Online Paging with Heterogeneous Cache Slots

Marek Chrobak[1], Samuel Haney[2], Mehraneh Liaee[3], Debmalya Panigrahi[4], Rajmohan Rajaraman[3], Ravi Sundaram[3], and Neal E. Young[1]

[1]University of California at Riverside; Email: marek@cs.ucr.edu, neal.young@ucr.edu
[2]Tumult Labs; Email: sam.m.haney@gmail.com
[3]Northeastern University; Email: {mehraneh, rraj, koods}@ccs.neu.edu
[4]Duke University; Email: debmalya@cs.duke.edu

Abstract

It is natural to generalize the k-Server problem by allowing each request to specify not only a point p, but also a subset S of servers that may serve it. To attack this generalization, we focus on uniform and star metrics. For uniform metrics, the problem is equivalent to a generalization of Paging in which each request specifies not only a page p, but also a subset S of cache slots, and is satisfied by having a copy of p in some slot in S. We call this problem *Slot-Heterogenous Paging*.

We parameterize the problem by specifying an arbitrary family $\mathcal{S} \subseteq 2^{[k]}$, and restricting the sets S to $\mathcal{S}$. If all request sets are allowed ($\mathcal{S} = 2^{[k]}$), the optimal deterministic and randomized competitive ratios are exponentially worse than for standard Paging ($\mathcal{S} = \{[k]\}$). As a function of $|\mathcal{S}|$ and the cache size k, the optimal deterministic ratio is polynomial: at most $O(k^2|\mathcal{S}|)$ and at least $\Omega(\sqrt{|\mathcal{S}|})$. For any laminar family $\mathcal{S}$ of height h, the optimal ratios are $O(hk)$ (deterministic) and $O(h^2 \log k)$ (randomized). The special case that we call *All-or-One Paging* extends standard Paging by allowing each request to specify a specific slot to put the requested page in. For All-or-One Paging the optimal competitive ratios are $\Theta(k)$ (deterministic) and $\Theta(\log k)$ (randomized), while the offline problem is $\mathbb{NP}$-hard. We extend the deterministic upper bound to the *weighted* variant of All-or-One Paging (a generalization of standard Weighted Paging), showing that it is also $\Theta(k)$.

Some of the upper bounds for the laminar case are shown via a reduction to a generalization of Paging in which each request specifies a set P of *pages*, and is satisfied by fetching any page from P into the cache. The optimal ratios for the latter problem (with laminar family of height h) are at most hk (deterministic) and hH_k (randomized).

1 Introduction

It is natural to generalize the k-Server problem to allow each request to specify not only a point p, but also a subset S of servers that may serve it. We call this generalization *Heterogenous* k-Server. To date, only a few special cases of this problem have been studied [44]. Here, following the strategy taken for other hard generalizations of k-Server [6, 7, 11, 22, 29, 38], we attack the problem by focusing on its restriction to uniform and star metrics. For uniform metrics, the problem is equivalent to a variant of Paging in which each request specifies a page p and a subset S of cache slots, to be satisfied by having a copy of p in some slot in S. We call this *Slot-Heterogenous Paging*. For star metrics the problem reduces to a weighted variant, where the cost of retrieving a page is the weight of the page. For reasons discussed below, we parameterize these problems by allowing the requestable sets S to be restricted to an arbitrary but pre-specified family $\mathcal{S} \subseteq 2^{[k]}$. (Restricting to $\mathcal{S} = \{[k]\}$ gives standard Paging and k-Server.) Next is a summary of our results, followed by a summary of related work.

Slot-Heterogenous Paging (Section 3). As we point out, Slot-Heterogenous Paging easily reduces (preserving the competitive ratio) to the so-0 Generalized k-Server problem in uniform metrics, for which upper bounds of $k2^k$ and $O(k^2 \log k)$ on the deterministic and randomized ratios are known [7, 11]. Our Theorems 3.2 (i) and 3.3 show that the optimal deterministic and randomized competitive ratios for Slot-Heterogenous Paging are at least $\Omega(2^k/\sqrt{k})$ and $\Omega(k)$. (The proofs of Theorems 3.2 and 3.3 use, respectively, adversary arguments and a reduction *from* standard Paging with a cache of size $\exp(\Theta(k))$.) Hence, the optimal ratios for Slot-Heterogenous Paging are exponentially worse than for standard Paging.

But these large ratios occur only for instances that use exponentially many distinct request sets S. This motivates us to study the optimal ratios as a function of the family $\mathcal{S}$ of requestable slot sets, mentioned above, and to try to identify natural families that admit more reasonable ratios.

Theorem 3.1 shows that the optimal deterministic ratio is at most $k^2|\mathcal{S}|$ for any family $\mathcal{S}$. Theorem 3.2 (ii) shows a complementary lower bound: for infinitely many families $\mathcal{S}$, every deterministic online algorithm has competitive ratio $\Omega(\sqrt{|\mathcal{S}|})$. Together Theorems 3.1 and 3.2 (ii) imply that, as a function of $|\mathcal{S}|$ and k, the optimal deterministic ratio for Slot-Heterogenous Paging is polynomial.

Slot-Laminar Paging (Section 5). We then consider the specific structure of $\mathcal{S}$, showing better bounds when $\mathcal{S}$ is *laminar*. This case, which we call *Slot-Laminar Paging*, models applications where slot (or server) capabilities are hierarchical. Laminarity implies $|\mathcal{S}| < 2k$, so (per Theorem 3.1 above) the optimal deterministic ratio is $O(k^3)$. Theorem 5.1 shows that the optimal deterministic and randomized ratios are $O(h^2 k)$ and $O(h^2 \log k)$, where $h \leq k$ is the height of $\mathcal{S}$. The proof (described later) is via a reduction to an intermediate problem that we call *Page-Laminar Paging*. Theorem 5.2 tightens the deterministic bound to $O(hk)$. Its proof refines the generic algorithm from Theorem 3.1. The dependence on k in these bounds is asymptotically tight, as Slot-Laminar Paging generalizes standard Paging.

All-or-One Paging (Section 6). *All-or-One Paging* is the restriction of Slot-Laminar Paging (with height $h = 2$) to $\mathcal{S} = \{[k]\} \cup \{\{j\}\}_{j\in[k]}$. That is, only two types of requests are allowed: *general requests* (allowing the requested page to be anywhere in the cache), and *specific requests* (requiring the page to be in a specified slot). Specific requests don't give the algorithm any choice, so may appear easy to handle, but in fact make the problem substantially harder than standard Paging—Theorems 6.1 and 6.3 show that its optimal deterministic and randomized competitive ratios are at least twice those for Paging and that the offline problem is $\mathbb{NP}$-hard. Nonetheless, by Theorem 5.1 the optimal deterministic and randomized ratios are $O(k)$ and $O(\log k)$, respectively, and offline All-or-One Paging can be $O(1)$-approximated in polynomial time. Theorem 6.2 tightens the deterministic upper bound to $3k - 2$.

Weighted All-Or-One Paging (Section 7). *Weighted All-Or-One Paging* extends All-or-One Paging so that each page has a non-negative weight and the cost of each retrieval is the weight of the page instead of 1. Theorem 7.1 shows that the optimal deterministic ratio for Weighted All-Or-One Paging is $O(k)$, matching the ratio for standard Weighted Paging (a.k.a. Weighted Caching) up to a small constant factor. The algorithm in the proof is implicitly a linear-programming primal-dual algorithm. The standard linear program for standard Weighted Paging does not force pages into specific slots. Indeed, doing so makes the natural integer linear program an $\mathbb{NP}$-hard multicommodity-flow problem. (Section 7 has an example that illustrates the challenge.) We augment the linear program to partially model the slot constraints.

Page-Laminar Paging (Section 4). As mentioned above, Theorem 5.1 gives upper bounds for Slot-Laminar Paging by reducing it to an intermediate problem that we call Page-Laminar Paging. This is a natural generalization of Paging in which each request is a set P of *pages* from an arbitrary but fixed laminar family $\mathcal{P}$, and is satisfiable by fetching any page from P into any slot in the cache. Theorem 4.1 shows that the optimal deterministic and randomized ratios for Page-Laminar Paging are at most hk and hH_k, where h is the height of the laminar family and $H_k = \sum_{i=1}^{k} 1/i = \ln k + O(1)$. The proof is by a reduction to standard Paging, which replaces each set request P by a request to one carefully chosen page in P, yielding an instance of Paging, while increasing the optimal cost by at most a factor of h.

Reducing Slot-Laminar Paging to Page-Laminar Paging. Theorem 5.1 reduces Slot-Laminar Paging to Page-Laminar Paging via a relaxation of Slot-Laminar Paging that drops the constraint that each slot holds at most one page, while still enforcing the cache-capacity constraint of k. This relaxed instance is naturally equivalent to an instance of Page-Laminar

problem	set family $\mathcal{S}$ (or $\mathcal{P}$)	deterministic	randomized	where
Slot-Heterogenous Paging	$2^{[k]} \setminus \{\emptyset\}$	$\le k2^k$	$\le O(k^2 \log k)$	via [7, 11]
– ”	arbitrary $\mathcal{S}$	$\le k \min(\lvert\mathcal{S}^*\rvert, \mathsf{mass}(\mathcal{S}))$		Thm. 3.1
– One-of-m Paging, $m \approx k/2$	$\binom{[k]}{m}$	$\ge \Omega(2^k/\sqrt{k})$	$\ge \Omega(k)$	Thms. 3.2(i), 3.3
– One-of-m Paging, any m	$\binom{[k]}{m}$	$\gtrsim \Omega((4k/m)^{m/2}/\sqrt{m})$		Thm. 3.2(ii)
– Slot-Laminar Paging	laminar $\mathcal{S}$, height h	$\le 2\,\mathsf{mass}(\mathcal{S}) \le 2hk$	$\le 3h^2 H_k$	Thms. 5.1, 5.2
– All-or-One Paging	$\{[k]\} \cup \{\{s\} : s \in [k]\}$	$\ge 2k - 1$	$\ge 2H_k - O(1)$	Thm. 6.1
– ”	”	$\le 3k - 2$		Thm. 6.2
Weighted All-Or-One Paging	$\{[k]\} \cup \{\{s\} : s \in [k]\}$	$\le O(k)$		Thm. 7.1
Page-Subset Paging restricted to	$\mathcal{P} = \binom{\text{all pages}}{m}$	$\ge \binom{k+m}{k} - 1$		[29]
”		$\le k(\binom{k+m}{k} - 1)$	$\le O(k^3 \log m)$	[22]
– Page-Laminar Paging	$\mathcal{P}$ laminar, height h	$\le hk$	$\le hH_k$	Thm. 4.1

Table 1: Summary of upper ($\le$) and lower ($\ge$) bounds on optimal competitive ratios. Here $\mathsf{mass}(\mathcal{S}) = \sum_{S\in\mathcal{S}} |S|$ and $\mathcal{S}^* = \bigcup_{S\in\mathcal{S}} 2^S$. The lower bound for One-of-m Paging holds for some but not all m and k—see Theorem 3.2(ii). Also, offline All-or-One Paging and its generalizations are $\mathbb{NP}$-hard (Theorem 6.3), as is offline Page-Subset Paging ([22]).

Paging. The proof then shows how any solution for the relaxed instance can be "rounded" back to a solution for the original Slot-Laminar Paging instance, losing an $O(h)$ factor in the cost and competitive ratio.

Related work. Paging and k-Server have played a central role in the theory of online computation since their introduction in the 1980s [12, 41, 46]. For k-Server, the optimal deterministic ratio is between k and $2k - 1$ [37]. Recent work [27] offers hope for closing this gap, while a recent breakthrough showed that the randomized ratio is $\Theta(\text{polylog}(k))$ ([39], see also [4, 17]). For weighted Paging the optimal ratios are k and $\Theta(\log k)$ [1, 5, 30, 42, 46].

The standard k-Server and Paging models assume homogenous (interchangeable) servers and cache slots. They don't model applications where servers have different capabilities, nor the fact that modern cache systems often partition the slots, sometimes dynamically, with some parts exclusively accessible by specific processors, cores, processes, threads, or page sets (e.g., [28, 40, 47–49]).

Restricted Caching is one previously studied model with *heterogenous* cache slots. It is the restriction of Slot-Heterogenous Paging in which each page p has one *fixed* set $S_p \subseteq [k]$ of slots, and each request to p requires p to be in some slot in S_p. For this problem the optimal randomized ratio is $O(\log^2 k)$ [19]. Better bounds are possible given some further restrictions on the sets, as in *Companion Caching*, which models a cache partitioned into set-associative and fully-associative parts [15, 16, 32, 43]. It is natural to ask whether *Restricted k-Server*—the restriction of Heterogenous k-Server that requires each point p to be served by a server in a *fixed* set S_p— might be easier than Heterogenous k-Server, but in general (specifically, in metric spaces with no isolated points, such as Euclidean spaces) the two problems are equivalent. The $\mathbb{NP}$-hardness result for (offline) Restricted Caching from [16] implies that offline Slot-Heterogenous Paging with $\mathcal{S} = \{\{s, k\} : s \in [k-1]\}$ is $\mathbb{NP}$-hard.

Other sophisticated online caching models include *Snoopy Caching*—in which multiple processors each have their own cache and coordinate to maintain consistency across writes [36], *Multi-Level Caching*—where the cost to access a slot depends on the slot [24], and *Writeback-Aware Caching*—where each page has multiple copies, each with a distinct level and weight, and each request specifies a page and a level, and can be satisfied by fetching a copy of this page at the given or a higher level [8, 9]. (This is a special case of weighted Page-Laminar Paging where $\mathcal{P}$ consists of pairwise-disjoint chains.) *Multi-Core Caching* models the fact that faults can change the request sequence (e.g. [35]).

Patel's master thesis [44] studies Heterogenous k-Server with just two types of requests—general requests (to be served by any server) and specialized requests (to be served by any server in a fixed subset S' of "specialized" servers)—and bounds the optimal ratios for uniform metrics and the line. Recent independent work on deterministic algorithms for online All-or-One Paging establishes a $2k - 1$ lower bound (matching our result) and a $2k + 14$ upper bound (improving our result) [21]. We note that an earlier version of our results on online All-or-One Paging, including the $2k - 1$ lower bound and a $3k$ upper bound on deterministic algorithms, appeared in [34].

Heterogenous k-Server reduces (see Section 3) to the *Generalized k-Server* problem, in which each server moves in its own metric space, each request specifies one point in each space, and the request is satisfied by moving any one server to the point in its space [38]. For uniform metrics, the optimal competitive ratios for this problem are between 2^k and $k2^k$ (deterministic) and between $\Omega(k)$ and $O(k^2 \log k)$ (randomized) [7, 11]. These ratios are exponentially worse than the ratios for standard k-Server. Heterogenous k-Server, parameterized by $\mathcal{S}$, provides a spectrum of problems that bridges the two extremes.

Weighted k-Server is a restriction of Generalized k-Server in which servers move in the same metric space but have different weights, and the cost is the weighted distance [33]. For this problem (in non-uniform metrics) the deterministic and randomized ratios are at least (respectively) doubly exponential [6, 7] and exponential [3, 23].

Page-Subset Paging, restricted to to m-element sets of pages, has been studied as (uniform) *Metrical Service Systems with Multiple Servers* [22, 29]. For this problem the deterministic ratio is at least $\binom{k+m}{k} - 1$ [29], while the randomized ratio is $O(k^3 \log m)$ [22]. The *k-Chasing* problem extends k-Server by having each request P be a convex subset of $\mathbb{R}^d$, to be satisfied by moving any server to any point in P [18]. For k-Chasing, no online algorithm is competitive even for $d = k = 2$ [18], while for $k = 1$ the ratios grow with d [2, 45].

In the *k-Taxi problem* each request (p, q) requires any server to move to p then (for free) to q. For this problem the optimal ratios are exponentially worse than for standard k-Server [20, 26].

2 Formal Definitions

Slot-Heterogenous Paging. A problem instance consists of a set $[k] = \{1, 2, \ldots, k\}$ of cache slots, a family $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$ of requestable slot sets, and a request sequence $\sigma = \{\sigma_t\}_{t=1}^{T}$, where each request has the form $\sigma_t = \langle p_t, S_t \rangle$ for some *page* p_t and set $S_t \in \mathcal{S}$. A solution is a sequence $\{C_t\}_{t=1}^{T}$ such that, at each time $t \in [T]$, the *cache configuration* C_t assigns at most one page to each slot in $[k]$, assigning page p_t to at least one slot in S_t. The objective is to minimize the number of *retrievals*, where a page p is retrieved in slot s at time t if C_t assigns p to s, but C_{t-1} does not (or $t = 1$).

Slot-Laminar Paging. This is the restriction of Slot-Heterogenous Paging to instances where $\mathcal{S}$ is *laminar*: every pair $R, R' \in \mathcal{S}$ of sets is either disjoint or nested. (This implies $|\mathcal{S}| \leq 2k$.) The *height* of $\mathcal{S}$ is the maximum h such that $\mathcal{S}$ contains a sequence of h strictly nested sets: $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_h$.

All-or-One Paging. This is the restriction of Slot-Laminar Paging to instances with $\mathcal{S} = \{[k]\} \cup \{\{j\}\}_{j \in [k]}$. That is, there are two types of requests: *general request* of the form $\langle p, [k] \rangle$, requiring page p to be in at least one slot, and *specific request* of the form $\langle p, \{j\} \rangle$ where $j \in [k]$, requiring page p to be in slot j. For convenience, $\langle p, * \rangle$ is a synonym for $\langle p, [k] \rangle$, while $\langle p, j \rangle$ is a synonym for $\langle p, \{j\} \rangle$.

One-of-m Paging. This is the restriction of Slot-Heterogenous Paging to instances with $\mathcal{S} = \binom{[k]}{m} = \{S \subseteq [k] : |S| = m\}$, that is, every request specifies m slots. This problem plays a key role in our lower bounds.

Weighted Slot-Laminar Paging. This is the natural extension of Slot-Laminar Paging in which each page p is assigned a non-negative weight $\mathsf{wt}(p)$, and the cost of retrieving p is $\mathsf{wt}(p)$ instead of 1.

Page-Subset Paging. An instance consists of k cache slots, a collection $\mathcal{P}$ of requestable sets of pages, and a request sequence $\pi = \{P_t\}_{t=1}^{T}$, where each P_t is drawn from $\mathcal{P}$. A solution is a sequence $\{C_t\}_{t=1}^{T}$ of cache configurations (as previously defined) so that, at each time $t \in [T]$, C_t assigns at least one page in P_t to at least one slot. The objective is to minimize the number of retrievals. (Slots are interchangeable here, so a cache configuration could be defined as a multiset of at most k pages, but using slot assignments is technically more convenient.)

Online algorithms and competitive ratio. We use the standard framework of competitive analysis. An algorithm $\mathbb{A}$ is *c-competitive* for a given cost minimization problem if, for each instance σ, $\mathbb{A}$ satisfies

$$\mathsf{cost}_{\mathbb{A}}(\sigma) \;\leq\; c \cdot \mathsf{opt}(\sigma) + b.$$

In this inequality, $\mathsf{cost}_{\mathbb{A}}(\sigma)$ is the cost of $\mathbb{A}$ on σ, $\mathsf{opt}(\sigma)$ is the optimum cost of σ, and b is a constant independent of σ. The algorithm is assumed to know the underlying set family $\mathcal{S}$ (or $\mathcal{P}$), but many of our algorithms work (or can be adapted to work) without knowing the set family in advance.

3 Slot-Heterogenous Paging

Any instance of Slot-Heterogenous Paging can be reduced to an instance of Generalized k-Server in uniform spaces, as follows. Represent each cache slot by a server in a uniform metric space whose points are the pages, then simulate each request $\langle p, S \rangle$ by a sufficiently long sequence of request vectors that have p in the coordinates in S and alternate between two different points on the remaining coordinates, in $[k] \setminus S$. Composing this reduction with the upper bounds from [7] yields immediate upper bounds of $O(k2^k)$ and $O(k^3 \log k)$ on the deterministic and randomized ratios for unrestricted Slot-Heterogenous Paging (that is, with $\mathcal{S} = 2^{[k]} \setminus \{\emptyset\}$).

Theorems 3.2 (i) (Section 3.2) and 3.3 (Section 3.3), show that these are tight within $\mathrm{poly}(k)$ factors: the optimal ratios are at least $\Omega(2^k/\sqrt{k})$ and $\Omega(k)$, respectively. But restricting $\mathcal{S}$ allows better ratios: Theorem 3.1 (Section 3.1) shows an upper bound of $k^2|\mathcal{S}|$ on the optimal deterministic ratio for any family $\mathcal{S}$. For One-of-m Paging, Theorems 3.1 and 3.2 (ii) (Section 3.2) imply that the optimal deterministic ratio is $O(k^{m+1})$ and $\Omega((4k/m)^{m/2}/\sqrt{m})$.

input: Slot-Heterogenous Paging instance $(k, \mathcal{S}, \sigma = (\sigma_1, \ldots, \sigma_T))$
1. let the initial cache configuration C_0 be arbitrary; let $\ell \leftarrow 1$ — *ℓ is the start of the current phase*
2. for each time $t \leftarrow 1, 2, \ldots, T$:
3.1. if the current configuration C_{t-1} satisfies the current request σ_t: ignore the request (take $C_t = C_{t-1}$)
3.2. else if any configuration satisfies all requests $\sigma_\ell, \sigma_{\ell+1}, \ldots, \sigma_t$: let C_t be any such configuration
3.3. else: let $\ell \leftarrow t$; let C_t be any configuration satisfying σ_t — *start the next phase*

Figure 1: Online algorithm EXHSEARCH for Slot-Heterogenous Paging.

3.1 Upper bounds for deterministic Slot-Heterogenous Paging

This section gives upper bounds on the optimal deterministic competitive ratio for Slot-Heterogenous Paging with any slot-set family $\mathcal{S}$, as a function of $\mathsf{mass}(\mathcal{S}) = \sum_{S\in\mathcal{S}} |S| \le k|\mathcal{S}|$ and $|\mathcal{S}^*| = |\bigcup_{S\in\mathcal{S}} 2^S|$. The first bound follows from an easy counting argument. The second bound uses a refinement of the rank method of [7], which bounds the number of steps of a natural exhaustive-search algorithm by the rank of a certain upper-triangular matrix.

Theorem 3.1. *Fix any $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$. The competitive ratio of Algorithm* EXHSEARCH *in Figure 1 for Slot-Heterogenous Paging with requestable sets from $\mathcal{S}$ is at most $k \cdot \min\{|\mathcal{S}^*|, \mathsf{mass}(\mathcal{S})\}$.*

The theorem implies that the competitive ratio of One-of-m Paging is polynomial in k when m is constant.

Proof of Theorem 3.1. By inspection of the algorithm, the length of each phase does not depend on previous phases. Focus on any one phase. We first bound the length, say L, of the phase. To ease notation and without loss of generality, assume the phase is the first (with $\ell = 1$) and the algorithm faults in each step t, that is C_{t-1} does not satisfy σ_t. (Otherwise first remove such requests; this doesn't change the algorithm's cost or increase the optimal cost.) So the following property holds:

(UT) *For each time $t \in [L]$, configuration C_{t-1} satisfies requests $\sigma_1, \sigma_2, \ldots, \sigma_{t-1}$, but not σ_t.*

Next we observe that Property (UT) implies $L \le \mathsf{mass}(\mathcal{S})$. The property implies that every request σ_t in this phase is distinct. (This is because for any $t' < t$, C_{t-1} satisfies $\sigma_{t'}$ but not σ_t, and the final cache configuration C_L satisfies all of these requests.) For any given set $S \in \mathcal{S}$, there at most $|S|$ distinct pages p such that $\langle p, S\rangle$ is requested in the phase (as C_L has each such page in a distinct slot in S) so the phase has at most $|S|$ requests that use set S. Summing over $S \in \mathcal{S}$:

$$L \;\le\; \textstyle\sum_{S\in\mathcal{S}} |S| \;=\; \mathsf{mass}(\mathcal{S}). \tag{1}$$

The second part of the above argument is tight, in that putting a distinct page in each slot $s \in [k]$ will satisfy $\mathsf{mass}(\mathcal{S})$ distinct requests. Next we show a bound that is tighter for some families $\mathcal{S}$.

Identify each page p with a distinct but arbitrary positive real number.[1] For each cache configuration C_t, let $C_t^i \in \mathbb{R}$ denote the page in slot i, if any, else 0. Define matrix $M \in \mathbb{R}^{L\times L}$ by

$$M_{st} \;=\; \prod_{i\in S_t} (C_{s-1}^i - p_t),$$

so that $M_{st} = 0$ if and only if $C_{s-1}^i = p_t$ for some $i \in S_t$, that is, if and only if C_{s-1} satisfies σ_t. So Property (UT) implies that M is upper-triangular and non-zero on the diagonal. So M has rank L.

Expanding the formula for M_{st}, we obtain

$$M_{st} \;=\; \sum_{S\subseteq S_t} \Big(\prod_{i\in S} C_{s-1}^i\Big)\cdot\Big(\prod_{i\in S_t\setminus S} -p_t\Big) \;=\; \sum_{S\subseteq S_t} \Big(\prod_{i\in S} C_{s-1}^i\Big)\cdot(-p_t)^{|S_t|-|S|} \;=\; \sum_{S\in\mathcal{S}^*} A_{sS}\cdot B_{St}\,,$$

where $\mathcal{S}^* = \bigcup_{S\in\mathcal{S}} 2^S$ and matrices $A \in \mathbb{R}^{L\times\mathcal{S}^*}$ and $B \in \mathbb{R}^{\mathcal{S}^*\times L}$ are defined by

$$A_{sS} \;=\; \prod_{i\in S} C_{s-1}^i \quad\text{and}\quad B_{St} \;=\; \begin{cases} (-p_t)^{|S_t|-|S|} & \text{if } S \subseteq S_t \\ 0 & \text{otherwise.}\end{cases}$$

That is, $M = AB$, where A and B (and therefore M) have rank at most $|\mathcal{S}^*|$. And M has rank L, so

$$L \;\le\; |\mathcal{S}^*|. \tag{2}$$

[1] We use $\mathbb{R}$, but the argument could be made over any sufficiently large field.

Consider any phase other than the last. Let t' and t'' be the start and end times. Suppose for contradiction that the optimal solution incurs no cost (has no retrievals) during $[t'+1, t''+1]$. Then its configuration at time t' satisfies all requests in $[t', t''+1]$, contradicting the algorithm's condition for terminating the phase. So the optimal solution pays at least 1 per phase (other than the last). In any phase of length L the algorithm pays at most kL (at most k per step). This and bounds (2) and (1) imply Theorem 3.1. □

Theorem 3.1 can be strengthened slightly by making the algorithm retrieve at most $\min(t,k)$ pages for the tth request in each phase.

3.2 Lower bounds for deterministic Slot-Heterogenous Paging

This section proves the lower bounds for Slot-Heterogenous Paging and One-of-m Paging given in Table 1.

Theorem 3.2. *(i) For all odd k, the optimal deterministic ratio for One-of-m Paging with $m = (k+1)/2$ is at least $\binom{k}{m} = \Omega(2^k/\sqrt{k})$. For all k the optimal ratio with $m = \lfloor (k+1)/2 \rfloor$ is $\Omega(2^k/\sqrt{k})$.*

(ii) For any even $m \geq 2$ and any $k > m$ that is an odd multiple of $m-1$, the optimal deterministic ratio for One-of-m Paging is at least $\binom{m-1}{m/2}\left(\frac{k}{m-1}\right)^{m/2} = \Theta((4k/m)^{m/2}/\sqrt{m}) = \Omega(\sqrt{|\mathcal{S}|})$, where $\mathcal{S} = \binom{[k]}{m}$.

Before proving Theorem 3.2, we prove a utility lemma (Lemma 3.1). It states that for any $\mathcal{S}$ the existence of a family $\mathcal{Z} \subseteq 2^{[k]}$ with certain properties implies a lower bound of $|\mathcal{Z}|$ on the competitive ratio. The proof of the theorem then constructs such families $\mathcal{Z}$ for appropriate families $\mathcal{S}$ of requestable sets.

Throughout this section $\overline{X}$ denotes the complement of the set $X \subseteq [k]$, that is $\overline{X} = [k] \setminus X$.

Lemma 3.1. *For some $\mathcal{S} \subseteq 2^{[k]}$, suppose there are set families $\mathcal{G} \subseteq \mathcal{S}$ and $\mathcal{Z} \subseteq 2^{[k]}$ such that*

(gz0) For each $X \subseteq [k]$ there is $S \in \mathcal{G}$ such that $S \subseteq X$ or $S \subseteq \overline{X}$.

(gz1) If $Z \in \mathcal{Z}$ then $\overline{Z} \notin \mathcal{Z}$.

(gz2) For each $S \in \mathcal{G}$ there is $Y \in \mathcal{Z}$ such that $S \not\subseteq Z$ and $S \not\subseteq \overline{Z}$ for all $Z \in \mathcal{Z} \setminus \{Y\}$.

Then the optimal deterministic competitive ratio for Slot-Heterogenous Paging with family $\mathcal{S}$ is at least $|\mathcal{Z}|$.

Proof. The proof is an adversary argument based on the following idea. At each step, the adversary chooses a request that forces the algorithm to fault but causes at most two faults total among a fixed set of $2|\mathcal{Z}|$ other solutions. At the end, the algorithm's total cost is at least $|\mathcal{Z}|$ times the average cost of these other solutions, so its competitive ratio is at least $|\mathcal{Z}|$. This general approach is common for lower bounds on deterministic online algorithms (see e.g. lower bounds on the optimal ratios for k-Server [41], for Metrical Task Systems [14] and for Generalized k-Server on uniform metrics [38]).

Here are the details. Let $\mathbb{A}$ be any deterministic online algorithm for Slot-Heterogenous Paging with slot-set family $\mathcal{S}$. The adversary will request just two pages, p_0 and p_1. For a set $X \subseteq [k]$, let C_X denote the cache configuration where the slots in X contain p_0 and the slots in $\overline{X}$ contain p_1. Without loss of generality assume that each slot of $\mathbb{A}$'s cache always holds p_0 or p_1—its cache configuration is C_X for some X.

At each step, if the current configuration of $\mathbb{A}$ is C_X, the adversary chooses $S \in \mathcal{G}$ such that either $S \subseteq X$ or $S \subseteq \overline{X}$. (Such an S exists by Property (gz0).) If $S \subseteq X$, then all slots in S hold p_0, and the adversary requests $\langle p_1, S\rangle$, causing a fault. Otherwise, $S \subseteq \overline{X}$, so all slots in S hold p_1. In this case the adversary requests $\langle p_0, S\rangle$, causing a fault. The adversary repeats this K times, where K is arbitrarily large. Since $\mathbb{A}$ faults at each step, the overall cost of $\mathbb{A}$ is at least K.

It remains to bound the optimal cost. Let $\widetilde{\mathcal{Z}} = \{\overline{Z} : Z \in \mathcal{Z}\}$. By Property (gz1), we have $\widetilde{\mathcal{Z}} \cap \mathcal{Z} = \emptyset$. For each $Z \in \mathcal{Z} \cup \widetilde{\mathcal{Z}}$ define a solution called the *Z-strategy*, as follows. The solution starts in configuration C_Z. It stays in C_Z for the whole computation, except that on requests $\langle p_a, S\rangle$ that are not served by C_Z (that is, when all slots of C_Z in S contain p_{1-a}), it retrieves p_a to any slot $j \in S$, serves the request, then retrieves the page p_{1-a} back into slot j, restoring configuration C_Z. This costs 2.

We next observe that in each step at most one Z-strategy faults (and pays 2). Assume that the request at a given step is to p_0 (the case of a request to p_1 is symmetric). Let this request be $\langle p_0, S\rangle$, where $S \in \mathcal{G}$. Let $Y \subseteq [k]$ be the set from Property (gz2). For all $Z \in (\mathcal{Z} \cup \widetilde{\mathcal{Z}}) \setminus \{Y, \overline{Y}\}$, then, $S \cap Z \neq \emptyset$, implying that configuration C_Z has a slot in S that contains p_0—in other words, configuration C_Z satisfies S. Also, either $S \cap Y \neq \emptyset$ or $S \cap \overline{Y} \neq \emptyset$, so one of the configurations C_Y or $C_{\overline{Y}}$ also satisfies S. Therefore only one Z-strategy (Y or $\overline{Y}$) might not satisfy S. So, in each step, at most one Z-strategy faults (and pays 2).

Thus the combined total cost for all Z-strategies (not counting the cost of moving to Z at the beginning) is at most $2K$. There are $2|\mathcal{Z}|$ such strategies, so their average cost is at most $(2K+k)/2|\mathcal{Z}|$. The cost of $\mathbb{A}$ is at least K, so the ratio is at least

$$\frac{K}{(2K+k)/2|\mathcal{Z}|} = \frac{|\mathcal{Z}|}{1+k/2K}.$$

Taking K arbitrarily large, the lemma follows. □

Now we use Lemma 3.1 to prove Parts (i) and (ii) of Theorem 3.2.

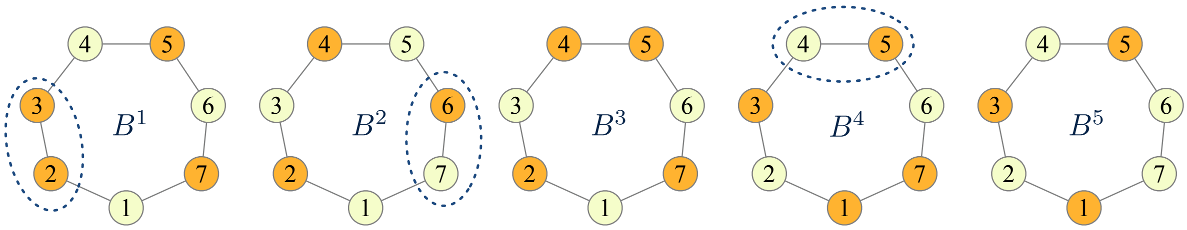

Figure 2: Illustration of the proof of Theorem 3.2 Part (ii) for $k = 35$, $m = 6$, and $\ell = 7$. The figure shows the partition of all slots into $m-1 = 5$ sets $B^1, \ldots, B^5$, each represented by a cycle. To avoid clutter, each slot b^e_c is represented by its index c within B^e. The picture shows set $S = \{b^1_2, b^1_3, b^2_6, b^2_7, b^4_4, b^4_5\} \in \mathcal{G}$, marked by dashed ovals. It also shows $Z_{S'} \in \mathcal{Z}$, represented by orange/shaded circles, for $S' = \{b^1_2, b^1_3, b^3_4, b^3_5, b^4_7, b^4_1\}$.

Proof (Theorem 3.2). Part (i). Recall $m = \lfloor (k+1)/2 \rfloor$. First consider the case that that k is odd. Apply Lemma 3.1, taking both $\mathcal{G}$ and $\mathcal{Z}$ to be $\binom{[k]}{m}$. Properties (gz0) and (gz1) follow directly from k being odd and the definitions of $\mathcal{G}$ and $\mathcal{Z}$. Property (gz2) also holds with $Y = S$. (For any $S \in \mathcal{G}$, every $Z \in \mathcal{Z}$ has size $|Z| = |S| > |\overline{Z}|$, so $S \not\subseteq \overline{Z}$, while $S \subseteq Z$ implies $Z = S$.) So by Lemma 3.1 the ratio is at least $|\mathcal{Z}| = \binom{k}{(k+1)/2} = \Omega(2^k/\sqrt{k})$. This proves Part (i) for odd k. For even k, apply the Part (i) for odd k to $k' = k-1$, using just cache slots in $[k']$, that is, using slot-set family $\mathcal{S}' = \binom{[k']}{m} \subseteq \binom{[k]}{m} = \mathcal{S}$, with slot k playing no role as it is never requested.

Part (ii). Fix such an m and k. Let $\ell = k/(m-1)$ so $\ell \geq 3$ is odd. Recall that $\mathcal{S} = \binom{[k]}{m}$ is the family of requestable slot sets. Partition $[k]$ arbitrarily into $m-1$ disjoint subsets $B^1, B^2, \ldots, B^{m-1}$, each of cardinality ℓ. For each B^e, order its slots arbitrarily as $B^e = \{b^e_1, b^e_2, \ldots, b^e_\ell\}$. For any index $c \in \{1, 2, \ldots, \ell\}$ and an integer i, let $c \oplus i$ denote $((c+i-1) \bmod \ell) + 1$. In other words, we view each B^e as an odd-length cycle, and this cyclic structure is important in the proof. Any consecutive pair $\{b^e_c, b^e_{c\oplus 1}\}$ of slots on this cycle is called an *edge*. Thus each cycle B^e has ℓ edges.

First we define $\mathcal{G} \subseteq \mathcal{S}$ for Lemma 3.1. The sets S in $\mathcal{G}$ are those obtainable as follows: choose any $m/2$ edges, no two from the same cycle, then let S contain the m slots in those $m/2$ chosen edges. (The six slots inside the three dashed ovals in Figure 2 show one S in $\mathcal{G}$.) This set of $m/2$ edges uniquely determines S, and vice versa.

We verify that $\mathcal{G}$ has Property (gz0) from Lemma 3.1. Indeed, consider any $X \subseteq [k]$. Call the slots in X *white* and the slots in $\overline{X}$ *black*. Each cycle B^e has odd length, so has an edge $\{b^e_c, b^e_{c\oplus 1}\}$ that is white (with two white slots) or black (with two black slots). So either (i) at least half the cycles have a white edge, or (ii) at least half have a black edge. Consider the first case (the other is symmetric). There are $m-1$ cycles, and m is even, so at least $m/2$ cycles have a white edge. So there are $m/2$ white edges with no two in the same cycle. The set S comprised of the m white slots from those edges is in $\mathcal{G}$, and is contained in X (because its slots are white). So $\mathcal{G}$ has Property (gz0).

Next we define $\mathcal{Z} \subseteq 2^{[k]}$ for Lemma 3.1. The set $\mathcal{Z}$ contains, for each set $S' \in \mathcal{G}$, one set $Z_{S'}$, defined as follows. For each of the $m/2$ cycles B^e having an edge $\{b^e_c, b^e_{c\oplus 1}\}$ in S', add to $Z_{S'}$ the two slots on that edge, together with the $(\ell-3)/2$ slots $b^e_{c\oplus 3}, b^e_{c\oplus 5}, \ldots, b^e_{c\oplus(\ell-2)}$. For each of the $m/2-1$ remaining cycles B^e, add to $Z_{S'}$ the $(\ell-1)/2$ slots $b^e_1, b^e_3, \ldots, b^e_{\ell-2}$. (The orange/shaded slots in Figure 2 show one set $Z_{S'}$ in $\mathcal{Z}$.) Then $Z_{S'}$ contains exactly $m/2$ edges (the ones in S') while its complement $\overline{Z}_{S'}$ contains exactly $m/2-1$ edges (one from each cycle with no edge in S'). This implies Property (gz1). Note that $Z_{S'} \neq Z_{S''}$ for different sets $S', S'' \in \mathcal{G}$.

Next we show Property (gz2). Given any set $S \in \mathcal{G}$, let $Y = Z_S \in \mathcal{Z}$. Consider any $Z_{S'} \in \mathcal{Z}$ such that $S \subseteq Z_{S'}$ or $S \subseteq \overline{Z}_{S'}$. We need to show $Z_{S'} = Z_S$, i.e., $S' = S$. It cannot be that $S \subseteq \overline{Z}_{S'}$, because S contains $m/2$ edges, whereas $\overline{Z}_{S'}$ contains $m/2-1$ edges. So $S \subseteq Z_{S'}$. But S and $Z_{S'}$ each contain exactly $m/2$ edges, which therefore must be the same. It follows from the definition of $Z_{S'}$ that $S' = S$. So Property (gz2) holds.

So $\mathcal{G}$ and $\mathcal{Z}$ have Properties (gz0)-(gz2) from Lemma 3.1. Directly from definition we have $|\mathcal{Z}| = |\mathcal{G}|$, while $|\mathcal{G}| = \binom{m-1}{m/2}\ell^{m/2}$ because there are $\binom{m-1}{m/2}$ ways to choose $m/2$ distinct cycles, and then for each of these $m/2$ cycles there are ℓ ways to choose one edge. Lemma 3.1 and $\ell = k/(m-1)$ imply that the optimal deterministic ratio is at least $f(m,k) = \binom{m-1}{m/2}(k/(m-1))^{m/2}$.

To complete the proof of part (ii) we lower-bound $f(m,k)$. We start by observing that

$$4^m \;=\; \Omega(\sqrt{m}\,(k/(k-m))^{k-m+1/2}). \tag{3}$$

This can be verified by considering two cases: If $k \geq m+2$ then, using $1+z \leq \mathrm{e}^z$, we have $\sqrt{m}\,(k/(k-m))^{k-m+1/2} = \sqrt{m}(1+m/(k-m))^{k-m+1/2} \leq \sqrt{m} \cdot \mathrm{e}^{5m/4} \leq 2 \cdot 4^m$, for all $m \geq 1$. In the remaining case, for $k = m+1$, we have $\sqrt{m}(k/(k-m))^{k-m+1/2} = \sqrt{m}(1+m)^{3/2} \leq 2 \cdot 4^m$. Thus (3) indeed holds.

Now, recalling that $f(m,k) = \binom{m-1}{m/2}(k/(m-1))^{m/2}$, we proceed in two steps. First,

$$\begin{aligned} f(m,k) \;&=\; \Theta\big((2^m/\sqrt{m}) \cdot (k/(m-1))^{m/2}\big) && (\textit{Stirling's approximation}) \\ &=\; \Theta\big((4k/m)^{m/2} \cdot (1+1/(m-1))^{m/2}/\sqrt{m}\big) && (\textit{rewriting}) \\ &=\; \Theta\big((4k/m)^{m/2}/\sqrt{m}\big) && (\textit{using } (1+1/(m-1))^{m/2} \leq \mathrm{e}) \end{aligned} \tag{4}$$

This gives us the first estimate on the competitive ratio in Theorem 3.2(ii). To obtain the second estimate, squaring both sides of (4), we obtain

$$\begin{aligned} f(m,k)^2 &= \Omega\big((4k/m)^m/m\big) = \Omega\big((k/m)^m \cdot 4^m/m\big) \\ &= \Omega\big((k/m)^m \cdot (k/(k-m))^{k-m+1/2}/\sqrt{m}\big) && (\textit{using } (3)) \\ &= \Omega(\tbinom{k}{m}) = \Omega(|\mathcal{S}|) && (\textit{Stirling's approximation}) \end{aligned}$$

Therefore $f(m,k) = \Omega(\sqrt{|\mathcal{S}|})$, as claimed, completing the proof of Theorem 3.2(ii). □

3.3 Lower bound for randomized Slot-Heterogenous Paging

Next we show a lower bound on the optimal competitive ratio for randomized algorithms:

Theorem 3.3. *The optimal randomized ratio for One-of-m Paging with $m = \lfloor k/2 \rfloor$ is $\Omega(k)$.*

The proof is by a reduction from standard Paging with some N pages and a cache of size $N-1$. For any N, this problem has optimal randomized competitive ratio $H_{N-1} = \Theta(\log N)$ [31]. This and the next lemma imply the theorem.

Lemma 3.2. *Every $f(k)$-approximation algorithm $\mathbb{A}$ for One-of-m Paging with $m = \lfloor k/2 \rfloor$ can be converted into an $O(f(k))$-approximation algorithm $\mathbb{B}$ for standard Paging with N pages and a cache of size $N-1$, where $N = \exp(\Theta(k))$, preserving the following properties: being polynomial time, online, and/or deterministic.*

Proof. Fix a sufficiently large k. Assume without loss of generality that k is even (otherwise apply the construction below to slots in $[k-1]$, ignoring slot k as it is never requested). Take $N = \lfloor \exp(k/16) \rfloor$.

To ease exposition, view the Paging problem with N pages and a cache of size $N-1$ as the following equivalent online *Cat and Rat* game on any set $\mathcal{H}$ of N *holes* (see e.g. [13, §11.3]). The input is a sequence $\mu = (R_0, C_1, \ldots, C_T)$ of holes (i.e., $R_0 \in \mathcal{H}$ and $C_t \in \mathcal{H}$ for all t). A solution is any sequence $(R_1, \ldots, R_T)$ of holes such that $R_t \neq C_t$ for all $t \in [T]$. Informally, at each time $t \in [T]$, the cat inspects hole C_t, and if the rat's hole R_{t-1} at time $t-1$ was C_t, the rat is required to move to some other hole $R_t \in \mathcal{H} \setminus \{C_t\}$. (For the solution to be online, R_t must be independent of $C_{t+1}, C_{t+2}, \ldots, C_T$ for all $t \geq 0$.) The goal is to minimize the number of times the rat moves, $|\{t \in [T] : R_t \neq R_{t-1}\}|$.

Reduce a given instance μ on a set $\mathcal{H}$ of N holes to an instance σ of One-of-$(k/2)$ Paging as follows. Use just two pages, p_0 and p_1. Call a cache configuration C *balanced* if $k/2$ of its cache locations contain p_0 and the others contain p_1. Recall $N = \lfloor \exp(k/16) \rfloor$. Fix any collection $\mathcal{B}$ of N balanced configurations such that the Hamming distance between every two distinct configurations in $\mathcal{B}$ is at least $k/16$. The existence of $\mathcal{B}$ can be shown using a greedy construction (as in [11]) or by a probabilistic proof: if one forms $\mathcal{B}$ by randomly sampling N times with replacement uniformly from the balanced configurations, then by a standard Chernoff bound and the naive union bound $\mathcal{B}$ has these properties with positive probability.

Claim 3.3. *For each configuration $C \in \mathcal{B}$, the set $\mathcal{B} \setminus \{C\}$ has a* forcing sequence *$F(C)$: a request sequence such that the cache configurations that satisfy all requests in it without cost are exactly those in $\mathcal{B} \setminus \{C\}$.*

To verify the claim, take $F(C)$ to be the sequence formed by (any ordering of) all those allowed requests that are satisfied by all configurations in $\mathcal{B}' = \mathcal{B} \setminus \{C\}$ (that is, all requests $\langle p_i, S \rangle$ such that $i \in \{0,1\}$ and $|S| = k/2$, with $S \cap S_i \neq \emptyset$ for all $C(S_0, S_1) \in \mathcal{B}'$). Each configuration in $\mathcal{B}'$ (by definition) satisfies all requests in $F(C)$, so we only need show the converse. Consider any configuration $C(S'_0, S'_1)$ not in $\mathcal{B}'$. We will show that there is a request $\langle p_i, S \rangle$ that isn't satisfied by $C(S'_0, S'_1)$ but is in $F(C)$, that is, it is satisfied by all configurations in $\mathcal{B}'$. In the case that $C(S'_0, S'_1)$ is balanced, by Observation 3.4, below, the request $\langle p_1, S'_0 \rangle$ suffices.

Observation 3.4. *Any request $\langle p_1, S \rangle$ (with $|S| = k/2$) is satisfied by every balanced configuration except $C(S, \overline{S})$.*

Next consider the case that $C(S'_0, S'_1)$ is not balanced. Assume without loss of generality that $|S'_0| > k/2$. (The other case is symmetric.) Let B_0 and B'_0 be any two size-$k/2$ subsets of S'_0 such that B_0 and B'_0 are at Hamming-distance 1 from each other. Then $C(B_0, \overline{B}_0)$ and $C(B'_0, \overline{B}'_0)$ are at Hamming distance 2, so cannot both be in $\mathcal{B}$ (using here the assumption k is sufficiently large). Assume without loss of generality that $C(B_0, \overline{B}_0)$ is not in $\mathcal{B}$. Then the request $\langle p_1, B_0 \rangle$ isn't satisfied by $C(S'_0, S'_1)$ (as $B_0 \subseteq S'_0$) but is satisfied by every configuration in $\mathcal{B}$, just because (by Observation 3.4) the only balanced configuration that doesn't satisfy the request is $C(B_0, \overline{B}_0)$, which is not in $\mathcal{B}$. This shows the claim.

Assume without loss of generality (by identifying holes with configurations) that $\mathcal{H} = \mathcal{B}$. Obtain σ from π as follows: for each time $t \in [T]$, replace the request C_t in π by $F(C_t)^k$, that is, k repetitions of the sequence that forces $\mathcal{B} \setminus \{C_t\}$.

Given an optimal solution $(R_1, \ldots, R_T)$ for $\mu = (R_0, C_1, \ldots, C_T)$, consider the corresponding solution D for σ that starts in configuration R_0, then, for each $t \in [T]$, responds to $F(C_t)^k$ by having its cache in configuration $R_t \in \mathcal{B} \setminus \{C_t\}$ for all requests in $F(C_t)^k$. For each $t \in [T]$, its response to $F(C_t)^k$ costs D nothing if $R_{t-1} = R_t$ (the rat didn't move) and otherwise costs at most k (to transition the cache from R_{t-1} to R_t). This shows $\mathsf{opt}(\sigma) \leq k\, \mathsf{opt}(\mu)$.

Now consider any $f(k)$-approximate solution D for σ. For each $t \in [T]$, as D responds to $F(C_t)^k$, it either incurs cost at least k, or uses at least one configuration (denote it R_t) in $\mathcal{B} \setminus \{C_t\}$. Assume without loss of generality that only the latter case occurs (otherwise modify D to move into any configuration in $\mathcal{B} \setminus \{C_t\}$ at the end of its response to $F(C_t)^k$; these modifications at most double D's cost), let R_0 be D's starting configuration, and define $D' = (R_1, \ldots, R_T)$. This D' is a valid solution to μ, because $R_t \in \mathcal{B} \setminus \{C_t\}$ for $t \in [T]$. Whenever the rat moves (i.e., $R_{t-1} \neq R_t$), by the definition of $\mathcal{B}$, the Hamming distance between R_{t-1} and R_t is $\Omega(k)$, so D paid $\Omega(k)$ to transition from R_{t-1} to R_t (possibly in multiple steps). It follows that $\mathsf{cost}(D') = O(\mathsf{cost}(D)/k) = O(f(k)\mathsf{opt}(\sigma)/k) = O(f(k)\mathsf{opt}(\mu))$ (as D is an $f(k)$-approximation and $\mathsf{opt}(\sigma) \leq k\,\mathsf{opt}(\mu)$). That is, D' is an $O(f(k))$-approximate solution for μ.

Finally, the solution D' for μ can be computed in polynomial time from D, and is online and/or deterministic provided D is. This proves the lemma. ☐

4 Page-Laminar Paging

Recall that Page-Laminar Paging generalizes Paging by allowing each request to be a set P of pages. The request P is satisfiable by having any page $p \in P$ in the cache. We require $P \in \mathcal{P}$, where $\mathcal{P}$ is a pre-specified laminar collection of sets of pages. This section shows the following upper bounds for Page-Laminar Paging. Throughout this section h denotes the height of $\mathcal{P}$.

Theorem 4.1. *Page-Laminar Paging admits the following polynomial-time algorithms: an hk-competitive deterministic online algorithm, an hH_k-competitive randomized online algorithm, and an offline h-approximation algorithm.*

The proof is by reduction to standard Paging. Known polynomial-time algorithms for standard Paging include an optimal offline algorithm [10], a deterministic k-competitive online algorithm [46] and a randomized H_k-competitive online algorithm [1]. Theorem 4.1 follows directly from composing these known results with the following reduction of Page-Laminar Paging to standard Paging:

Lemma 4.1. *Every $f(k)$-approximation algorithm $\mathbb{A}$ for Paging can be converted into an $hf(k)$-approximation algorithm $\mathbb{B}$ for Page-Laminar Paging, preserving the following properties: being polynomial-time, online, and/or deterministic.*

Proof. Let $\mathbb{A}$ be any (possibly online, possibly randomized) $f(k)$-approximation algorithm for Paging. Let Page-Laminar Paging instance π be the input to algorithm $\mathbb{B}$. For any set $P \in \mathcal{P}$, let $c_t(P)$ denote the child of P whose subtree contains π's most recent request to a proper descendant of P. This is the child c of P such that $P_{t'} \subseteq c$, where $t' = \max\{i \leq t : P_i \subset P\}$. If there is no such request (t' is undefined), then define $c_t(P) = P$. Define $p_t(P)$ inductively via $p_t(P) = p_t(c_t(P))$ when $c_t(P) \neq P$, and otherwise $p_t(P)$ is an arbitrary (but fixed) page in P. Call $c_t(P)$ and $p_t(P)$ the *preferred child* and *preferred page* of P at time t. At any time, P's preferred page can be found by starting at P and tracing the path down through preferred children.

Define a Paging instance σ from the given instance π by replacing each request P_t in π by its preferred page $p_t(P_t)$ (so $\sigma_t = p_t(P_t)$). Algorithm $\mathbb{B}$ just simulates Paging algorithm $\mathbb{A}$ on input σ, and maintains its cache exactly as $\mathbb{A}$ does. (Note that σ can be computed online, deterministically, in polynomial time.) Algorithm $\mathbb{B}$ is correct because any solution to σ is also a solution to π (because $\sigma_t = p_t(P_t) \in P_t$). And $\mathsf{cost}(\mathbb{B}(\pi)) = \mathsf{cost}(\mathbb{A}(\sigma))$. To finish proving the lemma, we show $\mathsf{opt}(\sigma) \leq h\,\mathsf{opt}(\pi)$.

For any requested set P, define a *P-phase* of π to be a maximal contiguous interval $[i, j] \subseteq [1, T]$ such that $\sigma_t \not\subset P$ for $t \in [i + 1, j]$. The P-phases for a given P partition $[1, T]$. Each P-phase $[i, j]$ (except possibly the first, with $i = 1$) starts with a request to a proper descendant of P, but there are no such requests during $[i + 1, j]$. It follows that $c_i(P)$ and $p_i(P)$ remain the preferred child and page of P throughout the phase, that the preferred child $c_i(P)$ of P also has the same preferred page $p_i(P)$ throughout the phase, and that $[i, j]$ is contained in some $c_i(P)$-phase. By definition of σ, each request to P in the given instance π is replaced in σ by a request to P's preferred page, $p_i(P)$.

Let $C = (C_1, \ldots, C_T)$ be an optimal solution for π. Figure 3 gives a "repair" algorithm that incrementally modifies C and π, phase by phase, maintaining the invariant that the current solution, denoted C', is always correct for the current instance, denoted π'. (In π', some requests will be to a page, rather than a set. Any such requests is satisfied only by having the requested page in the cache.) At the end the modified instance π' will equal σ, so that the modified solution C' will be a correct solution for σ.

Specifically, we will show the following claim (whose proof we postpone):

Claim 4.2. *The repair algorithm maintains the invariant that the current solution C' is correct for the current instance π', so at termination C' is a correct solution for σ.*

Next we bound the cost, as follows. Call a phase *costly* if its repair increases the cost of C', and *free* otherwise. We show that the number of costly phases is at most $(h - 1)\mathsf{cost}(C)$, and that the repair of each phase increases the cost of C' by at most 1. This implies that the final cost of C' is at most $\mathsf{cost}(C) + (h - 1)\mathsf{cost}(C) = h\,\mathsf{cost}(C)$, as desired. Specifically, we will show the following claims (whose proofs we postpone):

Claim 4.3. *For any requested set P, the repair of any P-phase $[i, j]$ increases the cost of C' by at most 1, and only if $j \neq T$.*

1. Initialize the current instance π' and current solution C' to the given instance π and its solution C.
2. Incrementally modify π' and C' by *repairing* each phase, as follows.
3. While there is an unrepaired phase, choose any unrepaired P-phase $[i, j]$ such that all proper descendants of P have already been repaired, then repair the chosen phase as follows:

4.1. Modify the current instance π' by replacing each request to P during $[i, j]$ in π' by a request to P's preferred page $p_i(P)$. (So, after all phases are repaired, the current instance π' will equal σ.)

4.2. Modify the current solution C' during $[i, j]$ accordingly, to ensure that C' continues to satisfy π'. To do that, we will establish a stronger property throughout $[i, j]$, namely: *whenever C' has at least one page in P cached, C' has P's preferred page cached.*

Let $c = c_i(P)$ and $p = p_i(P)$ be the preferred child and page of P at the start of the phase. Recall that throughout the entire phase c remains P's preferred child, while p remains the preferred page of both P and c. (In the case that no proper descendant of P has yet been requested, $c = P$.)

Say that time $t \in [i, j]$ *needs repair* if, at time t, C' caches at least one page in P, but not p.

For $t \leftarrow i, i+1, \ldots, j$, if time t needs repair, modify what C' caches at time t by replacing one of its currently cached pages from P, say q_t, by p, defining q_t greedily as follows

$$q_t = \begin{cases} q_{t-1} & \textit{if } q_{t-1} \textit{ is defined and still cached at time } t \\ \textit{any page in } P \textit{ cached at time } t & \textit{otherwise.} \end{cases}$$

This completes the repair of this P-phase $[i, j]$. The algorithm terminates after it has repaired all phases.

Figure 3: The algorithm that transforms (π, C) into (σ, C'), by repairing each phase.

Claim 4.4. *For any non-leaf set P, the number of costly P-phases is at most the cost paid by C for pages in P (that is, the number of retrievals of pages in P by C).*

Each leaf set P has only one P-phase $[1, T]$ (by definition of P-phase), so by Claim 4.3 only non-leaf sets have costly phases. Each page p is in at most $h-1$ non-leaf sets P, so Claim 4.4 implies that the total number of costly phases is at most $(h-1)\mathsf{cost}(C)$. This and Claim 4.3 imply that final cost of C' is at most $h\,\mathsf{cost}(C) = h\,\mathsf{opt}(\pi)$, proving Lemma 4.1.

It remains only to prove the three claims.

Proof of Claim 4.2. The invariant holds initially when $C' = C$ and $\pi' = \pi$, just because by definition C is an (optimal) solution for π. Suppose the invariant holds just before the repair of some P-phase $[i, j]$. We will show that it continues to hold after. The repair modifies π' by replacing each request to P during $[i, j]$ by a request to its preferred page $p = p_i(P)$. Consider any time $t \in [i, j]$. First consider the case that (before the repair) π' requested P at time t. In this case, C' cached some page in P, so (by the definition of the repair), after the repair C' has the preferred page p cached, so satisfies the modified request (for p). In the remaining case the repair doesn't modify the request at time t (by π'). In this case, either the repair doesn't modify the cache at time t (in which case C' continues to satisfy π'), or the repair replaces some page q_t in the cache by p. If that happens, the page q_t is also in P. Also, the request in π at time t cannot be to a proper descendant of P (by definition of P-phase) so the request in π' at time t is either to an ancestor of P, a set disjoint from P, or (if already repaired) a page not in P. (We use here that no proper ancestors of P have yet had their phases repaired.) In all three cases, after swapping p for q_t (with $p, q_t \in P$), the request must still be satisfied. So the invariant holds after the repair. At termination the invariant holds so C' is correct for σ. □

Proof of Claim 4.3. Consider the repair of any P-phase $[i, j]$ for any requested set P. This repair modifies the cache only at times in $t \in [i, j]$. Recall that the cost of C' at time t is the number of retrievals at time t, where a retrieval is a page that C' caches at time t but not at time $t-1$.

At each time $t \in [i, j]$ that needed repair (as defined in the algorithm), the repair of the phase replaced some page q_t in the cache at time t by the preferred page p. This can increase the cost only at times in $[i, j+1]$, and by at most 1 at each such time.

We claim the cost cannot increase at time i. Indeed, in the case that $i = 1$, the cost at time i equals the number of pages cached at time i, which a repair at time 1 doesn't change. In the case that $i > 1$, at time i no repair is done, because π requests a proper descendant d of P, and the d-phase containing $[i, j]$ has already been repaired, so π' requests $p_i(d)$ at time i, so (using the invariant) C' already caches $p_i(d)$ at time i. But by definition $p_i(P) = p_i(d)$, so C' already caches P's preferred page at time i. So the cost cannot increase at time i.

So consider any time $t \in [i+1, j]$. To prove the claim, we show that the repair didn't increase the cost at time t. The repair can introduce up to two new retrievals at time t: a new retrieval of p, and/or a new retrieval of q_{t-1}. We show that, for each retrieval that the repair introduced, it removed another one.

Suppose it introduced a new retrieval of p at time t. That is, it replaced q_t by p in the cache at time t and (after modification) C' doesn't cache p at time $t-1$. The latter property (by inspection of the repair algorithm) implies that C' caches no pages in P at time $t-1$ (before or after modification). It follows that the repair removed one retrieval of q_t at time t.

Now suppose the repair introduced a new retrieval of q_{t-1} at time t. That is, it removed q_{t-1} from the cache at time $t-1$ (replacing it by p) and (after modification) C' caches q_{t-1} at time t. The latter property implies that time t did not need repair (because if it did the repair would have taken $q_t = q_{t-1}$). So C' cached p at time t (before and after modification). So the repair removed a retrieval of p at time t.

So, for each retrieval introduced, another was removed, and the cost at time t didn't increase. □

Proof of Claim 4.4. Fix any non-leaf set P. First consider the repair of any P-phase $[i, j]$ with $i > 1$. Let $c = c_i(P) \neq P$ and $p = p_i(P) = p_i(c)$ be the preferred child and page throughout the phase. When the repair starts, the c-phase containing $[i, j]$ has already been repaired. By inspection, that repair established the following property of C' throughout $[i, j]$: *whenever C' has at least one page in c cached, C' has c's preferred page p cached.* None of the ancestors of c (including P) have had their phases repaired since then, so this property still holds just before the repair of P.

At time i (using here the definition of P-phase and that $i > 1$), the instance π requests a descendant of c, so C caches at least one page p' in c. Suppose that C doesn't evict p' during $[i + 1, j]$. Then, *at every time during $[i, j]$, C caches at least one page in c.* Each repair for c or a descendant of c preserves this property (because such a repair only replaces cached pages in c by other pages in c). Throughout $[i, j]$, then, C' also caches at least one page in c and, by the previous paragraph, must cache P's preferred page p. (Recall that P and c have the same preferred page throughout $[i, j]$.) In this case, by inspection of the algorithm the repair of this P-phase does not change C', and the phase is not costly. We conclude that, for the phase to be costly, C must evict p' during $[i + 1, j]$.

Note that $p' \in c \subset P$. Also, by Claim 4.3, this is not the final P-phase (that is, $j < T$). It follows that the number of costly P-phases with $i > 1$ is at most the number of evictions of pages in P by C, before the final P-phase (with $j = T$).

Regarding the P-phase $[i, j]$ with $i = 1$, if it is costly, then $j < T$ and, by the reasoning in the paragraph before last, in the final P-phase, there is a page p' in P that C either evicts or leaves in the cache at time T.

By the above reasoning, the number of costly P-phases is at most the number of evictions by C of pages in P, plus the number of pages left cached by C at time T. This sum is the number of retrievals of pages in P by C, proving Claim 4.4. □

□

5 Slot-Laminar Paging

This section gives upper bounds for Slot-Laminar Paging. Theorem 5.1 bounds the optimal ratios by $3h^2k$ (deterministic), $3h^2H_k$ (randomized) and $3h^2$ (offline polynomial-time approximation). Theorem 5.2 tightens the deterministic upper bound to $2hk$.

5.1 Upper bounds for randomized and offline Slot-Laminar Paging

Theorem 5.1. *Slot-Laminar Paging admits the following polynomial-time algorithms: a deterministic $3h^2k$-competitive online algorithm, a randomized $3h^2H_k$-competitive online algorithm, and an offline $3h^2$-approximation algorithm.*

Recall that h is the height of the laminar family. The factor of $3h$ in the theorem has not been optimized. For example, when $h = 2$, the $3h = 6$ can be improved to 3. Likewise, one can show that for $k = h = 2$ there is a 3-competitive algorithm for Slot-Laminar Paging. (This matches the lower bound from Theorem 6.1 and hints that $2k - 1$ might be achievable.)

Note that Section 5.2 gives a better deterministic algorithm.

The proof is by a reduction of Slot-Laminar Paging to Page-Laminar Paging, in Lemma 5.1 below. The reduction uses a natural relaxation of Slot-Laminar Paging that relaxes the constraint that each slot hold at most one page (but still enforces the cache-capacity constraint). This relaxation naturally gives an instance of Page-Laminar Paging. The reduction simulates the given Page-Laminar Paging algorithm on multiple instances of Page-Laminar Paging— one for each requested set $S \in \mathcal{S}$, obtained by relaxing the subsequence that contains just those requests contained in S — then aggregates the resulting Page-Laminar Paging solutions to obtain the Slot-Laminar Paging solution. Lemma 5.1 and Theorem 4.1 (for Page-Laminar Paging) immediately imply Theorem 5.1.

Lemma 5.1. *Every $f_h(k)$-approximation algorithm $\mathbb{A}$ for Page-Laminar Paging can be converted into a $3hf_h(k)$-approximation algorithm $\mathbb{B}$ for Slot-Laminar Paging, preserving the following properties: being polynomial-time, online, and/or deterministic.*

Proof. We first define the Page-Laminar Paging *relaxation* of a given Slot-Laminar Paging instance. The idea is to relax the constraint that each slot can hold at most one page, while keeping the cache-capacity constraint. The relaxed problem is equivalent to a Page-Laminar Paging instance over "virtual" pages corresponding to page/slot pairs, where a given virtual page $v(p,s)$ represents page p and slot s and can be placed in any slot.

Definition 5.2. *Fix any k-slot Slot-Heterogenous Paging instance $\sigma = (\sigma_1, \ldots, \sigma_T)$ with requestable slot-set family $\mathcal{S}$. For any page p and $S \in \mathcal{S}$, define $V(p,S) = \{v(p,s) : s \in S\}$, where $v(p,s)$ is a* virtual page *for the pair (p,s) (representing page p being in slot s). Define the* relaxation *of σ to be the k-slot Page-Subset Paging instance $\pi = (P_1, \ldots, P_T)$ defined by $P_t = V(p_t, S_t)$ (where $\sigma_t = \langle p_t, S_t \rangle$, for $t \in [T]$). The requestable set family for π is $\mathcal{P} = \big\{V(p,S) : p$ is any page and $S \in \mathcal{S}\big\}$.*

Instance π is a relaxation of σ in the sense that for any solution C for σ there is a solution D for π with $\mathsf{cost}(D) \le \mathsf{cost}(C)$. (Namely, have D keep in its cache the virtual pages $v(p,s)$ such that C has page p cached in slot s.) It follows that $\mathsf{opt}(\pi) \le \mathsf{opt}(\sigma)$. Crucially, if σ is slot-laminar with height h, then π is page-laminar with the same height h.

Next we define the algorithm $\mathbb{B}$. Fix the $f_h(k)$-approximation algorithm $\mathbb{A}$ for Page-Laminar Paging. Fix the input σ with $\sigma_t = \langle p_t, S_t \rangle$ (for $t \in [T]$) to Slot-Laminar Paging algorithm $\mathbb{B}$. We assume for ease of presentation that Algorithm $\mathbb{A}$ is an online algorithm, and present Algorithm $\mathbb{B}$ as an online algorithm. If $\mathbb{A}$ is not online, $\mathbb{B}$ can easily be executed as an offline algorithm instead.

Assume that the family $\mathcal{S}$ has just one root R with $|R| \le k$. (This is without loss of generality, as multiple roots, being disjoint, naturally decouple any Slot-Laminar Paging instance into independent problems, one for each root.)

For each $S \in \mathcal{S}$, define S's Slot-Laminar Paging *subinstance* σ_S to be obtained from σ by deleting all requests that are not subsets of S. Let π_S denote the (Page-Laminar Paging) relaxation of σ_S. Algorithm $\mathbb{B}$ on input σ executes, simultaneously, $\mathbb{A}(\pi_S)$ for every requestable set $S \in \mathcal{S}$, giving each execution $\mathbb{A}(\pi_S)$ its own independent cache of size $|S|$, composed of copies of the slots in S.

For each such S, Algorithm $\mathbb{B}$ will build its own solution, denoted $\mathbb{B}(\sigma_S)$, for σ_S, also using its own independent cache of size $|S|$ composed of copies of the slots in S. The desired solution to σ will then be $\mathbb{B}(\sigma_R)$ (note $\sigma = \sigma_R$).

For internal bookkeeping purposes only, in presenting Algorithm $\mathbb{B}$, we consider each virtual page $v(p,s)$ (as defined for Page-Laminar Paging) to be a *copy* of page p (functionally equivalently to p), and we have $\mathbb{B}$ maintain cache configurations that place these virtual pages in specific slots, with the understanding that the actual cache configurations are obtained by replacing each virtual page $v(p,s)$ (in whatever slot its in) by a copy of page p. When we analyze the cost, we will consider two copies $v(p,s)$ and $v(p',s')$ to be distinct unless $(p',s') = (p,s)$. In particular, if $\mathbb{B}$ evicts $v(p,s)$ while retrieving $v(p,s')$ (with $s' \ne s$) in the same slot, this contributes 1 to the cost of $\mathbb{B}$. We will upper bound $\mathbb{B}$'s cost overestimated in this way.

Correctness. Algorithm $\mathbb{B}$ will somehow maintain the following invariant over time:

For each requestable set S, for each virtual page $v(p,s)$ currently cached by $\mathbb{A}(\pi_S)$:

1. *the solution $\mathbb{B}(\sigma_S)$ caches $v(p,s)$ in some slot in S, and*
2. *if S has a child c with $s \in c$, and $\mathbb{B}(\sigma_c)$ has $v(p,s)$ in a slot s', then $\mathbb{B}(\sigma_S)$ has $v(p,s)$ in s'.*

The invariant suffices to guarantee correctness of the solution $\mathbb{B}(\sigma_S)$ for each instance σ_S. Indeed, when $\mathbb{B}(\sigma_S)$ receives a request $\langle p_t, S_t \rangle$, its relaxation $\mathbb{A}(\pi_S)$ has just received the request $\{v(p_t,s) : s \in S_t\}$, so $\mathbb{A}(\pi_S)$ is caching a virtual page $v(p_t,s)$ (for some $s \in S_t$) in S. By Condition 1, then, $\mathbb{B}(\sigma_S)$ also has $v(p_t,s)$ in some slot in S. In the case $S = S_t$, this suffices for $\mathbb{B}(\sigma_S)$ to satisfy the request. In the remaining case S has a child c with $S_t \subseteq c$, and $\mathbb{B}(\sigma_C)$ just received the same request, so (assuming inductively that $\mathbb{B}(\sigma_c)$ is correct for σ_c) $\mathbb{B}(\sigma_c)$ has $v(p_t,s)$ in some slot s' in S_t, so by Condition 2 of the invariant $\mathbb{B}(\sigma_S)$ has $v(p_t,s)$ in the same slot s' in S_t, as required. In particular, $\mathbb{B}(\sigma_R)$ will be correct for σ_R, as required.

To maintain the invariant $\mathbb{B}$ does the following for each requestable set S. Whenever the relaxed solution $\mathbb{A}(\pi_S)$ evicts a page $v(p,s)$, the solution $\mathbb{B}(\sigma_S)$ also evicts $v(p,s)$. Whenever $\mathbb{A}(\pi_S)$ retrieves a page $v(p,s)$, the solution $\mathbb{B}(\sigma_S)$ also retrieves $v(p,s)$, into any vacant slot in S (there must be one, because $\mathbb{A}(\sigma_S)$ caches at most $|S|$ pages). This retrieval can cause up to two violations of Condition 2 of the invariant: one at $\mathbb{B}(\sigma_S)$, because $v(p,s)$ is already cached by a child $\mathbb{B}(\sigma_c)$ but in some slot $s_1 \ne s'$; the other at the parent $\mathbb{B}(\sigma_P)$ of $\mathbb{B}(\sigma_S)$ (if any), because $v(p,s)$ is already cached by the parent, but in some slot $s_2 \ne s'$. In the case that the retrieval does create two violations (and $s_1 \ne s_2$), $\mathbb{B}$ restores the invariant by "rotating" the contents of the slots s_1, s', and s_2 in $\mathbb{B}(\sigma_S)$ *and in each ancestor*, as shown in Figure 4 (note that pages in gray are not moved). This restores the invariant at the expense of three retrievals at the root. (Note that y_{i+3} and z_{i+2} cannot be $v(p,s)$, so moving $v(p,s)$ out of slots s' and s_2 doesn't introduce a violation there.) Each remaining case can be handled similarly, also with at most three retrievals (in fact at most two) at the root.

before

	slot s_1	slot s'	slot s_2
root R :	x_1	y_1	z_1
	$\vdots$	$\vdots$	$\vdots$
	x_i	y_i	z_i
parent $\mathbb{B}(\sigma_P)$:	x_{i+1}	y_{i+1}	$v(p,s)$
$\mathbb{B}(\sigma_S)$:	x_{i+2}	$v(p,s)$	z_{i+2}
child $\mathbb{B}(\sigma_c)$:	$v(p,s)$	y_{i+3}	z_{i+3}

$\Longrightarrow$

after

slot s_1	slot s'	slot s_2
z_1	x_1	y_1
$\vdots$	$\vdots$	$\vdots$
z_i	x_i	y_i
$v(p,s)$	x_{i+1}	y_{i+1}
$v(p,s)$	x_{i+2}	z_{i+2}
$v(p,s)$	y_{i+3}	z_{i+3}

Figure 4: "Rotating" slots in $\mathbb{B}(\sigma_S)$ and ancestors to preserve the invariant.

input: Slot-Laminar Paging instance $(k, \mathcal{S}, \sigma = (\sigma_1, \ldots, \sigma_T))$

1. let C_0 be the empty cache configuration
2. for $t \leftarrow 1, 2, \ldots, T$, respond to the current request $\sigma_t = \langle p, S\rangle$ as follows:

3.1. if C_{t-1} satisfies the request: let $C_t = C_{t-1}$; let $R_t = R_{t-1} \cup \{\sigma_t\}$

3.2. else if the set of requests in the current phase is not coverable:

3.3.1. start new phase; use any slot in S to cover σ_t (set all other slots to free) to obtain C_t; let $R_t = \{\sigma_t\}$

3.4. else:

3.5.1. determine minimal sequences $\langle s_1, \ldots, s_m\rangle$, $\langle S_0 = S, S_1, \ldots, S_{m-1}\rangle$, and $\langle p_0 = p, p_1, \ldots, p_{m-1}\rangle$ with $m \le k + 1 - |S|$ such that
(i) for $1 \le i < m$, $S_{i-1} \subset S_i$ and $s_i \in S_{i-1}$
(ii) s_i currently covers $\langle p_i, S_i\rangle \in \mathsf{rep}(R_{t-1})$ for $1 \le i < m$ and
(iii) $s_m \in S_{m-1}$ is either free or currently covers $\langle p, S'\rangle \in \mathsf{rep}(R_{t-1})$ where $S \subset S'$

3.5.2. bring p_{i-1} to slot s_i to cover $\langle p_{i-1}, S_{i-1}\rangle$ for $1 \le i \le m$ to obtain C_t; let $R_t = R_{t-1} \cup \{\sigma_t\}$

Figure 5: Deterministic online Slot-Laminar Paging algorithm RefSearch

Total cost. Each retrieval by some $\mathbb{A}(\sigma_S)$ causes at most three retrievals in $\mathbb{B}(\sigma_R)$, so

$$\mathsf{cost}(\mathbb{B}(\sigma_R)) \;\le\; \sum_{S\in\mathcal{S}} 3\,\mathsf{cost}(\mathbb{A}(\pi_S)) \;\le\; \sum_{S\in\mathcal{S}} 3\,f_h(|S|)\mathsf{opt}(\pi_S) \;\le\; 3f_h(k)\sum_{S\in\mathcal{S}}\mathsf{opt}(\sigma_S) \;\le\; 3f_h(k)\,h\,\mathsf{opt}(\sigma_R).$$

The second step uses that $\mathbb{A}(\pi_S)$ is $f_h(|S|)$-competitive for π_S. The third step uses that π_S is a relaxation of σ_S so $\mathsf{opt}(\pi_S) \le \mathsf{opt}(\sigma_S)$, and that $|S| \le k$ so $f_h(|S|) \le f_h(k)$.[2] The last step uses that the sets within any given level $i \in \{1, 2, \ldots, h\}$ of the laminar family are disjoint, so $\mathsf{opt}(\sigma_R)$ is at least the sum, over the sets S within level i, of $\mathsf{opt}(\sigma_S)$. This shows that $\mathbb{B}$ is an $f_h(k)$-approximation algorithm. To finish, we observe that $\mathbb{B}$ is polynomial-time, online, and/or deterministic if $\mathbb{A}$ is. □

5.2 Improved upper bound for deterministic Slot-Laminar Paging

For Slot-Laminar Paging, this section presents a deterministic algorithm with competitive ratio $O(hk)$, improving upon the bound of $O(h^2k)$ from Theorem 5.1. The algorithm, RefSearch, refines ExhSearch. Like ExhSearch, it is phase-based and maintains a configuration that can cover all requests in a phase; however, in order to satisfy the next request in the current phase, the particular configuration is chosen by judiciously moving pages in certain slots that are serving requests along a path in the laminar hierarchy.

Theorem 5.2. *For Slot-Laminar Paging, Algorithm* RefSearch *(Fig. 5) has competitive ratio at most* $2\cdot\mathsf{mass}(\mathcal{S}) \le 2hk$.

Before we prove the theorem we define the terminology used in the algorithm and establish some useful properties. Say that a set R of requests is *coverable* if there exists a configuration that covers every request in R. To determine if a set R of requests is covered by a configuration, it is sufficient (and necessary) to examine the maximal subset of "deepest" requests in the laminar hierarchy, as observed in Claim 5.3 below. Formally, a request $\langle p, S\rangle$ is an *ancestor* (resp., *descendant*) of $\langle p, S'\rangle$ if $S \supseteq S'$ (resp., $S \subseteq S'$). For any set R of requests, define $\mathsf{rep}(R)$ as the set of requests in R that do not have any proper descendants in R. That is, $\mathsf{rep}(R) = \{\langle p, S\rangle \in R \mid \forall S' \subset S, \langle p, S'\rangle \notin R\}$. For $r = \langle p, S\rangle$, define $\mathsf{anc}(r, R) = \{\langle p, S'\rangle \in R | S \subseteq S'\}$.

[2] We assume here that $f_h(|S|) \le f_h(k)$, which is without loss of generality as one can simulate a cache of size $|S| \le k$ using a cache of size k by introducing artificial requests that force $k - |S|$ slots to be continuously occupied.

Claim 5.3. *Let R be a set of requests. R is coverable iff $\mathsf{rep}(R)$ is coverable. No two requests in $\mathsf{rep}(R)$ can be covered by the same slot.*

Proof. Since $\mathsf{rep}(R) \subseteq R$, if R is coverable, so is $\mathsf{rep}(R)$. Also, if C covers $\mathsf{rep}(R)$, then C covers R since every r in R is an ancestor of some r' in $\mathsf{rep}(R)$, and can be covered by the same slot covering r'. The second part of the claim holds since any two requests in $\mathsf{rep}(R)$ request either different pages or disjoint slot sets. $\square$

Algorithm REFSEARCH is defined in Figure 5. Following Claim 5.3, REFSEARCH maintains the current configuration by using a distinct slot to cover each request in $\mathsf{rep}(R)$, where R is the current set of requests in the phase, and designates any remaining slots as *free*. The following lemma establishes the validity of step 3.5.1.

Lemma 5.4. *Let R be a set of requests covered by configuration C, and $r = \langle p_0, S_0 \rangle$ be such that r is not covered by C and $R \cup \{r\}$ is coverable. Then, there exist sequences $\langle s_1, \ldots, s_m \rangle$, $\langle S_0, S_1, \ldots, S_{m-1} \rangle$, and $\langle p_0, p_1, \ldots, p_{m-1} \rangle$ where $m \leq k + 1 - |S|$ such that (i) for $1 \leq i < m$, $S_{i-1} \subset S_i$ and $s_i \in S_{i-1}$ (ii) s_i is currently covering request $\langle p_i, S_i \rangle \in \mathsf{rep}(R)$ for $1 \leq i < m$ and (iii) $s_m \in S_{m-1}$ is either a free slot or is currently covering $\langle p_0, S_m \rangle \in \mathsf{rep}(R)$. Furthermore, transforming C by moving page p_{i-1} to slot s_i, for $1 \leq i \leq m$, yields a configuration that covers $R \cup \{r\}$.*

Proof. The proof is by induction on the laminar hierarchy in top-down order. We begin with the following helper claim that analyzes when a particular request is not covered by a given configuration.

Lemma 5.5. *Let R be a set of requests covered by configuration C, and $r = \langle p, S \rangle$ be a request not covered by C. If $R \cup \{r\}$ is coverable, then there exists a slot s in S that is either free or covering $\langle p, S_s \rangle \in \mathsf{rep}(R)$ where $S \subset S_s$.*

Proof. Suppose, for the sake of contradiction, that C covers R but does not cover $\langle p, S \rangle$, and $R \cup \{r\}$ is coverable, yet every slot s is not free, and is covering $\langle p_s, S_s \rangle$ in $\mathsf{rep}(R)$ with $p_s \neq p$ or $S \subsetneq S_s$. Since no two requests in $\mathsf{rep}(R)$ can be covered by the same slot, $|\mathsf{rep}(R)| = k$. Since C does not cover $\langle p, S \rangle$, for every s either s is not at p or $S \subsetneq S_s$. Thus, for every s at p, S_s is neither an ancestor or descendant of S. Therefore, $|\mathsf{rep}(R \cup \{r\})| = |\mathsf{rep}(R) \cup \{r\}| = k + 1$, implying that $\mathsf{rep}(R) \cup \{r\}$ is not coverable. $\square$

We return to the proof of the lemma. For the induction base, consider $S_0 = [k]$. Since r is not covered by C, $R \cup \{r\}$ is coverable, and every slot set is subset of $[k]$, so by Lemma 5.5 there is a free slot $s_1 \in S_0$. The desired claim of the lemma holds with $m = 1$ and sequences $\langle s_1 \rangle$, $\langle S_0 \rangle$ and $\langle p_0 \rangle$ which satisfy (i), (ii), and (iii). Since s_1 is free, bringing page p_0 to slot s_1 yields a new configuration that covers $R \cup \{r\}$.

We now establish the induction step. Let R, C, and $r = \langle p_0, S_0 \rangle$ be as given. By Lemma 5.5 there are two cases. In the first case, there is a free slot $s_1 \in S_0$. Then the desired claim holds with $m = 1$, and sequences $\langle s_1 \rangle$, $\langle S_0 \rangle$ and $\langle p_0 \rangle$.

The remainder of this proof concerns the second case, in which there is a slot $s_1 \in S_0$ currently covering a request $r' = (p_1, S_1)$ in $\mathsf{rep}(R)$ with $S_0 \subset S_1$. Let C' denote the configuration, which is identical to C except that $C(s_1) = p_0$. Let $R' = R \cup \{r\} \setminus \mathsf{anc}(r', R)$. Then, C' covers R', but does not cover r'. However, since $R' \cup \{r'\}$ is a subset of $R \cup \{r\}$, which is coverable, $R' \cup \{r'\}$ is also coverable. Since $S_1 \supset S_0$, by the induction hypothesis, there are sequences $\langle s_2, \ldots, s_m \rangle$, $\langle S_1, S_2, \ldots S_{m-1} \rangle$ and $\langle p_1, p_2, \ldots, p_{m-1} \rangle$ where $m - 1 \leq k + 1 - |S_1|$ such that (i) $s_i \in S_{i-1}$ and $S_{i-1} \subset S_i$ for all $2 \leq i < m$; (ii) s_i is currently covering $(p_i, S_i) \in \mathsf{rep}(R')$; and (iii) either s_m is a free slot in C' or is currently covering a request $(p_1, S_m) \in \mathsf{rep}(R')$ such that $S_1 \subset S_m$. Furthermore, transforming C' to C'' by moving page p_{i-1} to s_i for $2 \leq i \leq m$, covers $R' \cup \{r'\}$. Note that s_m has to be a free slot in C' since the latter case of (iii) above cannot hold: any request (p_1, S_m) is in $\mathsf{anc}(r', R)$, all requests of which are excluded from R'.

We now establish the desired claim for C, R, and r. Consider sequences $\langle s_1, \ldots, s_m \rangle$, $\langle S_0, S_1, \ldots S_{m-1} \rangle$ and $\langle p_0, \ldots, p_{m-1} \rangle$. The desired conditions (i) and (ii) follow from (i) and (ii) of the induction hypothesis and the fact that in C, $s_1 \in S_0$ is currently covering a request (p_1, S_1) in $\mathsf{rep}(R)$ with $S_0 \subset S_1$. For (iii), note that since s_m is a free slot in C', either s_m is a free slot in C or (p_0, S_m) is in $\mathsf{rep}(R)$, where $S_0 \subset S_m$, thus establishing (iii). Finally, transforming C to C'' by moving p_{i-1} to s_i for $1 \leq i \leq m$, covers $R' \cup \{r'\}$; since $\mathsf{rep}(R \cup \{r\}) = \mathsf{rep}(R' \cup \{r'\})$, C'' also covers $R \cup \{r\}$. $\square$

Proof of Theorem 5.2. We first show that the number of page retrievals during a phase of the algorithm REFSEARCH is at most $2 \cdot \mathsf{mass}(\mathcal{S})$. Let R denote the set of requests in the current phase. Let C be the current configuration. Recall that the algorithm designates a distinct slot to cover each request in $\mathsf{rep}(R)$, and sets any remaining slot as free. We maintain a potential Φ, which equals the sum, over every slot s that is currently covering a request (p, S), of one plus the distance of S from the root of the laminar hierarchy.

We will show that Φ never decreases in a phase, and that for every step containing a page retrieval, Φ increases and the increase is at most one less than the number of page retrievals in the step. This implies that the number of page retrievals in a step is at most twice the increase in potential.

Consider new request $r = \langle p, S \rangle$. If the current configuration satisfies r, no page is retrieved, and Φ remains the same. If there is no configuration that can satisfy all requests in this phase, then the phase ends and Φ equals 0.

It remains to consider the case where $R \cup \{r\}$ is coverable, yet the current configuration covering R does not cover r. The algorithm computes minimal sequences $\langle s_i \rangle$, $\langle S_i \rangle$, and $\langle p_i \rangle$ satisfying the three conditions described in the algorithm;

these sequences exist by Lemma 5.4. Then, the algorithm replaces the page in each slot s_i, for $1 \leq i < m$: instead of covering $\langle p_i, S_i \rangle$, s_i now covers $\langle p_{i-1}, S_{i-1} \rangle$, each replacement increasing Φ by at least one.

Finally, we analyze slot s_m. There are two cases. The first case is where there does not exist $\langle p, S' \rangle \in \mathsf{rep}(R)$ such that $S \subset S'$. If s_m is not free, Lemma 5.4 states that s_m is covering a request of form $\langle p, S' \rangle \in \mathsf{rep}(R)$ where $S \subset S'$, which contradicts the assumption of this case. So, slot s_m must be free, and p_{m-1} is retrieved into s_m to cover $\langle p_{m-1}, S_{m-1} \rangle$. This increases Φ by at least one, establishing the desired claim.

In the second case, there exists $\langle p, S' \rangle \in \mathsf{rep}(R)$ such that $S \subset S'$. Note that in C there is a slot $s' \in S'$ that covers $\langle p, S' \rangle$ but does not cover r. As the algorithm uses s_1 to cover r, slot s' becomes free. Since $S \subset S'$ and $S \subset S_{m-1}$, either $S_{m-1} \subset S'$ or $S' \subset S_{m-1}$. There are two sub-cases. If $S_{m-1} \subset S'$, by condition (iii) of Lemma 5.4, s_m is either free in C or s_m is covering $\langle p, S' \rangle \in \mathsf{rep}(R)$. If s_m is free, then the algorithm uses s_m to cover $\langle p_{m-1}, S_{m-1} \rangle$, and since s' was set free from covering $\langle p, S' \rangle$, Φ increases by at least one, establishing the desired claim. Otherwise, $s_m = s'$ retrieves a page so that instead of covering $\langle p, S' \rangle$, it now covers $\langle p_{m-1}, S_{m-1} \rangle$, increasing Φ by at least one unit, again establishing the desired claim.

If $S' \subseteq S_{m-1}$, note that $s_m = s'$ and $S_{m-2} \subset S'$. Otherwise, the sequences $\langle s_1, \ldots, s_{m-2}, s' \rangle$, $\langle S_0, \ldots, S_{m-1} \rangle$, and $\langle p_0, p_1, p_{m-1} \rangle$ satisfy the three conditions of Lemma 5.4, implying that the sequences selected by the algorithm are not minimal. The slot s_m now covers $\langle p_{m-1}, S_{m-1} \rangle$ instead of covering $\langle p', S' \rangle$; this decreases Φ. However, the increase in Φ due to the movement of s_{m-1} from covering $\langle p_{m-1}, S_{m-1} \rangle$ to covering $\langle p_{m-2}, S_{m-2} \rangle$ is greater than this decrease since $S_{m-2} \subset S' \subset S_{m-1}$, implying that the increase in Φ is at most one less than the number of page retrievals.

Since Φ cannot exceed $\mathsf{mass}(\mathcal{S})$, the maximum number of page retrievals in a phase by the algorithm is at most $2 \cdot \mathsf{mass}(\mathcal{S})$. Furthermore, the optimal algorithm has a cost of at least one per phase since a phase ends if no configuration can cover all requests in the phase and the new incoming request. This shows that the algorithm is $2 \cdot \mathsf{mass}(\mathcal{S})$-competitive. □

6 All-or-One Paging

Recall that All-or-One Paging is the extension of standard Paging that allows two types of requests: A general request for a page p, denoted $\langle p, * \rangle$, can be served by having p in any cache slot. A specific request $\langle p, j \rangle$, where $j \in [k]$, must be served by having p in slot j of the cache. (Section 2 gives a formal definition.) It is a restriction of Slot-Laminar Paging with $h = 2$.

For All-or-One Paging, this section first shows that the optimal ratios are at least $2k-1$ (deterministic) and $2H_k - O(1)$ (randomized). It then shows that the optimal deterministic ratio is at most $3k - 2$, and finishes by showing that the offline problem is $\mathbb{NP}$-hard.

6.1 Improved lower bound for All-or-One Paging

Theorem 6.1. *Every online deterministic (resp., randomized) algorithm $\mathbb{A}$ for the All-or-One Paging problem has competitive ratio at least $2k - 1$ (resp., $2H_k - O(1)$).*

Proof. We construct an instance with $k + 1$ pages $p_1, p_2, \ldots, p_{k+1}$. The adversary strategy consists of L phases, for some sufficiently large integer L. When each phase starts, both the algorithm and the adversary have exactly the same cache configuration. To describe one phase, assume without loss of generality that when the phase starts each page p_i, for $i = 1, 2, \ldots, k$, is in cache slot i. The adversary request sequence in this phase starts with the following $2k - 1$ requests:

$$\langle p_{k+1}, * \rangle \,,\ \langle p_{i_1}, i_1 \rangle \,,\ \langle p_{k+1}, * \rangle \,,\ \langle p_{i_2}, i_2 \rangle \,,\ \ldots \langle p_{k+1}, * \rangle \,,\ \langle p_{i_{k-1}}, i_{k-1} \rangle \,,\ \langle p_{k+1}, * \rangle \,,$$

where i_j is the index of the cache slot which $\mathbb{A}$ used to serve the jth general request. Note that in this sequence after each specific request the cache content of $\mathbb{A}$ is the same as in the beginning. (Without loss of generality we can assume that at each fault $\mathbb{A}$ changes only one cache slot.)

The adversary can serve the above sequence by fetching p_{k+1}, at the beginning of the phase, into any slot other than $i_1, i_2, .., i_{k-1}$. To finish the phase, the adversary can force $\mathbb{A}$ to reach the same configuration as the one of the adversary, by repeatedly making specific requests, free for her, to cache locations where $\mathbb{A}$ has a different page. Then the adversary's total cost for the phase is 1 and the total cost of $\mathbb{A}$ is at least $2k - 1$.

The above lower bound can be combined with the strategy for proving an H_k lower bound for Paging to give a lower bound of $2H_k - O(1)$ on the competitive ratio of randomized algorithms for All-or-One Paging. □

6.2 Improved upper bound for All-or-One Paging

This section shows that the optimal deterministic ratio for All-or-One Paging is at most $3k-2$. This strengthens the bound of $4k$ implied by Theorem 5.2.

Theorem 6.2. *The competitive ratio of Algorithm* FLUSHANDMARK *(Figure 6) is at most $3k - 2$.*

input: All-or-One Paging instance $(k, \sigma = (\sigma_1, \ldots, \sigma_T))$
At each step, each cache slot is either empty or *full*, and some full slots may be *marked*.
The algorithm partitions the input sequence into *phases*. When a phase starts, all slots are empty.
1. for each $t \leftarrow 1, 2, \ldots, T$, respond to the current request $\sigma_t = \langle p, s \rangle$ as follows:
2.1. flush the cache and start a new phase if any of the following conditions holds:
(pe1) σ_t is a general request, the cache is full, and p is not in the cache
(pe2) σ_t is a specific request, the cache is full, and each slot containing p is marked and differs from s
(pe3) σ_t is a specific request, p is not in slot s, and s is marked
3. if σ_t is a general request and p is not in the cache:
4.1. place p in any empty slot (which exists by Condition (pe1))
5. else if σ_t is a specific request and s is not marked: *— if s is marked it contains p (by Condition (pe3))*
6.1. if there some other unmarked slot contains p: remove p from that slot
6.2. place p in s and mark s (if s was full, evict the page in s first)

Figure 6: Deterministic online All-or-One Paging algorithm FLUSHANDMARK.

Proof. A *filling event* is a step when a page is put in an empty slot. A *marking event* is a step when a non-marked slot gets marked. After a marking event of slot s, s will remain marked throughout the phase (and its content will not change). Thus there are at most k marking events. Marking slot s can also empty some other full slot if it has the requested page. The number of such *emptying events* is at most k. Each filling event of a slot s is either the first filling of s or the last event involving slot s was an emptying event. So the total number of filling events is at most $2k$. Each page retrieval corresponds to either a filling or a marking event; therefore the cost per phase is at most $3k$.

This cost bound can be improved to $3k - 2$ as follows. If the number of marking events is at most $k - 1$, then the number of emptying events is also at most $k - 1$, so the number of filling events is at most $2k - 1$, giving us the cost of at most $3k - 2$. So assume that there are exactly k marking events, and consider the last one, say a specific request $\langle p, s \rangle$. By the definition of marking events, s is unmarked. There are two cases:

- If s is empty when $\langle p, s \rangle$ is issued then s is the only empty slot, so instead of the request $\langle p, s \rangle$, view this as a $\langle p, * \rangle$-request and bring p to s. Then the bound above for the case with at most $k - 1$ marking events applies.
- If s is full but not marked, then it must be hold p, because otherwise the phase would end. Thus the request on p does not cost anything, and it does not empty any slots, so there are at most $2k - 1$ filling events, so the total cost is at most $3k - 2$.

We next lower bound the adversary cost. We show that the adversary cost is at least one per every *shifted phase*, where shifted phase i starts with the second request of phase i and ends with the first request of phase $i + 1$ (both inclusive). We assume that the phases are numbered $1, 2, \ldots$.

Fix phase j. Consider the request $\langle p, s \rangle$ that ends phase j; that is, $\langle p, s \rangle$ is the first request in phase $j + 1$. If condition (pe3) holds, then slot s is marked and does not have p; therefore among the requests in phase j and the request $\langle p, s \rangle$, we have two different specific requests for slot s, so the adversary retrieves at least one page to slot s in shifted phase j, and pays at least 1.

In the rest of the argument, assume that (pe3) is not true. Suppose that either (pe1) or (pe2) holds. Then the algorithm's cache is full when $\langle p, s \rangle$ is issued, and its slots are of two types: (i) marked slots, each resulting from a specific request to this slot, and (ii) slots that contain pages that were fetched into the cache on a general request and not requested specifically later. All these requests occurred during phase j. After the first request phase j, the adversary must have the requested page in the cache, and if she did not fault in the remaining steps of this phase then the adversary must have exactly the same pages in the cache as the algorithm, and the marked pages must be in the same slots. But then, in Case (pe1), the adversary would not have p in its cache, and in Case (pe2), she would not have p in slot s. Thus the adversary must pays at least 1 in shifted phase j.

For any sequence of requests, the cost of Algorithm FLUSHANDMARK is at most $3k - 2$ times the number of phases, and the cost of the adversary is at least the number of full shifted phases, which is at most one less than the number of phases. The cache is initially empty, so in the first shifted phase the adversary cost is actually k. (With a different assumption about the initial configuration, we would have to allow an additive $O(k)$ term in the competitive bound.) This yields the desired $3k - 2$ upper bound on the competitive ratio of FLUSHANDMARK. □

The analysis above is tight. To see this, consider an instance with k pages $p_1, p_2, \ldots, p_k$. The adversary has each p_i in slot i. Algorithm FLUSHANDMARK starts with empty cache. After making general requests $\langle p_i, * \rangle$, for $i = 1, 2, \ldots, k$, breaking the tie, assume that the algorithm loads each p_i into slot $i + 1$, for $i = 1, 2, \ldots, k - 1$, and p_k into slot 1. So far the algorithm's cost is k. Now the adversary makes a specific request $\langle p_1, 1 \rangle$, on which the algorithm moves p_1 from slot 2 to slot 1, evicting p_k, and marks slot 1. Next the adversary requests $\langle p_2, * \rangle$, on which the algorithm fetches p_k into slot

2. This is followed by request $\langle p_2, 2\rangle$, on which the algorithm moves p_2 from slot 3 to slot 2, evicting p_k, and marking slot 2. This process is then repeated for slots $3, 4, \dots, k-1$, adding $2k-2$ to the algorithm's cost, for the total of $3k-2$. This example can be modified to show that even if Algorithm FLUSHANDMARK does not actually empty cache slots, but instead only designates them as "virtually" empty, there is a refined adversary strategy that does not require tie breaking and still forces the ratio to be at least $3k-2$.

Theorems 6.1 and 6.2 still leave a gap between the lower and upper bounds. We leave closing this gap as an open problem. One can show that for $k = 2$ there is a 3-competitive algorithm, matching the lower bound from Theorem 6.1, and hinting that $2k-1$ might be achievable. The main difficulty in improving the upper bound of $3k-2$ is in maintaining a good estimate for the optimum cost. This is not unexpected though, given the $\mathbb{NP}$-hardness result in Section 6.3, next.

6.3 NP-completeness of offline All-or-One Paging

Theorem 6.3. *The offline version of All-or-One Paging is $\mathbb{NP}$-complete.*

Proof. Let $G = (V, E)$ be a graph with vertex set $V = \{0, 1, \dots, n-1\}$. Given an integer k, $1 \le k \le n$, we compute in polynomial time a request sequence σ and an integer F such that the following equivalence holds: G has a vertex cover of size k if and only if there is a schedule for σ whose cost with a cache size $k+2$ is at most F.

At a fundamental level our proof resembles the argument in [25], where $\mathbb{NP}$-completeness of an interval packing problem was proved. The basic idea of our proof is to represent the vertices by a collection of intervals with specified endpoints that are to be packed into a strip of width k. These intervals will be represented by pairs of requests, one at the beginning and one at the end of the interval, and the strip to be packed is the cache. Since the strip's capacity is bounded by k, only a subset of intervals can be packed, and the intervals that are packed correspond to a vertex cover.

There will actually be many "bundles" of such intervals, with each bundle containing n intervals corresponding to the n vertices. If we had $|E|$ bundles and if we forced each bundle's packing (that is, the packed subset) to be the same, we could add an edge-gadget to each bundle that will verify that all edges are covered. While it does not seem possible to design these bundles to force all bundles' packings to be equal, there is a way to design them to ensure that the packing of each bundle is dominated (in the sense to be defined shortly) by the next one, and this dominance relation has polynomial depth. So with polynomially many bundles we can ensure that there will be $|E|$ consecutive equally packed bundles, allowing us to verify whether the vertex set corresponding to this packing is indeed a correct vertex cover.

Set dominance. We consider the family of all k-element subsets of V. For any two k-element sets $X, Y \subseteq V$, we say that Y *dominates* X, and denote it $X \preceq Y$, if there is a 1-to-1 function $\psi : X \to Y$ such that $x \le \psi(x)$ for all $x \in X$. We write $X \prec Y$ iff $X \preceq Y$ and $X \ne Y$. The dominance relation is a partial order. The following lemma from [25] will be useful:

Lemma 6.1. *Let $X_1, X_2, \dots, X_p \subseteq V$ be sets of cardinality k such that $X_1 \prec X_2 \prec \dots \prec X_p$. Then $p \le k(n-k)$.*

Cover chooser. We start by specifying the "cover chooser" sequence σ' of requests. In this sequence some time slots will not have assigned requests. Some of these unassigned slots will be used later to insert requests representing edge gadgets.

Let $m = |E| + 1$. (For notation-related reasons, it is convenient to have m be one larger than the number of edges.) Let also $P = k(n-k) + 1$ and $B = mP$. In σ' we will use the following pages and requests:

- We have nB pages $x_{b,j}$, for $b = 0, 1, \dots, B-1$ and $j = 0, 1, \dots, n-1$. For each page $x_{b,j}$ there are two general requests $\langle x_{b,j}, *\rangle$ in σ' at time steps $\tau_{b,j} = 9(bn+j)$ and $\tau'_{b,j} = 9(bn+j) + 9n - 6$. These requests are called *vertex requests*. They are grouped into *bundles* of requests, where bundle b consists of all $2n$ general requests to pages $x_{b,0}, x_{b,1}, \dots, x_{b,n-1}$. See Figures 7 and 8 for illustration.
- We have B pages y_b, for $b = 0, 1, \dots, B-1$. For each page y_b we have two specific requests $\langle y_b, k+1\rangle$ and $\langle y_b, k+2\rangle$ in σ', at times $\theta_b = \tau'_{b,0} - 2 = \tau_{b,n-1} + 1$ and $\theta'_b = \tau'_{b,0} - 1 = \tau_{b,n-1} + 2$, respectively. For each b, these requests are called *b-blocking requests*, because for each page $x_{b,j}$ in bundle b we have $\theta_b, \theta'_b \in [\tau_{b,n-1}, \tau'_{b,0}]$, so these two requests make it impossible to have both requests $\langle x_{b,j}, *\rangle$ served in cache slots $k+1$ or $k+2$ with only one fault.

Slots $1, 2, \dots, k$ in the cache will be referred to as *vertex* slots. Slot $k+1$ is called the *edge-gadget* slot, and slot $k+2$ is called the *junkyard* slot.

Let $F' = (2n-k+2)B$. For any schedule S of σ' and any bundle b, denote by $V_{S,b}$ the set of vertices $j \in V$ for which σ' has a hit in S on request $\langle x_{b,j}, *\rangle$ at time $\tau'_{b,j}$. In other words, S keeps $x_{b,j}$ in the cache throughout the time interval $[\tau_{b,j}, \tau'_{b,j}]$.

Lemma 6.2. *(a) The minimum number of faults on σ' in a cache of size $k+2$ is F'. (b) If S is a schedule for σ' with at most F' faults, then for any $b = 0, 1, \dots, B-2$ we have $V_{S,b} \preceq V_{S,b+1}$.*

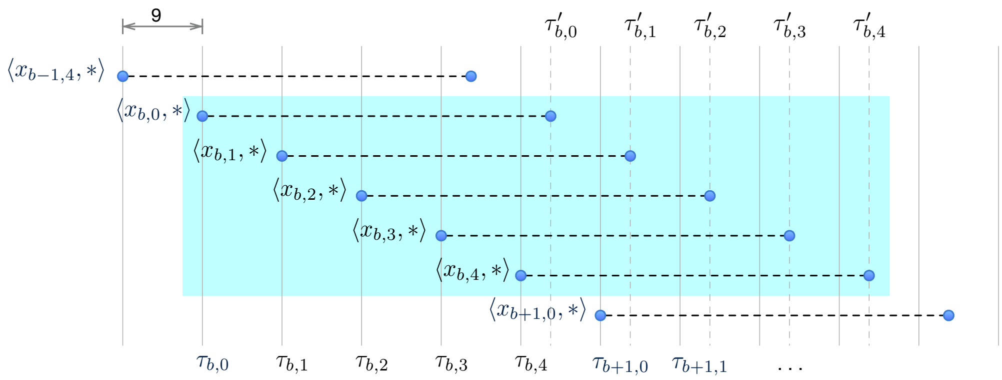

Figure 7: The sequence of vertex requests, for $n = 5$. The shaded region contains requests from bundle b.

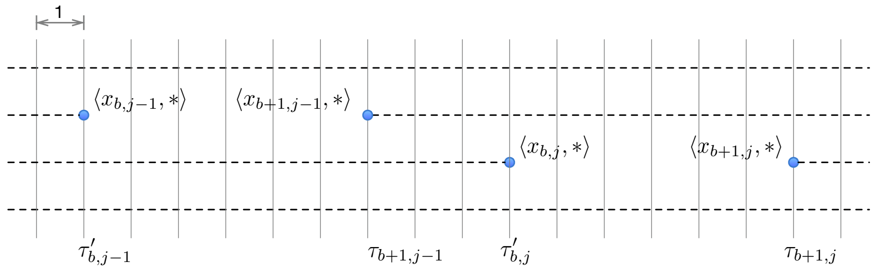

Figure 8: A more detailed picture showing relations between general requests to pages $x_{b,j-1}$, $x_{b,j}$, $x_{b+1,j-1}$, and $x_{b+1,j}$, where $1 \le j \le n-1$.

Proof. (a) There are $2B$ specific b-blocking requests $\langle y_b, k+1\rangle$ and $\langle y_b, k+2\rangle$ and all of these are faults. Consider a bundle b. For this bundle, for each j, the two requests $\langle x_{b,j}, *\rangle$ at times $\tau_{b,j}$ and $\tau'_{b,j}$ are separated by requests $\langle y_b, k+1\rangle$ and $\langle y_b, k+2\rangle$. Thus if S has a hit at time $\tau'_{b,j}$ then page $x_{b,j}$ must have been stored in one of the vertex slots $1, 2, \ldots, k$ throughout the time interval $[\tau_{b,j}, \tau'_{b,j}]$. As there are k vertex slots, S can have at most k hits on the requests in bundle b. So, including the faults at $\langle y_b, k+1\rangle$ and $\langle y_b, k+2\rangle$, the number of faults in S associated with this bundle b will be at least $2 + k + 2(n-k) = 2n - k + 2$. We thus conclude that the total number of faults is at least F'.

It is also possible to achieve only F' faults on σ', as follows: for each b, and for each vertex $j = 0, 1, \ldots, k-1$, at time $\tau_{b,j}$ load $x_{b,j}$ into cache slot $j+1$ and keep it there until time $\tau'_{b,j}$. For $j = k, \ldots, n-1$, load each request to $x_{b,j}$ into slot $k+2$. This will give us exactly F' faults.

(b) If S makes at most F' faults, since there are $2B$ faults on the blocking requests and for each bundle S makes at least $2n-k$ faults on vertex requests, S must make *exactly* $2n-k$ faults on vertex requests from each bundle, including one request for each vertex $j \in V_{S,b}$ and two requests for each vertex $j \notin V_{s,b}$. If $u \in V_{S,b}$ and $x_{b,u}$ is stored by S in slot ℓ of the cache throughout its interval $[\tau_{b,u}, \tau'_{b,u}]$, and if some $x_{b+1,v}$, for $v \in V_{S,b+1}$, is stored by S in slot ℓ throughout its interval $[\tau_{b+1,v}, \tau'_{b+1,v}]$, then we must have $v \ge u$. This is because otherwise we would have $\tau_{b+1,v} < \tau'_{b,u}$, that is the intervals of $x_{b,u}$ and $x_{b+1,v}$ would overlap, so we would fault at least three times on the requests to these pages. This implies part (b). ☐

We partition all bundles into *phases*, where phase $p = 0, 1, \ldots, P-1$ consists of m bundles $b = pm, pm+1, \ldots, pm+m-1$. (Recall that $m = |E| + 1$.) The corollary below states that there is a phase p in which all sets $V_{S,b}$ must be equal. It follows directly from Lemmas 6.1 and 6.2, by applying the pigeonhole principle.

Corollary 6.3. *If S is a schedule for σ' with F' faults, then there is index p, $0 \le p \le P-1$, for which $V_{S,pm} = V_{S,pm+1} = \cdots = V_{S,pm+m-1}$.*

Edge gadget. For each fixed phase p, we create m edge gadgets, one for each edge. Ordering the edges arbitrarily, the gadget for the eth edge, where $0 \le e \le m-2$, will be denoted $\boldsymbol{\omega}_{p,e}$, and it will consist of 8 requests between times θ'_{pm+e} and θ_{pm+e+1}, that is in the region where bundles $pm+b$ and $pm+b+1$ overlap.

Let the eth edge be (u, v), where $u < v$. Edge gadget $\boldsymbol{\omega}_{p,e}$ will consist of the following requests:

- Two specific requests $\langle z_{p,u}, k+2\rangle$ at times $\tau'_{pm+e,u} + 2$ and $\tau'_{pm+e,u} + 4$, and two specific requests $\langle z_{p,v}, k+2\rangle$ at times $\tau'_{pm+e,v} + 2$ and $\tau'_{pm+e,v} + 4$.
- General requests $\langle g_{p,u}, *\rangle$, $\langle g_{p,v}, *\rangle$, at times $\tau'_{pm+e,u} + 3$ and $\tau'_{pm+e,v} + 3$.

- A pair of requests $\langle h_{p,u}, k+1\rangle$, $\langle h_{p,u}, *\rangle$, the first one specific and the second one general, at times $\tau'_{pm+e,u}+1$ and $\tau'_{pm+e,v}+1$, respectively.
- A pair of requests $\langle h_{p,v}, *\rangle$, $\langle h_{p,v}, k+1\rangle$, the first one general and the second one specific, at times $\tau'_{pm+e,u}+5$ and $\tau'_{pm+e,v}+5$, respectively.

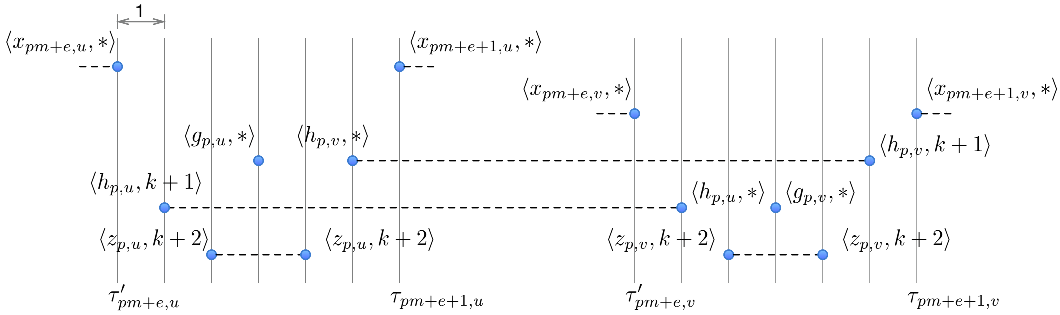

Figure 9: Gadget $\boldsymbol{\omega}_{p,e}$. Requests $\langle x_{pm+e,u}, *\rangle$, $\langle x_{pm+e+1,u}, *\rangle$, $\langle x_{pm+e,v}, *\rangle$, and $\langle x_{pm+e+1,v}, *\rangle$ are not part of this gadget; they are shown only to illustrate how gadget $\boldsymbol{\omega}_{p,e}$ fits into the overall request sequence.

Consider now possible schedules of gadget $\boldsymbol{\omega}_{p,e}$. Notice that this gadget will require 7 faults even if scheduled all by itself, since we need to make two faults on requests $\langle g_{p,u}, *\rangle$, $\langle g_{p,v}, *\rangle$, at least two faults on requests to pages $\langle z_{p,u}, k+2\rangle$, $\langle z_{p,v}, k+2\rangle$, and at least three faults on requests $\langle h_{p,u}, k+1\rangle$, $\langle h_{p,u}, *\rangle$, $\langle h_{p,v}, *\rangle$, and $\langle h_{p,v}, k+1\rangle$. (This is because if we retain page $h_{p,u}$ in slot $k+1$ until time $\tau'_{pm+e,v}+1$, so that we have a hit on $\langle h_{p,u}, *\rangle$, then we will fault on both requests $\langle h_{p,v}, *\rangle$, and $\langle h_{p,v}, k+1\rangle$.) Another important observation is that if we fault only 7 times on $\boldsymbol{\omega}_{p,e}$ then one of requests $\langle g_{p,u}, *\rangle$, $\langle g_{p,v}, *\rangle$ must be put in a vertex cache slot (that is, one of slots $1, 2, \ldots, k$). A schedule that puts $\langle g_{p,u}, *\rangle$ in a vertex slot is called a u-*schedule* of $\boldsymbol{\omega}_{p,e}$ and a schedule that puts $\langle g_{p,v}, *\rangle$ in a vertex slot is called a v-*schedule* of $\boldsymbol{\omega}_{p,e}$. (A schedule of $\boldsymbol{\omega}_{p,e}$ can be both a u-schedule schedule and a v-schedule.)

Complete reduction. Let $F = F' + 7P(m-1)$. Our request sequence σ constructed for G consists of σ' and of all $P(m-1)$ edge gadgets $\boldsymbol{\omega}_{p,e}$ defined above inserted into σ at their specified time steps. (At some time steps there will not be any requests.) To complete the proof it is now sufficient to show the following claim.

Claim 6.4. *G has a vertex cover of size k if and only if σ has a schedule with at most F faults in a cache of size $k+2$.*

($\Rightarrow$) Suppose that G has a vertex cover U of size k. We schedule σ as follows. Each vertex $j \in U$ is assigned to some uniqe vertex cache slot and all $2B$ requests $\langle x_{b,j}, *\rangle$ associated with vertex j are served in this slot. This will create kB faults. For $j \notin U$, all requests $\langle x_{b,j}, *\rangle$ are served in the junkyard slot $k+2$ at cost $2(n-k)B$. Together with the $2B$ blocking requests $\langle y_b, k+1\rangle$ and $\langle y_b, k+2\rangle$, this will give us $F' = (2n-k+2)B$ faults. For each $p = 0, 1, \ldots, P-1$, and for each $e = 0, 1, \ldots, m-2$, we do this: Let the eth edge of G be (u, v). Since U is a vertex cover, we either have $u \in U$ or $v \in U$. If $u \in U$, we use the u-schedule for gadget $\boldsymbol{\omega}_{p,e}$, with $g^*_{p,u}$ served in the cache slot associated with u. If $v \in U$, we use the v-schedule for gadget $\boldsymbol{\omega}_{p,e}$, with $\langle g_{p,v}, *\rangle$ served in the cache slot associated with v. This will give us 7 faults for this gadget, adding up to $7P(m-1)$ faults on all edge gadgets. Then the total number of faults on σ will be $F' + 7P(m-1) = F$.

($\Leftarrow$) Now suppose that there is a schedule S for σ with at most F faults. By the earlier observations, we know that S must have exactly F faults, including exactly F' faults on the request in σ' and exactly 7 faults per each edge gadget. As there are F' faults on σ', we can find some p, $0 \le p \le P-1$, such that $V_{S,pm} = V_{S,pm+1} = \cdots = V_{S,pm+m-1}$, per Corollary 6.3. Let $U = V_{S,pm}$. For each $j \in U$, all requests $x^*_{b,j}$, for $b = pm, pm+1, \ldots, pm+m-1$, must be in the same slot, that we refer to as the slot associated with vertex j. The size of U is k, and we claim that U must be a vertex cover. To show this, let (u, v) be an edge, and let e be its index. Schedule S makes 7 faults on gadget $\boldsymbol{\omega}_{p,e}$, so for this gadget it must be either a u-schedule or a v-schedule. If it is a u-schedule then $\langle g_{p,u}, *\rangle$ is served in some vertex cache slot. But the only vertex slot available in that time step is the slot associated with vertex u. This means that u must be in U. The case of a v-schedule is symmetric. Thus we obtain that either $u \in U$ or $v \in U$. This holds for each edge, implying that U is a vertex cover. This proves the claim, and the theorem. ☐

7 Weighted All-Or-One Paging

This section initiates the study of Heterogenous k-Server in non-uniform metrics. Weighted All-Or-One Paging is the natural weighted extension of All-or-One Paging (allowing general and specific requests) in which the pages have weights and the cost of retrieving a page is its weight. (This is equivalent to Heterogenous k-Server in star metrics with $\mathcal{S} = \{[k]\} \cup \{[s] : s \in [k]\}$.) The section proves the following theorem:

input: Weighted All-Or-One Paging instance (k, σ) where $\sigma_t = \langle p_t, s_t \rangle$ for $t \in [T]$

1. initialize $\mathsf{cap}[t] \leftarrow \mathsf{credit}[t] \leftarrow 0$ for each $t \in [T]$

2. assume WLOG that $\langle p_t, s_t \rangle = \langle 0, t \rangle$ for $t \in [k]$ — *k specific requests to artificial weight-zero page in each slot*

3. for $t \leftarrow k+1, k+2, \ldots, T$:

3.1. if $\langle p_t, s_t \rangle$ is a specific request with no equivalent request t' (s.t. $\langle p_{t'}, s_{t'} \rangle = \langle p_t, s_t \rangle$) in the cache:

3.1.1. evict any cached general request to page p_t, and any cached request in slot s_t

3.1.2. put t in slot s_t — *note* $\mathsf{cap}[t] = \mathsf{credit}[t] = 0$

3.2. else if $\langle p_t, s_t \rangle$ is a general request not satisfied by any cached request t' (s.t. $p_{t'} = p_t$):

3.2.1. define
$$\begin{cases} \ell_t(s) := \max\{t' \le t : s_{t'} = s\} & \text{— the last (most recent) specific request to slot } s \\ A := \{s \in [k] : \mathsf{wt}(p_t) \le 2\,\mathsf{cap}[\ell_t(s)] \text{ and slot } s \text{ does not hold a specific request}\} \\ B := \{s \in [k] : \text{slot } s \text{ holds a general request of weight at least } \mathsf{wt}(p_t)/2\} \end{cases}$$

3.2.2. while $|A| \le |B|$:

3.2.2.1. continuously raise $\mathsf{cap}[\ell_t(s)]$ for $s \in [k]$ and $\mathsf{credit}[t']$ for each cached request t', at unit rate,

3.2.2.2. evicting each request t' such that $\mathsf{credit}[t'] = \mathsf{wt}(p_{t'})$, and updating A and B continuously

3.2.3. choose a slot $s \in A \setminus B$; evict the request t' currently in slot s (if any)

3.2.4. put t in slot s — *note* $\mathsf{credit}[t] = 0$

3.3. else: classify the (already satisfied) request as *redundant* and ignore it

Figure 10: An $O(k)$-competitive online algorithm for Weighted All-Or-One Paging. For technical convenience, we present the algorithm as caching request times rather than pages, with the understanding that request t represents page p_t.

Theorem 7.1. *Weighted All-Or-One Paging has a deterministic $O(k)$-competitive online algorithm.*

The bound is optimal up to a small constant factor, as the optimal ratio for standard Weighted Paging is k. Figure 10 shows the algorithm. It is implicitly a linear-programming primal-dual algorithm. Note that the standard linear program for standard Weighted Paging doesn't have constraints that force pages into specific slots — indeed, those constraints make even the unweighted problem an NP-hard special case of Multicommodity Flow. A small example that illustrates the challenge is to take $k = 2$ slots then repeatedly request a weight-1 page (in any slot) and weight-zero pages in slots 1 and 2. The weight-zero requests force the weight-1 page to be evicted with each round, so that the cost is the number of rounds. But without the specific-slot constraints the total cost would be at most 1, so that the classical linear-program relaxation cannot be used to bound the competitive ratio.

To ease notation, assume that the first k requests are specific requests for an artificial weight-zero item in each of the k slots. (This is without loss of generality as these requests cost nothing.) Consider any execution of the algorithm on a k-slot instance σ. Assume that each request is not redundant (per Step 3.3). (This is without loss of generality, as the algorithm ignores redundant requests, and removing them doesn't increase the optimum cost.) For technical convenience, we think of the algorithm and the optimal solution as caching request *times* rather than pages, with the understanding that request t represents page p_t.

We prove the following key lemma, then the theorem.

Lemma 7.1. *Suppose that Step 3.2.2.1 is executing in response to a request t. In any optimal solution, just after the solution has responded to request t, either the optimal solution has evicted some request t' currently cached by the algorithm, or, for some slot $s \in [k]$, after the most recent specific request $\ell_t(s)$ to slot s the optimal solution has incurred cost more than $\mathsf{cap}[\ell_t(s)]$ for retrievals into s.*

Proof. Assume otherwise for contradiction. That is, there is an optimal solution that, just after responding to request t, both *(a)* caches every request t' that is cached by the algorithm, and *(b)* for every slot $s \in [k]$, has incurred cost at most $\mathsf{cap}[\ell_t(s)]$ for retrievals to slot s after the most recent specific request $\ell_t(s)$ to slot s. By *(a)*, and the fact that the optimal solution caches the current general request p_t, this optimal solution has at least $|B| + 1$ generally requested pages of weight at least $\mathsf{wt}(p_t)/2$ in the cache. We claim that *the optimal solution must cache each such page $p_{t'}$ in a slot in* A, which implies $|B| + 1 \le |A|$, contradicting the loop condition. To verify the claim, note that by definition of A each slot s not in A either has $\mathsf{cap}[\ell_t(s)] < \mathsf{wt}(p_t)/2 \le \mathsf{wt}(p_{t'})$ (so by *(b)* the optimal solution cannot cache $p_{t'}$ in s) or holds a specific request in the algorithm's cache, and therefore (by *(a)*) is also occupied by a specific request in the optimal solution's cache. ☐

Proof (Theorem 7.1). Fix an optimal solution. For each $t \in [T]$, let $x_t \in \{0, 1\}$ be an indicator variable for the event that the solution evicts request t before there is another request $t' > t$ that can be satisfied by the page/slot pair that satisfied t. Let $R \subseteq [T]$ contain the specific requests, and for each $t \in R$, let y_t be the amount the solution pays to retrieve pages into slot s_t before the next specific request to slot s_t (if any). Define the *pseudo-cost* of the optimal solution to be $\sum_{t=1}^{T} \mathsf{wt}(p_t)x_t + \sum_{t\in R} y_t$. The pseudo-cost is at most $2\,\mathsf{opt}(\sigma)$. As the algorithm proceeds, define the *residual cost* to be

$$\sum_{t=1}^{T} \max(0, \mathsf{wt}(p_t)x_t - \mathsf{credit}[t]) + \sum_{t\in R} \max(0, y_t - \mathsf{cap}[t]).$$

The residual cost is initially the pseudo-cost (at most $2\,\mathsf{opt}(\sigma)$), and remains non-negative throughout, so the total decrease in the residual cost is at most $2\,\mathsf{opt}(\sigma)$. By Lemma 7.1,[3] whenever the algorithm is raising credits and capacities at time t, there is either a cached request t' with $x_{t'} = 1$ and $\mathsf{credit}[t'] < \mathsf{wt}(p_{t'})$, or there is a slot s with $y_{t'} > \mathsf{cap}[t']$, where $t' = \ell_t(s)$. It follows that the residual cost is decreasing at least at unit rate.

On the other hand, the algorithm is raising k capacities and at most k credits, so the value of $\phi = \sum_{t=1}^{T} \mathsf{credit}[t] + \sum_{t\in R} \mathsf{cap}[t]$ is increasing at rate at most $2k$. It follows that the final value of ϕ is at most $4k\,\mathsf{opt}(\sigma)$. To finish, we show that the algorithm's cost is $O(1)$ times the final value of ϕ plus $\mathsf{opt}(\sigma)$.

Assume that the sequence ends with k specific requests (one per slot) to a weight-zero page. (This is without loss of generality because such requests don't increase the optimal cost.) Count the costs that the algorithm pays as follows:

1. *Requests remaining in the cache at the end (time T).* By the assumption on the last k requests, these cost nothing to bring in. All other requests are evicted. To finish we bound the evicted requests.

2. *Requests evicted in Line 3.2.2.2.* Each such request t' is evicted only after $\mathsf{credit}[t']$ reaches $\mathsf{wt}(p_{t'})$. So these have total weight at most $\sum_{t'=1}^{T} \mathsf{credit}[t']$.

3. *Specific requests t' evicted from slot s_t in Line 3.1.1.* Throughout the time interval $[t', t-1]$, the algorithm has $p_{t'}$ in slot $s_{t'} = s_t$, and σ has neither an equivalent specific request nor a general request to p_t (by our non-redundancy assumption). The optimal solution opt has $p_{t'}$ in slot $s_{t'}$ at time t', but not at time t, so evicts it during $[t'+1, t]$. So the total cost of such requests is at most the total weight of specific requests evicted by opt, and thus at most $\mathsf{opt}(\sigma)$.

4. *General requests evicted from slot s_t in Line 3.1.1.* By Line 3.2.3, any general request in slot s_t at time t has weight at most $2\,\mathsf{cap}[\ell_{t-1}(s_t)]$. So the total weight of such requests is at most $\sum_{t'\in R} 2\,\mathsf{cap}[t']$.

5. *General requests to page p_t evicted in Line 3.1.1.* The algorithm replaces each such general request t' by a specific request t (which it later evicts, unless the weight is zero) to the same page. Have general request t' *charge* its cost $\mathsf{wt}(p_{t'}) = \mathsf{wt}(p_t)$, and any amount charged to t' (in Item 6 below), to specific request t. (We analyze the charging scheme for Items 5 and 6 below.)

6. *General requests t' evicted in Line 3.2.3.* Have request t' charge the cost of its eviction, and any amount charged to t' to general request t. Note that by inspection $\mathsf{wt}(p_{t'}) \le \mathsf{wt}(p_t)/2$.

Charging scheme for Items 5 and 6 above. In Items 5 and 6 above, each specific request t is charged by at most one general request t' of the same weight. Each general request t is charged by at most one (general) request t' of half the weight. By induction, one can show that each general request is charged at most its weight, so that each specific request is charged at most twice its weight. So the total cost of requests in Items 5 and 6 above is at most twice the total cost of all other requests. □

8 Open Problems

The results here suggest many open problems and avenues for further research. Closing or tightening gaps left by our upper and lower bounds would be of interest. For example, for Slot-Heterogenous Paging, is the upper bound in Theorem 3.1 tight for every $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$, within $\mathrm{poly}(k)$ factors? What are the optimal randomized ratios as a function of $|\mathcal{S}|$ and k? For Slot-Laminar Paging, do the optimal ratios need to depend on the height h? For Weighted All-Or-One Paging, is the optimal randomized ratio $O(\mathrm{polylog}(k))$? The status of Heterogenous k-Server in arbitrary metric spaces is widely open. Can ratio dependent only on k be achieved? This question, while challenging, should still be easier to resolve for Heterogenous k-Server than for Generalized k-Server.

References

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

[3] The lemma gives linear constraints on the vectors (x, y). Minimizing the pseudo-cost subject to these constraints is a linear-program relaxation of the problem. Our argument implicitly defines a dual solution whose cost is our lower bound on $\mathsf{opt}(\sigma)$.

[2] C. J. Argue, Anupam Gupta, Ziye Tang, and Guru Guruganesh. Chasing convex bodies with linear competitive ratio. *Journal of the ACM*, 68(5):32:1–32:10, August 2021.

[3] Nikhil Ayyadevara and Ashish Chiplunkar. The randomized competitive ratio of weighted k-server is at least exponential. *CoRR*, abs/2102.11119, 2021.

[4] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 267–276. IEEE Computer Society, 2011.

[5] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 507–517. IEEE Computer Society, 2007.

[6] Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 493–504. IEEE Computer Society, 2017.

[7] Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 992–1001, 2018.

[8] Nikhil Bansal, Joseph (Seffi) Naor, and Ohad Talmon. Efficient online weighted multi-level paging. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 94–104. ACM, 2021.

[9] Nathan Beckmann, Phillip B. Gibbons, Bernhard Haeupler, and Charles McGuffey. Writeback-aware caching. In Bruce M. Maggs, editor, *1st Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Salt Lake City, UT, USA, January 8, 2020*, pages 1–15. SIAM, 2020.

[10] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[11] Marcin Bienkowski, Lukasz Jez, and Pawel Schmidt. Slaying Hydrae: Improved bounds for generalized k-server in uniform metrics. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[12] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[13] Allan Borodin and Ran El-Yaniv. On randomization in on-line computation. *Inf. Comput.*, 150(2):244–267, 1999.

[14] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[15] Mark Brehob, Richard J. Enbody, Eric Torng, and Stephen Wagner. On-line restricted caching. *J. Sched.*, 6(2):149–166, 2003.

[16] Mark Brehob, Stephen Wagner, Eric Torng, and Richard J. Enbody. Optimal replacement is NP-hard for nonstandard caches. *IEEE Trans. Computers*, 53(1):73–76, 2004.

[17] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. K-server via multiscale entropic regularization. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16. ACM, 2018.

[18] Sébastien Bubeck, Yuval Rabani, and Mark Sellke. Online multiserver convex chasing and optimization. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2093–2104. SIAM, 2021.

[19] Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive algorithms for restricted caching and matroid caching. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2014.

[20] Niv Buchbinder, Christian Coester, and Joseph (Seffi) Naor. Online k-taxi via double coverage and time-reverse primal-dual. In Mohit Singh and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2021.

[21] Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide. The k-server with preferences problem. In *SPAA '22: 34rd ACM Symposium on Parallelism in Algorithms and Architectures*, 2022. To appear.

[22] Ashish Chiplunkar and Sundar Vishwanathan. Metrical service systems with multiple servers. *Algorithmica*, 71(1):219–231, 2015.

[23] Ashish Chiplunkar and Sundar Vishwanathan. Randomized memoryless algorithms for the weighted and the generalized k-server problems. *ACM Trans. Algorithms*, 16(1), December 2019.

[24] Marek Chrobak and John Noga. Competitive algorithms for relaxed list update and multilevel caching. *J. Algorithms*, 34(2):282–308, 2000.

[25] Marek Chrobak, Gerhard J. Woeginger, Kazuhisa Makino, and Haifeng Xu. Caching is hard — even in the fault model. *Algorithmica*, 63(4):781–794, 2012.

[26] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1136–1147. ACM, 2019.

[27] Christian Coester and Elias Koutsoupias. Towards the k-server conjecture: A unifying potential, pushing the frontier to the circle. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 57:1–57:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[28] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Trans. Archit. Code Optim.*, 8(4), January 2012.

[29] Esteban Feuerstein. Uniform service systems with k servers. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Cláudio L. Lucchesi, and Arnaldo V. Moura, editors, *LATIN'98: Theoretical Informatics*, volume 1380, pages 23–32, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[30] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[31] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[32] Amos Fiat, Manor Mendel, and Steven S. Seiden. Online companion caching. In Rolf Möhring and Rajeev Raman, editors, *Algorithms — ESA 2002*, pages 499–511, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[33] Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theor. Comput. Sci.*, 130(1):85–99, 1994.

[34] Samuel Haney. *Algorithms for Networks With Uncertainty*. PhD thesis, Duke University, 2019.

[35] Shahin Kamali and Helen Xu. Multicore paging algorithms cannot be competitive. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 547–549, 2020.

[36] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[37] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[38] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theor. Comput. Sci.*, 324(2-3):347–359, 2004.

[39] James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *IEEE 59th Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, 7-9 October, 2018*, 2018.

[40] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B. Lee. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 406–418, 2016.

[41] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

[42] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[43] M. Mendel and Steven S. Seiden. Online companion caching. *Theoretical Computer Science*, 324(2):183–200, 2004. Online Algorithms: In Memoriam, Steve Seiden.

[44] Jignesh Patel. Restricted k-server problem. Master's thesis, Michigan State University, 2004.

[45] Mark Sellke. Chasing convex bodies optimally. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 1509–1518. Society for Industrial and Applied Mathematics, 2020.

[46] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[47] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, page 494–505, New York, NY, USA, 2007.

[48] Ying Ye, Richard West, Zhuoqun Cheng, and Ye Li. COLORIS: A dynamic cache partitioning system using page coloring. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 381–392, 2014.

[49] Wei Zang and Ann Gordon-Ross. CaPPS: cache partitioning with partial sharing for multi-core embedded systems. *Des. Autom. Embed. Syst.*, 20(1):65–92, 2016.