^e_1\, \alpha^e_1\, a^e_1\, \ldots\, \bar{a}^e_i\, \alpha^e_i\, a^e_i\, \ldots\, \bar{a}^e_H\, \alpha^e_H\, a^e_H\; b^e_1\, \beta^e_1\, \bar{b}^e_1\, \ldots\, b^e_i\, \beta^e_i\, \bar{b}^e_i\, \ldots\, b^e_H\, \beta^e_H\, \bar{b}^e_H. \qquad (1)$$

Preliminaries for the proof. Instead of minimizing the service cost, we maximize the total saving 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.

At each time, there exists at most one page of size three that can possibly be cached. This follows since each block contains at most one request to a page of size three and each page of size three is requested only twice in two

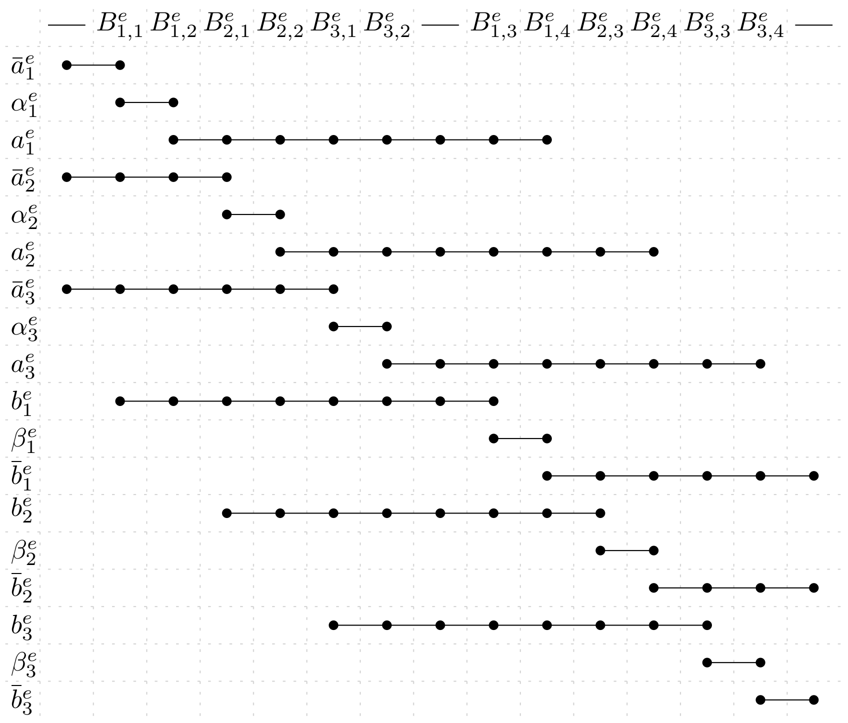

Fig. 2 Requests on all pages associated with the edge e when $H = 3$. Each column represents some block(s). The labeled columns represent the blocks in the heading, the first column represents every block before $B^e_{1,1}$, the middle column represents every block between $B^e_{3,2}$ and $B^e_{1,3}$,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

consecutive blocks. Since the cache size $C = 2m + 1$ is odd, this implies that 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 saving on the n vertex-pages is 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^e_B be the set of pages in S_B associated with the edge e. We use $s_B = |S_B|$ and $s^e_B = |S^e_B|$ 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 an edge-page can possibly be in S_B only if it is requested both in B and in the previous block. The total saving on edge-pages is then equal to $\sum_B s_B$ where the sum is over all blocks. In particular, the maximal possible saving on the edge-pages is $(d - 1)mH$, using the fact that S_I is empty. We shall show that the maximum saving is $(d-1)mH + K$ 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 saving on vertex-pages is $n/(n+1) < 1$ which is less than a saving achieved by any single 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 Appendix A. 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 $s^e_B = 1$ (see Appendix A for details). Using this we show below that for each edge e, at least one of the pages α^e_1 and β^e_1 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^e_1 and β^e_1 is cached between its two requests. Because pages α^e_1 and β^e_1 are forbidden, there is b^e_1 in $S_{B^e_{1,2}}$ and a^e_1 in $S_{B^e_{1,4}}$. Thus somewhere in the two blocks $B^e_{1,2}$ and $B^e_{1,3}$, we must switch from caching b^e_1 to caching a^e_1. However, this is impossible, because the order of requests implies that we would have to cache both b^e_1 and a^e_1 at some moment, or none of them would be in $B^e_{1,3}$ (see Fig. 3). However, there is no place in the cache for such an operation, as $s^{e'}_B = 1$ for every e' and $B \neq I$ and in each block, the requests associated with any e' are not interleaved with those associated to e.

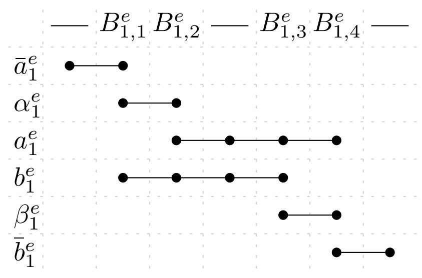

Fig. 3 Pages associated with one edge when $H = 1$

In the fault model, the corresponding claim $s^e_B = H$ does not hold. Instead, we prove that the value of s^e_B cannot change much during the service and when we use H large enough, we still get a working reduction.

3 Proof of Correctness

In this section, we show that the reduction described in the previous section 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 saving at least $(d-1)mH + K$. First the easy direction, which holds for any value of the parameter H.

Lemma 3.1 *Let G be a graph and $\mathcal{I}_G$ the corresponding caching instance from Reduction 2.1. 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 saving 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. Fig. 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 saving on those pages is $(d-1)mH$. We know that the pages fit in the cache because of the observations made in Section 2.

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. □

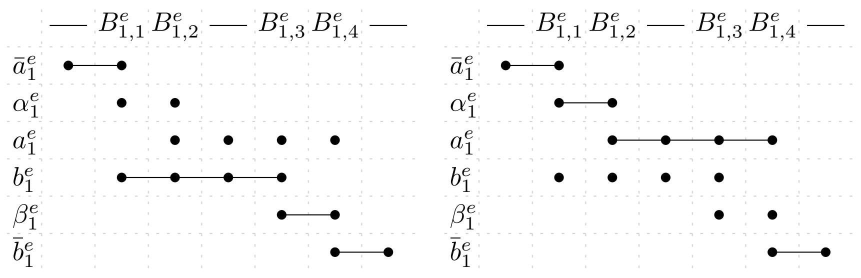

Fig. 4 The two ways of caching in Lemma 3.1

We prove the converse in a sequence of lemmata. In Section 4, 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 3.2 Let $\mathcal{T}_G$ be an instance of general caching corresponding 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 2.1. The request sequence is again 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 saving achieved on edge-pages is 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 is requested in B, $\overline{B}$, or any block between them.

Lemma 3.3 *The instance from Reduction 2.1 satisfies Properties 3.2.*

Proof All properties (a), (b), (c), (d), (f) follow directly from Reduction 2.1 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). We need to verify that when the page a_i^e is requested, no page $\bar{a}_j^e$ for $j \le i$ is requested and that when the page $\bar{b}_i^e$ is requested, no $\bar{a}^e$-page and no page b_j^e for $j \le i$ is requested. This can be seen easily when we explicitly write down the request sequences for each kind of block, see Table 2. □

For the following claims, let $\mathcal{T}_G$ be an instance fulfilling Properties 3.2. We fix a service of $\mathcal{T}_G$ with the total saving 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'.

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 $\gamma_B^e = |s_{B'}^e - s_B^e|$ (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.

Table 2 Request sequences on all pages associated with an edge e

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

Lemma 3.4 *We have (summing over all blocks except for the initial one)*

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

Proof The total saving on vertex-pages is at most n due to the property (a). Using the property (b), $s_I = 0$, and the definition of δ_B, the total saving on edge-pages is equal to

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

The total saving is assumed to be at least $(d-1)mH$, thus summing the savings form vertex- and edge-pages we obtain

$$n + (d-1)mH - \sum_{B\in\mathcal{B}\setminus\{I\}} \delta_B \ge (d-1)mH.$$

The claim of the lemma now 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 3.5 *For each edge $e \in E$,*

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

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

$$\sum_{B\in\overline{\mathcal{B}}} \left|s^{\le k}_{B'} - s^{\le k}_B\right| \le 3n. \tag{2}$$

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

$$s_F^{\leq k} - s_{I'}^{\leq k} \geq -\delta_F. \tag{3}$$

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 $\mathcal{P}$ and Lemma 3.4 give us the first bound

$$\sum_{B \in \mathcal{P}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) \leq \sum_{B \in \mathcal{P}} \delta_B \leq n. \tag{4}$$

Using the fact $\mathcal{P} \mathbin{\dot\cup} \mathcal{N} = \overline{\mathcal{B}}$ and (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 (4) and $\delta_F \leq n$ which follows by Lemma 3.4, we obtain the second bound

$$-\sum_{B \in \mathcal{N}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) \leq \sum_{B \in \mathcal{P}} \left(s_{B'}^{\leq k} - s_B^{\leq k} \right) + \delta_F \leq 2n. \tag{5}$$

Combining the bounds (4) and (5), we prove (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 (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 (2) for $k - 1$ and k together with the formula $|x - y| \leq |x| + |y|$:

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

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 3.6 *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

$$\Phi_{I'} = 0 \quad \text{and} \quad \Phi_F \geq H - \delta_F. \tag{6}$$

Now we bound the increase of the potential function as

$$\Phi_{B'} - \Phi_B \leq \delta_B + \sum_{\ell=1}^{k-1} \gamma_B^{e_\ell} + \varepsilon_B. \tag{7}$$

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 (7) is correct.

Summing (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 (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 3.4 and 3.5 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 3.4); we are done. □

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

Lemma 3.7 *Suppose that there exists a service of $\mathcal{T}_G$ with the total saving 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 vertices such that the corresponding page p_v is cached between its two requests. There are at least K vertices in W because the maximal saving on edge-pages is $(d-1)mH$.

Consider an arbitrary edge $e = \{u, v\}$. Due to Lemma 3.6, 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. Thus the cache of size $2mH+1$ is full, as it contains mH edge-pages including one page of size three. 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 2.1 is indeed polynomial. Lemmata 3.1, 3.3 and 3.7 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 saving at least $(d-1)mH+K$. We showed that the problem 3Caching(fault,optional) is indeed strongly NP-hard.

4 Bit Model

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

Reduction 4.1 Let G be a graph and $\mathcal{I}_G$ the corresponding instance of the problem 3Caching(fault,optional) from Reduction 2.1. 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 Fig. 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)). The new instance has the total of $\tilde{d} = d + 5(d-1) = 6d - 5$ blocks. This time we prove that the maximal total saving is $(\tilde{d}-1)mH + K$ where K is the cardinality of the maximum independent set in G.

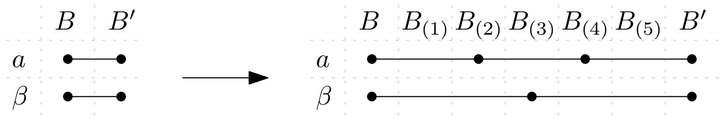

Fig. 5 The modification of the instance for a page a of size two and a page β of size three

Lemma 4.2 *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 saving at least $(\tilde{d}-1)mH + K$.*

Proof We consider the service of the original instance $\mathcal{I}_G$ described in the proof of Lemma 3.1 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 saving on vertex-pages is 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, a saving two is achieved three times. For a page of size three, a saving three is incurred twice. On each page in the new service we save six instead of one. Therefore, the total saving on edge-pages is $6(d-1)mH = (\tilde{d}-1)mH$.

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

Lemma 4.3 *Suppose that there exists a service of the modified instance $\widetilde{\mathcal{I}}_G$ with the total saving 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 3.7. We just need to verify that the modified instance fulfills Properties 3.2.

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 is a saving two on a request on the page of size two, we assign one unit of the saving to the block where the saving was achieved and one unit of the saving 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 is a saving three on a request on the page of size three, we assign one unit of the saving to the block where the saving was achieved and one unit of the saving to each of the two previous blocks. As a consequence, the total saving 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. $\square$

Lemmata 4.2 and 4.3 imply that we have a valid polynomial-time reduction and so the problem 3Caching(bit,optional) is strongly NP-hard.

5 Forced Policy

Theorem 5.1 *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 under optional policy to the corresponding variant of caching with under forced policy. Let us have an instance of caching under the optional policy with the cache size C 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 under the forced policy. The cache size is $C' = C + M$. 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 saving S if and only if there is a service of the forced instance with saving 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 saving 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 C slots in the cache after q_i is served. This way we obtain a service of the optional instance with the same saving.

Using the strong NP-hardness of the problems 3Caching(fault,optional) and 3Caching(bit,optional) proven in Sections 2 and 3 and the observation that the reduction preserves the maximal size of a page, we obtain the strong

NP-hardness of 3Caching(fault,forced) and 3Caching(bit,forced). □

Acknowledgments. We are grateful to the anonymous reviewers, in particular for bringing the articles [10,16] to our attention and for suggestions that helped us to improve the presentation.

References

1. Achlioptas, D., Chrobak, M., Noga, J.: Competitive analysis of randomized paging algorithms. Theoretical Computer Science **234**(1-2), 203–218 (2000). DOI 10.1016/S0304-3975(98)00116-9. URL http://dx.doi.org/10.1016/S0304-3975(98)00116-9. A preliminary version appeared at ESA 1996.
2. Adamaszek, A., Czumaj, A., Englert, M., Räcke, H.: An $O(\log k)$-competitive algorithm for generalized caching. In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1681–1689 (2012). DOI 10.1137/1.9781611973099. URL http://dl.acm.org/citation.cfm?id=2095116.2095249
3. Albers, S., Arora, S., Khanna, S.: Page replacement for general caching problems. In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 31–40 (1999). URL http://dl.acm.org/citation.cfm?id=314500.314528
4. Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: A mazing $2+\varepsilon$ approximation for unsplittable flow on a path. In: C. Chekuri (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pp. 26–41. SIAM (2014). DOI 10.1137/1.9781611973402.3. URL http://dx.doi.org/10.1137/1.9781611973402.3
5. Bansal, N., Buchbinder, N., Naor, J.: Randomized competitive algorithms for generalized caching. SIAM Journal on Computing **41**(2), 391–414 (2012). DOI 10.1137/090779000. URL http://dx.doi.org/10.1137/090779000. A preliminary version appeared at STOC 2008.
6. Bansal, N., Chakrabarti, A., Epstein, A., Schieber, B.: A quasi-PTAS for unsplittable flow on line graphs. In: J.M. Kleinberg (ed.) Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 721–729. ACM (2006). DOI 10.1145/1132516.1132617. URL http://doi.acm.org/10.1145/1132516.1132617
7. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. Journal of the ACM **48**(5), 1069–1090 (2001). DOI 10.1145/502102.502107. URL http://doi.acm.org/10.1145/502102.502107. A preliminary version appeared at STOC 2000.
8. Batra, J., Garg, N., Kumar, A., Mömke, T., Wiese, A.: New approximation schemes for unsplittable flow on a path. In: P. Indyk (ed.) Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 47–58. SIAM (2015). DOI 10.1137/1.9781611973730.5. URL http://dx.doi.org/10.1137/1.9781611973730.5
9. Belady, L.A.: A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal **5**(2), 78–101 (1966). DOI 10.1147/sj.52.0078. URL http://dx.doi.org/10.1147/sj.52.0078
10. Bonsma, P.S., Schulz, J., Wiese, A.: A constant-factor approximation algorithm for unsplittable flow on paths. SIAM Journal on Computing **43**(2), 767–799 (2014). DOI 10.1137/120868360. URL http://dx.doi.org/10.1137/120868360
11. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press (1998)
12. Brodal, G.S., Moruz, G., Negoescu, A.: Onlinemin: A fast strongly competitive randomized paging algorithm. Theory of Computing Systems **56**(1), 22–40 (2015). DOI 10.1007/s00224-012-9427-y. URL http://dx.doi.org/10.1007/s00224-012-9427-y. A preliminary version appeared at WAOA 2011.

13. Chrobak, M., Karloff, H.J., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM Journal on Discrete Mathematics **4**(2), 172–181 (1991). DOI 10.1137/0404017. URL `http://dx.doi.org/10.1137/0404017`. A preliminary version appeared at SODA 1990.
14. Chrobak, M., Larmore, L.L., Lund, C., Reingold, N.: A better lower bound on the competitive ratio of the randomized 2-server problem. Information Processing Letters **63**(2), 79–83 (1997). DOI 10.1016/S0020-0190(97)00099-9. URL `http://dx.doi.org/10.1016/S0020-0190(97)00099-9`
15. Chrobak, M., Woeginger, G.J., Makino, K., Xu, H.: Caching is hard – Even in the fault model. Algorithmica **63**(4), 781–794 (2012). DOI 10.1007/s00453-011-9502-9. URL `http://dx.doi.org/10.1007/s00453-011-9502-9`. A preliminary version appeared at ESA 2010.
16. Darmann, A., Pferschy, U., Schauer, J.: Resource allocation with time intervals. Theoretical Computer Science **411**(49), 4217–4234 (2010). DOI 10.1016/j.tcs.2010.08.028. URL `http://dx.doi.org/10.1016/j.tcs.2010.08.028`
17. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. Journal of Algorithms **12**(4), 685–699 (1991). DOI 10.1016/0196-6774(91)90041-V. URL `http://dx.doi.org/10.1016/0196-6774(91)90041-V`. A preliminary version appeared in 1988.
18. Folwarczný, L., Sgall, J.: General caching is hard: Even with small pages. In: K.M. Elbassioni, K. Makino (eds.) Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings, *Lecture Notes in Computer Science*, vol. 9472, pp. 116–126. Springer (2015). DOI 10.1007/978-3-662-48971-0_11. URL `http://dx.doi.org/10.1007/978-3-662-48971-0_11`
19. Irani, S.: Page replacement with multi-size pages and applications to web caching. Algorithmica **33**(3), 384–409 (2002). DOI 10.1007/s00453-001-0125-4. URL `http://dx.doi.org/10.1007/s00453-001-0125-4`. A preliminary version appeared at STOC 1997.
20. McGeoch, L.A., Sleator, D.D.: A strongly competitive randomized paging algorithm. Algorithmica **6**(6), 816–825 (1991). DOI 10.1007/BF01759073. URL `http://dx.doi.org/10.1007/BF01759073`. A preliminary version appeared in 1989.
21. Young, N.E.: On-line file caching. Algorithmica **33**(3), 371–383 (2002). DOI 10.1007/s00453-001-0124-5. URL `http://dx.doi.org/10.1007/s00453-001-0124-5`. A preliminary version appeared at SODA 1998.

A The Simple Proof

In this appendix, 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 Section 2. We present it with a complete description of the simplified reduction, so that it can be read independently of the rest of the paper. This appendix can therefore serve as a short proof of the hardness of general caching.

Theorem A.1 *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 5.1.

The Reduction

The reduction described here will be equivalent to Reduction 2.1 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 $e_1, \dots, e_m$.

The cache size is $C = 2m + 1$. 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^e_1, B^e_2, B^e_3, B^e_4, 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 Fig. 6.

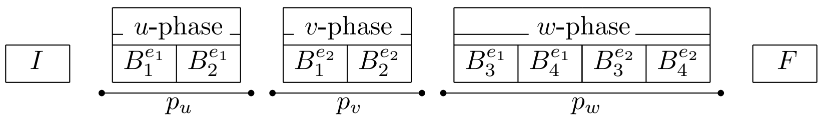

Fig. 6 An example of phases, blocks and requests on vertex-pages for a graph with three vertices u, v, w and two edges $e_1 = \{u, w\}$, $e_2 = \{v, w\}$ when $H = 2$

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 $e_1, \dots, e_m$ in this order. For each edge e, we make one or more requests to the associated pages according to Table 3.

Figure 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 saving compared to the service which does not use the cache at all. This is clearly equivalent when considering the decision version of the problem.

Table 3 Requests associated with an edge e

Block	Requested pages
before B_1^e	$\bar{a}^e$
B_1^e	$\bar{a}^e$, α^e, b^e
B_2^e	α^e, a^e, b^e
between B_2^e and B_3^e	a^e, b^e
B_3^e	a^e, b^e, β^e
B_4^e	a^e, β^e, $\bar{b}^e$
after B_4^e	$\bar{b}^e$

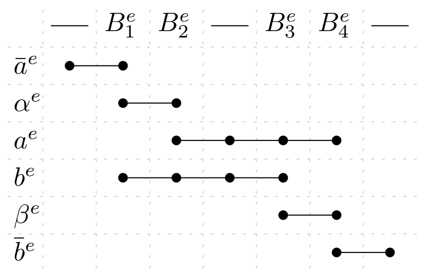

Fig. 7 Requests on all pages associated with the edge e. Each column represents some block(s). The four labeled columns represent the blocks in the heading, the first column represents every block before B_1^e, the middle column represents every block between B_3^e and B_4^e, and the last column represents every block after B_4^e. The requests in one column are ordered from top to bottom.

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 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 saving on the n vertex-pages is 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 saving on edge-pages is equal to $\sum_B s_B$ where the sum is over all blocks. In particular, the maximal possible saving on the edge-pages is $(d-1)m$, using the fact that S_I is empty.

We prove that there is a service with the total saving 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.

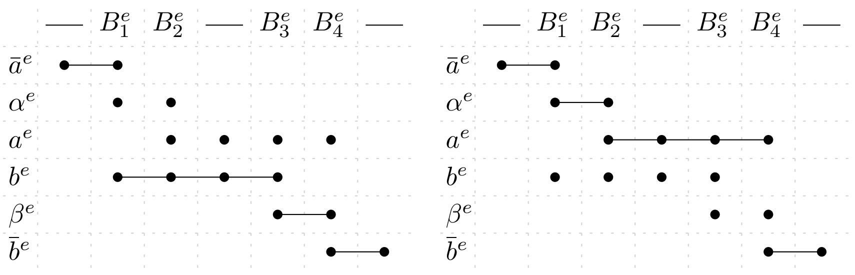

Fig. 8 The two ways of caching in Lemma A.2

Lemma A.2 *Suppose that G has an independent set W of size K. Then there exists a service with the total saving $(d-1)m + K/(n+1)$.*

Proof For any e, denote $e = \{u, v\}$ 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 saving on those pages is $(d-1)m$. See Fig. 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 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 saving at least $(d-1)m$. For a block B, let B' denote the following block.

Claim A.3 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 saving on edge-pages is $\sum_B s_B \leq (d-1)m$. We need an equality. □

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

Claim A.4 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 $k \leq m$

$$s_B^{\leq k} = k. \tag{8}$$

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 (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 (8). □

Claim A.5 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 A.6 *Suppose that there exists a valid service with the total saving* $(d-1)m + K/(n+1)$*. 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 saving implies 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 A.2 and A.6 together show that we constructed a valid polynomial-time reduction from the problem of independent set to general caching. Therefore, Theorem A.1 is proven.