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 Riverside
[2]Tumult Labs
[3]Northeastern University
[4]Duke University

Abstract

It is natural to generalize the online 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 date, only a few special cases of this problem have been studied. The objective of the work presented in this paper has been to more systematically explore this generalization in the case of 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*.

In realistic settings only certain subsets of cache slots or servers would appear in requests. Therefore we parameterize the problem by specifying a family $\mathcal{S} \subseteq 2^{[k]}$ of requestable slot sets, and we establish bounds on the competitive ratio as a function of the cache size k and family $\mathcal{S}$:

- If all request sets are allowed ($\mathcal{S} = 2^{[k]} \setminus \{\emptyset\}$), the optimal deterministic and randomized competitive ratios are exponentially worse than for standard Paging ($\mathcal{S} = \{[k]\}$).
- As a function of $|\mathcal{S}|$ and 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 of laminar $\mathcal{S}$ 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. The optimal deterministic ratio for *weighted* All-or-One Paging is $\Theta(k)$. Offline All-or-One Paging is $\mathbb{NP}$-hard.

Some results for the laminar case are shown via a reduction to the 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).

Keywords: *Online algorithms, Competitive analysis, Paging, Caching, k-Server problem*

---

*Conference and journal versions of this paper appear in STACS and Algorithmica [26, 27].

†Research partially supported by National Science Foundation grants CCF-1536026 and CCF-2153723.

‡Research partially supported by National Science Foundation grants CCF-1527084 and CCF-1535972.

§Research partially supported by National Science Foundation grants CCF-1535929 and CCF-1909363.

¶Research partially supported by National Science Foundation grants CCF-1527084, CCF-1535972, CCF-1750140, CCF-1955703, an Army Research Office grant W911NF2110230, and the Indo-US Joint Center for Algorithms under Uncertainty.

‖Research partially supported by National Science Foundation grants CCF-1535929 and CCF-1909363.

**Research partially supported by National Science Foundation grants CCF-1535929 and IIS-2039945.

††Research partially supported by National Science Foundation grant CCF-1619463.

1 Introduction

The standard k-Server and Paging models assume homogenous (interchangeable) servers and cache slots. They don't model applications where servers have different capabilities e.g., content-delivery networks such as Akamai with multi-level distributed caches [46, 54], or modern cache systems that partition the slots, sometimes dynamically, with differential accessibility by specific processors, cores, processes, threads, or page sets (e.g., [34, 47, 55–58]).

This motivates us to generalize the online 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 [23, 51]. Significant insights into k-Server and its extensions [6, 7, 12, 24, 28, 29, 35, 43] have been obtained by first examining simpler versions of the problem. Following this strategy, we embark on a systematic study of the Heterogenous k-Server problem when the underlying metric space is uniform or has a star topology. 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 k cache slots, to be satisfied by having a copy of p in some slot in S. We call this problem *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.)

As common in the study of online optimization problems, we employ competitive analysis. We use the standard definifion of the competitive ratio of an online algorithm (see Section 2). In the discussion below, by the *optimal competitive ratio* for a given problem we mean the smallest competitive ratio achievable by an online algorithm.

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 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, 12].

- We show that the optimal deterministic and randomized competitive ratios for Slot-Heterogenous Paging are at least $\Omega(2^k/\sqrt{k})$ and $\Omega(k)$, respectively (Theorems 3.2 (i) and 3.3).

Hence, the optimal ratios for Slot-Heterogenous Paging are exponentially worse than for standard Paging. The proofs of Theorems 3.2 and 3.3 employ some novel ideas that may be useful for other problems: the lower bound in Theorems 3.2 (i) uses an adversary argument that requires the construction of a set family not yet studied in the literature, while the proof of Theorem 3.3 is carried out via a reduction *from* standard Paging with a cache of size $\exp(\Theta(k))$.

The large competitive ratios in these lower bounds occur only for instances that use exponentially many distinct request sets S. In realistic settings only certain subsets of cache slots or servers can appear in requests, namely those that represent capabilities or functionalities relevant in a given setting. This motivates us to study the optimal ratios as a function of the cache size k and the family $\mathcal{S}$ of requestable slot sets, and to try to identify natural families that admit more reasonable ratios.

- We show that the optimal deterministic ratio is at most $k^2|\mathcal{S}|$ for any family $\mathcal{S}$ (Theorem 3.1). Theorem 3.2 (ii) shows a complementary lower bound: for infinitely many values of k there is a family $\mathcal{S} \subseteq 2^{[k]}$, for which 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.

Page-Laminar Paging (Section 4). We take a brief detour to consider *Page-Subset Paging*, a natural generalization of Paging in which each request is a set P of *pages* from an arbitrary but fixed family $\mathcal{P}$, and is satisfiable by fetching any page from P into any slot in the cache. We focus on its special case of *Page-Laminar Paging*, where this set family $\mathcal{P}$ is laminar.

- We show 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 $H_k = \sum_{i=1}^{k} 1/i = \ln k + O(1)$ (Theorem 4.1).

The proof is by a reduction that 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.

Slot-Laminar Paging (Section 5). We then return to Slot-Heterogenous Paging, now considering the specific structure of $\mathcal{S}$, showing better bounds when $\mathcal{S}$ is laminar. This case, which we call *Slot-Laminar Paging*, is intended to model scenarios where server capabilities are hierarchical. For example, some proposed cache partitioning systems (see [58] and the references therein) divide the cache into parts, some exclusive to certain processes and other fully or partially shared. Such a cache partitioning strategy can be modeled as a simple laminar structure.

Laminarity implies that $|\mathcal{S}| < 2k$, so (per Theorem 3.1 above) the optimal deterministic ratio is $O(k^3)$.

- We show that the optimal deterministic and randomized ratios for Slot-Laminar Paging are $O(h^2 k)$ and $O(h^2 \log k)$, where $h \leq k$ is the height of $\mathcal{S}$ (Theorem 5.1). We next tighten the deterministic bound to $O(hk)$ (Theorem 5.2).

The proof of Theorem 5.1 is via a reduction to Page-Laminar Paging (discussed above), while the proof of Theorem 5.2 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.

Reducing Slot-Laminar Paging to Page-Laminar Paging. The reduction of Slot-Laminar Paging to Page-Laminar Paging in Theorem 5.1 is achieved 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 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.

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. Recent independent work on All-or-One Paging [23] has shown that the optimal deterministic ratio is twice that of Paging, to within an additive constant.

- We show that the optimal randomized ratio of All-or-One Paging is also at least twice that for Paging (Theorem 6.1), while Theorem 5.1 upper bounds the optimal randomized ratio to within a constant factor of that for Paging. We also show that the offline problem is $\mathbb{NP}$-hard (Theorem 6.2), in sharp contrast to even k-Server, which can be solved in polynomial time for arbitrary metrics.

Weighted All-Or-One Paging (Section 7). To gain insight into Heterogenous k-Server in non-uniform metrics we investigate *Weighted All-Or-One Paging*, which 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.

- We show that the optimal deterministic ratio for Weighted All-Or-One Paging is $O(k)$, matching the ratio for standard Weighted Paging up to a constant factor (Theorem 7.1).

problem	set family $\mathcal{S}$ (or $\mathcal{P}$)	deterministic	randomized	where
Slot-Heterogenous Paging	$2^{[k]} \setminus \{\emptyset\}$	$\leq k2^k$	$\leq O(k^2 \log k)$	via [7, 12]
”	arbitrary $\mathcal{S}$	$\leq k \min(\lvert\mathcal{S}^*\rvert, \mathsf{mass}(\mathcal{S}))$		Thm. 3.1
One-of-m Paging, $m \approx k/2$	$\binom{[k]}{m}$	$\geq \Omega(2^k/\sqrt{k})$	$\geq \Omega(k)$	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	$\leq (2h-1)k$	$\leq 3h^2 H_k$	Thms. 5.1, 5.2
All-or-One Paging	$\{[k]\} \cup \{\{s\} : s \in [k]\}$	$\geq 2k-1$	$\geq 2H_k - 1$	[23, 38], Thm. 6.1
”	”	$\leq 2k+14$		[23]
Weighted All-Or-One Paging	$\{[k]\} \cup \{\{s\} : s \in [k]\}$	$\leq O(k)$		Thm. 7.1
Page-Subset Paging restr. to	$\mathcal{P} = \binom{\text{all pages}}{m}$	$\geq \binom{k+m}{k} - 1$		[35]
”		$\leq k(\binom{k+m}{k} - 1)$	$\leq O(k^3 \log m)$	[24]
Page-Laminar Paging	$\mathcal{P}$ laminar, height h	$\leq hk$	$\leq hH_k$	Thm. 4.1

Table 1: Summary of upper ($\leq$) and lower ($\geq$) bounds on optimal competitive ratios. Here $\mathsf{mass}(\mathcal{S}) = \sum_{S\in\mathcal{S}} |S|$ and $\mathcal{S}^* = \bigcup_{S\in\mathcal{S}} 2^S$. By $\binom{X}{m}$ we denote the family of all m-element subsets of a set X. The lower bound for One-of-m Paging holds for some but not all m and k—see Theorem 3.2(ii). The upper bound for Slot-Laminar Paging in the deterministic case (Theorem 5.1) is in fact $2 \cdot \mathsf{mass}(\mathcal{S}) - k$, which is at most $(2h-1)k$. Also, offline All-or-One Paging and its generalizations are $\mathbb{NP}$-hard (Theorem 6.2), as is offline Page-Subset Paging ([24]).

The algorithm in the proof is implicitly a linear-programming primal-dual algorithm. With this approach the crucial obstacle to overcome is that 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 show how to augment the linear program to partially model the slot constraints.

Related work. Paging and k-Server have played a central role in the theory of online computation since their introduction in the 1980s [13, 48, 53]. For k-Server, the optimal deterministic ratio is between k and $2k-1$ [42]. Recent work [33] offers hope for closing this gap, and substantial progress towards resolving the randomized case has been reported in [4, 18, 19]. For Weighted Paging the optimal ratios are k (deterministic) and $\Theta(\log k)$ (randomized) [1, 5, 36, 44, 45, 49, 53].

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 $O(\log^2 k)$ [21]. Better bounds are possible given further restrictions on the sets, as in *Companion Caching*, which models a cache partitioned into set-associative and fully associative parts [16, 17, 50]. 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—is easier than Heterogenous k-Server. While the two problems are different for many metric classes, they can be shown to be equivalent in metric spaces with no isolated points, such as Euclidean spaces. The $\mathbb{NP}$-hardness result for Restricted Caching from [17] 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 [40], *Multi-Level Caching*, where the cost to access a slot depends on the slot [30], 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. [39]).

Patel's master thesis [51] 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 and a $2k+14$ upper bound [23]. Earlier work in [38] presents a $2k-1$ lower bound and a $3k$ upper bound on deterministic algorithms.

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 requested point in its space [43]. 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) [7,12]. 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 [37]. For this problem the deterministic and randomized ratios are at least (respectively) doubly exponential [6,7] and exponential [3,25], even in uniform metrics.

For Page-Subset Paging restricted to m-element sets of pages, the optimal ratios are between $\binom{k+m}{k}-1$ and $k(\binom{k+m}{k}-1)$ (deterministic) and between $\Omega(\log km)$ and $O(k^3 \log m)$ (randomized) [24, 35]. This problem has been studied as uniform *Metrical Service Systems with Multiple Servers (MSSMS)*. MSSMS is the generalization of k-Server where each request is a *set* of points, one of which needs to be covered by some server.

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 [20]. For k-Chasing, no online algorithm is competitive even for $d = k = 2$ [20], while for $k = 1$ the ratios grow with d [2,52].

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 [22,32].

2 Formal Definitions

Most of the set-theoretic notation and terminology used in our paper is standard. By $\mathbb{R}$ we denote the set of real numbers. For a non-negative integer j, we use notation $[j]$ for the set of first j positive integers: $[j] = \{1, 2, \ldots, j\}$. If X is any set then 2^X denotes the power set of X, that is $2^X = \{Y : Y \subseteq X\}$, and (adapting the notation for binomial coefficients) we use notation $\binom{X}{m}$ for the family of all m-element subsets of X.

Heterogenous k-Server. In this natural generalization of the well-known k-Server problem, each request specifies a subset of servers that may serve the request. As in k-Server, we are given k servers, numbered $1, 2, \ldots, k$, that reside in a metric space $\mathcal{M}$. We are also given a family $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$ of requestable sets of servers. The objective is to serve a given request sequence $\{\rho_t\}_{t=1}^T$, where each request is specified as a pair $\rho_t = \langle r_t, S_t \rangle$, for some point $r_t \in \mathcal{M}$ and set $S_t \in \mathcal{S}$. To serve the request $\rho_t = \langle r_t, S_t \rangle$, one of the servers in S_t must be moved to r_t. The cost of this move is the distance from its previous location to r_t. The overall cost of serving the request sequence $\{\rho_t\}_{t=1}^T$ is the total movement cost of all servers.

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 *cache configuration* C is a function that specifies the content of each slot $s \in [k]$; this content is either a single page (said to be assigned to the slot) or empty. Configuration C is said to *satisfy* a request $\langle p, S \rangle$ if it assigns page p to at least one slot in S. A *solution* for a given request sequence σ is a sequence $\{C_t\}_{t=1}^T$ of cache configurations such that, for each $t \in [T]$, C_t satisfies request σ_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 $t = 1$). An event when C_{t-1} does not assign p_t to any slot in S_t is called a *fault*. Obviously a fault triggers a retrieval but, while this is not strictly necessary, it is convenient to also allow an algorithm to make spontaneous retrievals, either by fetching into the cache a non-requested page or by moving pages within the cache.

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 $|\mathcal{S}| \leq 2k$.) A laminar family $\mathcal{S}$ can be naturally represented by a forest (a collection of disjoint rooted trees), with a set R being a descendant of R' if $R \subseteq R'$. When discussing Slot-Laminar Paging we routinely use tree-related terminology. For example, we refer to some sets in $\mathcal{S}$ as leaves, roots, or internal nodes, and we also use other common relations between the nodes of a rooted tree: of being a child, parent, or ancestor. The height h of a laminar family $\mathcal{S}$ is one more than the maximum height of a tree in $\mathcal{S}$, that is the maximum h for which $\mathcal{S}$ 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*, of the form $\langle p, [k] \rangle$, requiring page p to be in at least one slot of the cache, and *specific*, of the form $\langle p, \{j\} \rangle$, $j \in [k]$, 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 $\langle p, \{j\} \rangle$.

Weighted All-Or-One Paging. This is the natural extension of All-or-One 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.

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 a set of m slots.

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) such 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.)

Page-Laminar Paging. This is the restriction of Page-Subset Paging to instances where $\mathcal{P}$ is *laminar*.

Generalized k-Server. In this variant of k-Server, 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 requested point in its space [43].

Approximation algorithms. An algorithm $\mathbb{A}$ for a given cost minimization problem is called a *c-approximation algorithm* if, for each instance σ, $\mathbb{A}$ satisfies $\mathsf{cost}_{\mathbb{A}}(\sigma) \leq c \cdot \mathsf{opt}(\sigma) + b$, where $\mathsf{cost}_{\mathbb{A}}(\sigma)$ 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. We follow the standard convention that when we are considering $\mathbb{A}$ as an offline algorithm, the constant b must be 0.

Online algorithms and competitive ratio. In the online variants of the paging and server problems the requests arrive online, one per time step, and an online algorithm must satisfy each request before subsequent requests are revealed. An online algorithm $\mathbb{A}$ is called *c-competitive* if $\mathbb{A}$ is a c-approximation algorithm. As common in the literature, we use the term "optimal deterministic (resp. randomized) competitive ratio" to refer to the optimal ratio of of a deterministic (resp. randomized) online algorithm for the given problem.

For Slot-Heterogenous Paging (or Page-Subset Paging) problems that we study, we assume that the algorithm knows in advance the underlying set family $\mathcal{S}$ (or $\mathcal{P}$). This assumption is natural, given that each set family defines a different computational problem, and it's needed for algorithms customized to a specific family (for example, for All-or-One Paging). On the other hand, the generic Algorithm ExhSearch for Slot-Heterogenous Paging in Section 3.1 does not use any information about the family $\mathcal{S}$. Its adaptation to Slot-Laminar Paging in Section 5.2, called RefSearch, is presented under the assumption that the laminar family $\mathcal{S}$ is known. With some care this assumption can be removed, although at the cost of introducing distracting complications in the proof.

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. Each request $\langle p, S\rangle$ in the instance of Slot-Heterogenous Paging is simulated by a long sequence $\rho_{p,S}$ of requests in the instance of Generalized k-Server. (Recall that a request in Generalized k-Server is given as a vector of points, one for each server.) In $\rho_{p,S}$, each request specifies point p for each server in S. For each other server, in $[k] \setminus S$, we choose two arbitrary different points in its space, and the requests in $\rho_{p,S}$ alternate between these two points. If we make $\rho_{p,S}$ sufficiently long, these alternating requests force any competitive online algorithm serving $\rho_{p,S}$ to move one of the servers in S to p, which specifies the slot in S to use for serving request $\langle p, S\rangle$. This reduction works both for deterministic and randomized algorithms.

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 $\mathcal{S} = 2^{[k]} \setminus \{\emptyset\}$).

Theorem 3.2 (i) (Section 3.2) and Theorem 3.3 (Section 3.3) show that these bounds are tight within $\operatorname{poly}(k)$ 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, Theorem 3.1 and Theorem 3.2 (ii) imply that the optimal deterministic ratio is $O(k^{m+1})$ and $\Omega((4k/m)^{m/2}/\sqrt{m})$.

3.1 Upper bounds for deterministic Slot-Heterogenous Paging

This section gives upper bounds on the optimal deterministic competitive ratios for Slot-Heterogenous Paging with any slot-set family $\mathcal{S}$, as a function of $\mathsf{mass}(\mathcal{S}) = \sum_{S\in\mathcal{S}} |S|$ and $|\mathcal{S}^*|$, where $\mathcal{S}^* = \bigcup_{S\in\mathcal{S}} 2^S$. These two quantities capture the "size" or the "complexity" of $\mathcal{S}$, although in a different way, and their mutual relation depends on the structure of $\mathcal{S}$. In general we have $\mathsf{mass}(\mathcal{S}) \le k|\mathcal{S}| \le k|\mathcal{S}^*|$. But $|\mathcal{S}^*|$ could be as small as $O(\mathsf{mass}(\mathcal{S})/k)$ (say, when $\mathcal{S} = 2^{[k]} \setminus \{\emptyset\}$) or it could be exponentially larger (say, when $\mathcal{S} = \{[k]\}$).

In the theorem below, the bound using $\mathsf{mass}(\mathcal{S})$ follows from an easy counting argument. The proof of the second bound, in terms of $|\mathcal{S}^*|$, uses the rank method of [7]. This method estimates the number of steps in one phase of a natural exhaustive-search algorithm by the rank of a certain upper-triangular matrix. Our argument is a natural refinement of this approach—in essence, the original proof from [7] directly applies to the case when $\mathcal{S}$ is the power set of $[k]$, and we show how to customize it to an arbitrary given set family $\mathcal{S}$.

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. Assume without loss of generality that the algorithm faults in each step t, that is C_{t-1} does not satisfy $\sigma_t = \langle p_t, S_t\rangle$. (Otherwise first remove such requests; this doesn't change the algorithm's cost or increase the optimal cost.)

We first bound the maximum length of any phase. The argument is the same for each phase. To ease notation assume the phase is the first (with $\ell = 1$). Let L be the length of the phase. By the initial assumption, the following holds:

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 $t \leftarrow 1, 2, \ldots, T$:

2.1. if current configuration C_{t-1} satisfies request σ_t: ignore the request (set $C_t = C_{t-1}$)

2.2. else:

2.2.1. if any configuration satisfies all requests $\sigma_\ell, \sigma_{\ell+1}, \ldots, \sigma_t$: let C_t be any such configuration

2.2.2. 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.

(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.*

The final configuration C_L in the phase satisfies all requests in the phase. In particular, for each given set $S \in \mathcal{S}$, if the phase has a request $\langle p, S\rangle$ for some page p, then C_L has p in some slot in S. Associate request $\langle p, S\rangle$ with this slot. Then different requests involving S will be associated with different slots. Therefore, *(i) there are at most $|S|$ distinct requests in the phase that use any given set $S \in \mathcal{S}$.* Property (UT) implies that *(ii) every request σ_t in this phase is distinct* (indeed, for any $t' < t$, C_{t-1} satisfies $\sigma_{t'}$ but not σ_t). Observations (i) and (ii) imply the bound $L \le \sum_{S\in\mathcal{S}} |S| = \mathsf{mass}(\mathcal{S})$.

(As an aside, the above argument uses only that every request in the phase is distinct, a weaker condition than (UT). Given only that property, the above bound on L is tight for every $\mathcal{S}$ in the following sense: consider any configuration C that puts a distinct page in each slot $s \in [k]$, and a request sequence σ that requests in any order every pair $\langle p, S\rangle$ such that $S \in \mathcal{S}$ and C assigns p to a slot in S. Then σ is satisfied by the single configuration C, while having $\mathsf{mass}(\mathcal{S})$ distinct requests.)

Next we give a second bound on L that is tighter for some families $\mathcal{S}$. Identify each page p with a distinct but arbitrary real number[1]. For each cache configuration C_t, let $C_t^i \in \mathbb{R}$ denote (the real number associated with) 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} satisfies $\sigma_t = \langle p_t, S_t\rangle$, that is $C_{s-1}^i = p_t$ for some $i \in S_t$. So Property (UT) implies that M is upper-triangular and non-zero on the diagonal. Thus 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 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, $M = AB$; A and B (and M) have rank at most $|\mathcal{S}^*|$. And M has rank L, so $L \le |\mathcal{S}^*|$. To bound the optimum cost, 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 $[t'+1, t''+1]$.

[1] We use real numbers for convenience. The proof works for elements from any sufficiently large field.

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 the two upper bounds on L imply Theorem 3.1. □

3.2 Lower bounds for deterministic Slot-Heterogenous Paging

We establish our 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 Lemma 3.1, which establishes a general lower bound on the competitive ratio for Slot-Heterogenous Paging for some requestable set families $\mathcal{S}$. Namely, if, for a given $\mathcal{S}$, we can identify a family $\mathcal{Z} \subseteq 2^{[k]}$ with certain properties then no online algorithm can have competitive ratio smaller than $|\mathcal{Z}|$. The proof of Theorem 3.2 then constructs such families $\mathcal{Z}$ for appropriate families $\mathcal{S}$ of requestable sets.

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

Lemma 3.1. *For some $\mathcal{S} \subseteq 2^{[k]}$, suppose there are two 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 $S \nsubseteq Z$ and $S \nsubseteq \overline{Z}$ for all $Z \in \mathcal{Z} \setminus \{Y\}$.

Then the optimal deterministic 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. We consider a suitably chosen fixed set of $2|\mathcal{Z}|$ solutions. At each step, the adversary chooses a request, using one of the sets in $\mathcal{G}$, which forces the algorithm to fault but among our chosen solutions only at most two will fault. At the end, the algorithm's total cost is at least $|\mathcal{Z}|$ times the average cost of these chosen 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 [48], for Metrical Task Systems [15] and for Generalized k-Server on uniform metrics [43]).

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 Q_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 Q_X for some X.

At each step, if the current configuration of $\mathbb{A}$ is Q_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 (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 Q_Z. It stays in Q_Z for the whole computation, except that on requests $\langle p_a, S\rangle$ that are not served by Q_Z (that is, when in configuration Q_Z all slots of S contain p_{1-a}), it retrieves p_a to any slot $j \in S$, serves the request, then retrieves p_{1-a} back into slot j, restoring configuration Q_Z. This costs 2.

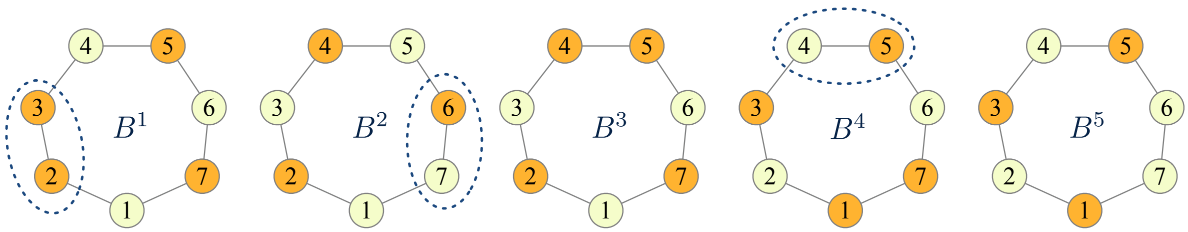

Figure 2: Illustration of the proof of Theorem 3.2 Part (ii) for $k = 35$, $m = 6$, and $\ell = 7$. The figure shows the partition of all slots into $m - 1 = 5$ sets $B^1, \ldots, B^5$, each represented by a cycle. To avoid clutter, each slot b^e_c is represented by its index c within B^e. The picture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

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). Then, for all $Z \in (\mathcal{Z} \cup \widetilde{\mathcal{Z}}) \setminus \{Y, \overline{Y}\}$ we have $S \cap Z \neq \emptyset$, implying that configuration Q_Z has a slot in S that contains p_0—in other words, configuration Q_Z satisfies S. Also, either $S \cap Y \neq \emptyset$ or $S \cap \overline{Y} \neq \emptyset$, so one of the configurations Q_Y or $Q_{\overline{Y}}$ 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 at most k for 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. □

Proof of Theorem 3.2. Part (i). Recall that $m = \lfloor (k+1)/2 \rfloor$. First consider the case when 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 $Z \in \mathcal{Z}$ satisfies $|Z| = |S| > |\overline{Z}|$, so $S \not\subseteq \overline{Z}$, while $S \subseteq Z$ implies $Z = S$.) Thus, 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, let $k' = k - 1$. Then apply Part (i) to k' 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. This proves Part (i).

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 observe that

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

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 (1) indeed holds. Now, recalling that $f(m,k) = \binom{m-1}{m/2}(k/(m-1))^{m/2}$, we derive

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

This gives us one estimate on the competitive ratio in Theorem 3.2(ii). To obtain a second estimate, squaring both sides of (2), we obtain

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

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

It is worth noting that, as can be seen from the proofs in this section, both Theorem 3.2 and Lemma 3.1 hold even for instances with just two pages.

3.3 Lower bound for randomized Slot-Heterogenous Paging

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

Theorem 3.3. *The optimal randomized ratio for One-of-m Paging with cache size k and $m = \lfloor k/2 \rfloor$ is $\Omega(k)$, even for inputs that use only two pages.*

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 $H_{N-1} = \Theta(\log N)$ [36]. This and the next lemma imply the theorem.

Lemma 3.2. *Every $f(k)$-competitive (randomized) online algorithm $\mathbb{A}$ for One-of-m Paging with $m = \lfloor k/2 \rfloor$ can be converted into an $O(f(k))$-competitive (randomized) online algorithm $\mathbb{B}$ for standard Paging with N pages and a cache of size $N-1$, where $N = 2^{\Theta(k)}$.*

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 $N = \lfloor e^{k/16} \rfloor$.

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. [14, §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 $t \geq 1$.) The goal is to minimize the number of times the rat moves, that is $|\{t \in [T] : R_t \neq R_{t-1}\}|$.

The claimed algorithm $\mathbb{B}$ for Paging will work by reducing a given instance μ on a set $\mathcal{H}$ of N holes to an instance σ of One-of-$(k/2)$ Paging, simulating $\mathbb{A}$ on σ, and converting its solution to a solution for μ. This instance σ uses just two pages, p_0 and p_1. To describe the reduction, we need a few more definitions and observations.

For two disjoint sets $S_0, S_1 \subseteq [k]$, by $Q(S_0, S_1)$ we denote the cache configuration that assigns p_0 to slots in S_0 and p_1 to slots in S_1, with the remaining slots empty. A configuration $Q(S_0, S_1)$ is called *balanced* if $|S_0| = |S_1| = k/2$. (This obviously implies that $\overline{S}_0 = S_1$, where $\overline{S}_0 = [k] \setminus S_0$.) Note that the Hamming distance between any two balanced configurations (that is, the number of slots whose contents differ in these configurations) is always even.

The following easy observation will be useful:

Observation 3.3. *Let $S \subseteq [k]$ with $|S| = k/2$. Any request $\langle p_0, S \rangle$ is satisfied by every balanced configuration except $Q(\overline{S}, S)$, and any request $\langle p_1, S \rangle$ is satisfied by every balanced configuration except $Q(S, \overline{S})$.*

For any set $\mathcal{C}$ of cache configurations, a *forcing sequence for $\mathcal{C}$* is a request sequence such that the cache configurations that satisfy all requests in this sequence without cost are exactly those in $\mathcal{C}$.

Claim 3.4. *Let $\mathcal{C}$ be a set of balanced cache configurations such that any two configurations in $\mathcal{C}$ are at Hamming distance strictly greater than 2. Then there is a request sequence $\psi(\mathcal{C})$ that is forcing for $\mathcal{C}$.*

To verify the claim, take $\psi(\mathcal{C})$ to be the sequence formed by (any ordering of) all those allowed requests that are satisfied by all configurations in $\mathcal{C}$. Specifically, $\psi(\mathcal{C})$ consists of 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 $Q(S_0, S_1) \in \mathcal{C}$. By definition, each configuration in $\mathcal{C}$ satisfies all requests in $\psi(\mathcal{C})$.

It remains to show that for any configuration $Q(S'_0, S'_1) \notin \mathcal{C}$ there is a request in $\psi(\mathcal{C})$ not satisfied by $Q(S'_0, S'_1)$. In the case that $Q(S'_0, S'_1)$ is balanced, we can take this request to be $\langle p_1, S'_0 \rangle$, because, by Observation 3.3, it is included in $\psi(\mathcal{C})$ but it is not satisfied by $Q(S'_0, S'_1)$.

Next consider the case that $Q(S'_0, S'_1)$ is not balanced. Without loss of generality, assume that $Q(S'_0, S'_1)$ has no empty slots, since filling empty slots with p_0 or p_1 can only increase the number of requests in $\psi(\mathcal{C})$ satisfied by $\psi(\mathcal{C})$. Likewise assume by symmetry that $|S'_0| > k/2$. Let B_0 and B'_0 be any two size-$k/2$ subsets of S'_0 such that $|B_0 \setminus B'_0| = |B'_0 \setminus B_0| = 1$. Then $Q(B_0, \overline{B}_0)$ and $Q(B'_0, \overline{B}'_0)$ are at Hamming distance 2 so both cannot be in $\mathcal{C}$. Assume without loss of generality that $Q(B_0, \overline{B}_0)$ is not in $\mathcal{C}$. Then request $\langle p_1, B_0 \rangle$ is satisfied by every configuration in $\mathcal{C}$ because, by Observation 3.3, the only balanced configuration that doesn't satisfy this request is $Q(B_0, \overline{B}_0)$, which is not in $\mathcal{C}$. So $\langle p_1, B_0\rangle$ is in $\psi(\mathcal{C})$. On the other hand, since $B_0 \subseteq S'_0$, request $\langle p_1, B_0\rangle$ is not satisfied by $Q(S'_0, S'_1)$. This proves Claim 3.4.

We now describe how Algorithm $\mathbb{B}$ computes its solution $R = (R_1, \ldots, R_T)$ for a given request sequence μ. To streamline presentation, we present it first as an offline algorithm. At the beginning $\mathbb{B}$ chooses any collection $\mathcal{C}$ of N balanced configurations such that the Hamming distance between every two distinct configurations in $\mathcal{C}$ is at least $k/16$. Such a collection $\mathcal{C}$ can be constructed using a greedy method, as in [12]. (Here we only need existence, which can be also established by a probabilistic proof: if one forms $\mathcal{C}$ by randomly and uniformly sampling N times with replacement from the balanced configurations, then by a standard Chernoff bound and the naïve union bound $\mathcal{C}$ has the required property with positive probability.) Algorithm $\mathbb{B}$ lets $\mathcal{H} = \mathcal{C}$, identifying holes with cache configurations. Then, for each time $t \in [T]$, it replaces the request C_t in μ by $\psi(\mathcal{C} \setminus \{C_t\})^k$, that is, k repetitions of the forcing sequence for $\mathcal{C} \setminus \{C_t\}$. (This sequence exists if k is large enough, by Claim 3.4 and the choice of $\mathcal{C}$.) This produces sequence σ, an instance of One-of-m Paging, with $m = k/2$.

Next, $\mathbb{B}$ simulates $\mathbb{A}$ on σ. Let D be a solution for σ produced by $\mathbb{A}$. Without loss of generality we can assume that, for each $t \in [T]$, as D responds to $\psi(\mathcal{C} \setminus \{C_t\})^k$ it uses at least one configuration $P \in \mathcal{C} \setminus \{C_t\}$. (This is because otherwise D would incur cost at least k on each such $\psi(\mathcal{C} \setminus \{C_t\})^k$. We could then replace D by a solution D' that at the end of each such forcing sequence moves into any configuration in $\mathcal{C} \setminus \{C_t\}$. Being in a different configuration could increase the future cost of D' by at most k, so the total cost of D' will be at most twice that of D.) Finally $\mathbb{B}$ takes R_t to be P. The produced sequence $R = (R_1, \ldots, R_T)$ is a valid solution to μ, because $R_t \in \mathcal{C} \setminus \{C_t\}$ for $t \in [T]$.

To complete the description of $\mathbb{B}$, it remains to observe that R can indeed be produced in an online fashion, because R_t does not depend on any future requests in μ. Also, $\mathbb{B}$ is deterministic if $\mathbb{A}$ is.

Claim 3.5. $\mathsf{opt}(\sigma) \leq k\,\mathsf{opt}(\mu)$.

To prove this claim, 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 $R^*_0 = R_0$, then, for each $t \in [T]$, responds to $\psi(\mathcal{C} \setminus \{C_t\})^k$ by having its cache in configuration $R^*_t \in \mathcal{C} \setminus \{C_t\}$ for all requests in $\psi(\mathcal{C} \setminus C_t)^k$. For each $t \in [T]$, the response of D^* to $\psi(\mathcal{C} \setminus \{C_t\})^k$ costs 0 if $R^*_{t-1} = R^*_t$ (the rat didn't move) and otherwise at most k (to transition the cache from R^*_{t-1} to R^*_t). This proves Claim 3.5.

Claim 3.6. *Algorithm $\mathbb{B}$ is $O(f(k))$-competitive.*

Since $\mathbb{A}$ is $f(k)$-competitive, $\mathsf{cost}(D) \leq f(k)\mathsf{opt}(\sigma) + c_k$, where $c_k \in \mathbb{R}$ is determined by k alone. Whenever the rat moves (i.e., $R_{t-1} \neq R_t$), by the definition of $\mathcal{C}$, the Hamming distance between R_{t-1} and R_t is $\Omega(k)$, so D pays $\Omega(k)$ to transition from R_{t-1} to R_t (possibly in multiple steps). Using Claim 3.5, we obtain that $\mathsf{cost}(R) = O(\mathsf{cost}(D)/k) = O((f(k)\mathsf{opt}(\sigma) + c_k)/k) = O(f(k)\mathsf{opt}(\mu) + c_k/k)$. So R is an $O(f(k))$-competitive solution for μ. This proves Claim 3.6, completing the proof of the lemma. □

4 Upper Bounds for 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, whose height we denote by h. (See the example in Figure 3.) To our knowledge, this problem has not been yet studied in the literature. In particular, we do not know whether the optimum solution can be computed in polynomial time.

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.*

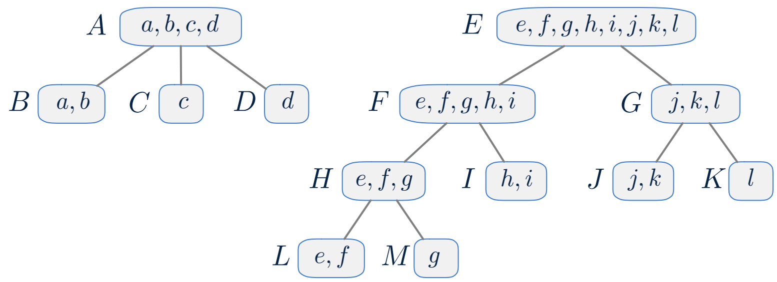

Figure 3: An example of a laminar family $\mathcal{P}$ of height 4.

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 [53] and a randomized H_k-competitive online algorithm [1]. Theorem 4.1 follows directly from composing these known results with the following lemma.

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 time step t and 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 $t' = \max\{i \le t : P_i \subset P\}$. If there is no such request (t' is undefined or P is a leaf), 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 $\mathsf{opt}(\sigma) \le h\,\mathsf{opt}(\pi)$.

For any requested set P, define a *P-phase* of π to be a maximal contiguous interval $[i, j] \subseteq [1, T]$ such that $\pi_t \not\subset P$ 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 P-phase $[i, j]$ is contained in some $c_i(P)$-phase. By definition of σ, each request to P in the given instance π in interval $[i, j]$ is replaced in σ by a request to P's preferred page, $p_i(P)$.

Example. Consider the laminar family $\mathcal{P}$ in Figure 3. Consider also a request sequence π whose requests $\pi_{61}, \pi_{62}, \ldots, \pi_{79}$ are shown in the table below:

time step :	$\cdots$	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	$\cdots$
sequence π :	$\cdots$	A	B	H	E	C	K	A	I	D	E	M	B	H	A	D	E	G	A	F	$\cdots$
E-phases :	$\cdots$	–	–	H	–	–	K	–	I	–	–	M	–	H	–	–	–	G	–	F	$\cdots$
F-phases :	$\cdots$	–	–	H	–	–	–	–	I	–	–	M	–	H	–	–	–	–	–	–	$\cdots$
sequence σ :	$\cdots$	a	a	e	e	c	l	c	h	d	h	g	a	g	a	d	g	l	d	g	$\cdots$

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:

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

3.2. 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 cached.*

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 $q_t \in P$ by p, where q_t is defined 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 4: The algorithm that transforms (π, C) into (σ, C'), by repairing each phase.

The third row in the table shows E-phases, marking the beginning of each phase with the request at that step. The fourth row shows F-phases. Assume that at time 61 the preferred page of each set in $\mathcal{P}$ is the leftmost page in the leftmost leaf of its subtree. For example, $p_{61}(E) = e$. Then the fifth row shows the sequence of preferred pages that forms the resulting sequence σ.

To prove that $\mathsf{opt}(\sigma) \le h\,\mathsf{opt}(\pi)$, we will start with an optimal solution for π and convert it into a solution of σ while increasing the cost at most by a factor of h. So let $C = (C_1, \ldots, C_T)$ be an optimal solution for π. The conversion of C into a solution for σ is given in Figure 4, and is described as an algorithm that incrementally modifies, or "repairs", both C and π, phase by phase. It maintains 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 request 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 $j \neq T$.*

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).*

By the definition of P-phases, each leaf set P has only one P-phase $[1, T]$, 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 the 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 Step 3.2 of the repair algorithm) after the repair C' has the preferred page p cached, and thus C' satisfies the modified request (for p).

The other case is when π' requested page at time t is not P. Then the repair doesn't modify the request in π' at time t. 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 an already repaired 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 $[i, j]$. 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 Step 3.2 of the algorithm), the repair of the phase replaced some page q_t in the cache at time t by the preferred page $p = p_i(P)$. 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, and now our invariant implies that 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 p at time i. Thus the cost cannot increase at time i.

Next, 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 Step 3.2 of the 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). Thus the repair removed a retrieval of p at time t.

Overall, for each retrieval introduced at times in $[i, j]$, another was removed, and therefore the cost at times in $[i, j]$ didn't increase. The cost increase at time $j + 1$, if any, can only be caused by the repair at time $t = j$ that replaced q_j by p, so this increase is at most 1. This completes the proof of the claim. □

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.

Since $i > 1$, the definition of P-phases implies that at time i 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 $[i, j]$ 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. □

As explained earlier, Claims 4.2, 4.3 and 4.4 imply the lemma, thus the proof is now complete. □

5 Slot-Laminar Paging

In this section we prove upper bounds for Slot-Laminar Paging given in Table 1. Recall that in Slot-Laminar Paging the family $\mathcal{S}$ is assumed to be a laminar family of slot sets whose height we denote by h. Theorem 5.1 bounds the optimal ratios by $3h^2k$ (deterministic), $3h^2H_k$ (randomized) and $3h^2$ (offline polynomial-time approximation). The proof of Theorem 5.1 (Section 5.1) is by a reduction of Slot-Laminar Paging to Page-Laminar Paging, studied in Section 4. Theorem 5.2, presented in Section 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 online $3h^2k$-competitive algorithm, a randomized online $3h^2H_k$-competitive algorithm, and an offline $3h^2$-approximation algorithm.*

Our focus here is on uniform treatment of the three variants of Slot-Laminar Paging in the above theorem. The ratios in this theorem have not been optimized. For example, in Section 5.2 we give a better deterministic algorithm. For the special case when $h = 2$ the problem can be reduced to All-or-One Paging, for which the ratio can be improved even further [23].

The proof of Theorem 5.1 is by a reduction of Slot-Laminar Paging to Page-Laminar Paging, presented in Lemma 5.1 below. The reduction uses a relaxation of Slot-Laminar Paging that relaxes the constraint that each slot hold at most one page (but still enforces the cache-capacity constraint), yielding an instance of Page-Laminar Paging. The reduction simulates the given Page-Laminar Paging algorithm on multiple

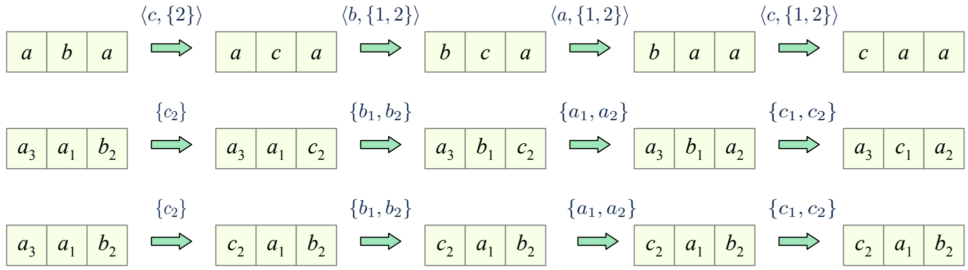

Figure 5: At the top, an instance of Slot-Laminar Paging and its solution C. In the middle row, the corresponding solution C' of the relaxed instance. The bottom row shows a cheaper solution of the relaxed instance.

instances of Page-Laminar Paging— one for each 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 global 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 $v(p, s)$ corresponding to page/slot pairs (p, s). This virtual page can be placed in any slot, although it represents page p being in slot s.

Formally, this relaxation is defined as follows. Fix any k-slot Slot-Laminar 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). 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 $\mathcal{P} = \big\{V(p, S) : p \text{ is any page and } S \in \mathcal{S}\big\}$. Crucially, if $\mathcal{S}$ is slot-laminar with height h, then $\mathcal{P}$ is page-laminar with the same height h.

Instance π is a relaxation of σ in the sense that for any solution C for σ there is a solution C' for π with $\mathsf{cost}(C') \le \mathsf{cost}(C)$. Namely, this C' mimics C by keeping in its cache the virtual pages $v(p, s)$ such that C has page p cached in slot s. It follows that $\mathsf{opt}(\pi) \le \mathsf{opt}(\sigma)$.

Example. We illustrate these concepts with a simple example for a cache of size $k = 3$ (see Figure 5). Let the laminar family be $\mathcal{S} = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 2, 3\}\}$. Suppose that the initial cache has page a in slots 1 and 3 and page b in slot 2. To simplify notation for virtual pages, in this example we write p_s instead of $v(p, s)$. Suppose also that in the corresponding relaxed instance the cache has initially virtual pages a_1, b_2, a_3 (the slot assignment here is not significant). Now, consider the request sequence, $\sigma = (\langle c, \{2\}\rangle, \langle b, \{1, 2\}\rangle, \langle a, \{1, 2\}\rangle, \langle c, \{1, 2\}\rangle)$, and its solution C shown in the top row in Figure 5. This solution has cost 3. The sequence π that is the relaxation of σ is $\pi = (\{c_2\}, \{b_1, b_2\}, \{a_1, a_2\}, \{c_1, c_2\})$. The second row shows the solution C' for π corresponding to C. Its cost is also 3. The bottom row shows a solution for π whose cost is only 1.

For the rest of the proof, we 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.

Next we define the algorithm $\mathbb{B}$. Fix an $f_h(k)$-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 $\mathbb{B}$. For ease of presentation we will focus on the online case: we assume that Algorithm $\mathbb{A}$ is an online algorithm, and we convert it into Algorithm $\mathbb{B}$ that is also online. If $\mathbb{A}$ is offline, the reduction we give below can be naturally modified to produce $\mathbb{B}$ as an offline algorithm instead.

The description of $\mathbb{B}$, below, is top-down: we start with an overview that explains the fundamental strategy. We then formulate an invariant that $\mathbb{B}$ needs to maintain to ensure correctness. Next we provide the details, proving that $\mathbb{B}$ indeed satisfies this invariant. We wrap up the proof by bounding $\mathbb{B}$'s cost.

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. Guided by $\mathbb{A}$, for each such S, Algorithm $\mathbb{B}$ will internally build its own solution, denoted $\mathbb{B}(\sigma_S)$, for σ_S, also using its own designated cache of size $|S|$ composed of copies of the slots in S. These solutions are not independent—the choices made for some S may affect the solution constructed for its ancestors. (This will be captured by Invariant (I) given below.) We stress that all the above actions are internal to $\mathbb{B}$. The actual output produced by $\mathbb{B}$ on input σ for a cache of size k is $\mathbb{B}(\sigma_R)$. (Recall that $\sigma_R = \sigma$.) All other solutions σ_S are used by $\mathbb{B}$, roughly, only as a way to represent information about the past.

For internal bookkeeping purposes, in presenting Algorithm $\mathbb{B}$, for each page p, we let the virtual pages $v(p, s)$ (for every slot s, as defined for Page-Laminar Paging) represent copies of page p. 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 it's in) by a copy of page p. So, when a virtual page $v(p, s)$ is in a slot s' (with, possibly, $s' \neq s$) this satisfies any request $\langle p, S' \rangle$ with $s' \in S'$. We then overcount the cost by considering 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 $s' \neq s$) in the same slot, we still charge 1 to the cost of $\mathbb{B}$. We will upper bound $\mathbb{B}$'s cost overcounted in this way.

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

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

(I1) the solution $\mathbb{B}(\sigma_S)$ caches $v(p, s)$ in some slot in S, and

(I2) if S has a child c with $s \in c$, and $\mathbb{B}(\sigma_c)$ has $v(p, s)$ in its cache c, then in $\mathbb{B}(\sigma_S)$ copy $v(p, s)$ is in the same slot as in $\mathbb{B}(\sigma_c)$.

As explained in Section 2, by a child of S in Condition (I2) we mean a child of S in the forest representation of the laminar family $\mathcal{S}$.

The invariant above 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 (I1), 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 (I2) 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.

To maintain Invariant (I) $\mathbb{B}$ 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 $v(p, s)$. After this eviction both Conditions (I1) and (I2) will be preserved. 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 (I2) 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 $s_1 \neq s'$; 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 $s_2 \neq s'$. In the case that the retrieval does create two

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

Figure 6: "Rotating" slots in $\mathbb{B}(\sigma_S)$ and ancestors to preserve the invariant. Pages in grey are not moved.

violations (and $s_1 \neq s_2$), $\mathbb{B}$ 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 6. 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. Thus this rotation indeed restores the invariant, at the expense of three retrievals at the root. (The retrievals at other nodes only modify the internal state of $\mathbb{B}$.) There are three other cases: two violations with $s_1 = s_2$, one violation at $\mathbb{B}(\sigma_S)$, or one violation at its parent, but all these three cases can be handled similarly, also with at most three retrievals (in fact at most two) at the root.

Total cost. Each retrieval by $\mathbb{A}(\pi_S)$ causes at most 3 retrievals in $\mathbb{B}(\sigma_R)$, so $\mathsf{cost}(\mathbb{B}(\sigma_R))$ is at most

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

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 $\mathsf{opt}(\pi_S) \leq \mathsf{opt}(\sigma_S)$, and that $|S| \leq k$ so $f_h(|S|) \leq f_h(k)$.[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 a $3hf_h(k)$-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 satisfy 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. 7) has competitive ratio at most* $2 \cdot \mathsf{mass}(\mathcal{S}) - k \leq (2h-1)k$.

We begin by defining the terminology used in the algorithm and the proof, and establish some useful properties. Recall that a configuration D satisfies a request $r = \langle p, S\rangle$ if there exists a slot s in S such that s holds p in D; in this case, we also say that slot s satisfies r in D. A configuration D is said to *satisfy a set* R of requests if it satisfies every request in R. A set R of requests will be called *satisfiable* if there exists a configuration that satisfies R. To determine if a set R of requests is satisfied by a configuration, it

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

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

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

1.1. if $t = 1$ or $R_{t-1} \cup \{\sigma_t\}$ is not satisfiable: let $R_{t-1} = \emptyset$ and empty the cache — *start new phase*

1.2. let $R_t = R_{t-1} \cup \{\sigma_t\}$

1.3. if C_{t-1} satisfies $\sigma_t = \langle p, S \rangle$: let $C_t = C_{t-1}$ — *redundant request*

1.4. else: — *non-redundant request*

1.4.1. find sequences $\langle s_1, \ldots, s_m \rangle$, $\langle S_0 = S, S_1, \ldots, S_{m-1} \rangle$, and $\langle p_0 = p, p_1, \ldots, p_{m-1} \rangle$ s.t.
(i) $S_{i-1} \subsetneq S_i$ and slot $s_i \in S_{i-1}$ of C_{t-1} satisfies $\langle p_i, S_i \rangle \in \mathsf{rep}(R_{t-1})$, for $1 \le i < m$, and
(ii) slot $s_m \in S_{m-1}$ of C_{t-1} either
(ii.1) does not satisfy any requests in $\mathsf{rep}(R)$, or
(ii.2) satisfies a request $\langle p, S' \rangle \in \mathsf{rep}(R_{t-1})$ such that $S' \supsetneq S_{m-2}$

1.4.2. to obtain C_t and satisfy $\langle p_{i-1}, S_{i-1} \rangle$, place p_{i-1} in slot s_i, for $1 \le i \le m$

Figure 7: Deterministic online Slot-Laminar Paging algorithm REFSEARCH. Note that in Step 1.4.1 we have $m \le k + 1 - |S|$, and that in (ii), if s_m satisfies $\langle p, S' \rangle \in \mathsf{rep}(R_{t-1})$ then $m \ge 2$ (because C_{t-1} does not satisfy σ_t); thus S_{m-2} is well-defined.

is sufficient (and necessary) to examine the maximal subset of "deepest" requests in the laminar hierarchy. 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, $\mathsf{rep}(R) = \{\langle p, S \rangle \in R : \forall S' \subsetneq S, \langle p, S' \rangle \notin R\}$. For $r = \langle p, S \rangle$, define $\mathsf{anc}(r, R) = \{\langle p, S' \rangle \in R : S \subseteq S'\}$. Lemma 5.2 establishes some basic properties of $\mathsf{rep}(R)$.

Lemma 5.2. *Let R be a set of requests. Then,*
(i) In any configuration, each slot can satisfy at most one request in $\mathsf{rep}(R)$.
(ii) A configuration satisfies R if and only if it satisfies $\mathsf{rep}(R)$.
(iii) R is satisfiable iff for any requestable set S, $\mathsf{rep}(R)$ *has at most $|S|$ requests to subsets of S.*

Proof. (i) This part holds because any two requests in $\mathsf{rep}(R)$ request either different pages or disjoint slot sets. (ii) Since $\mathsf{rep}(R) \subseteq R$, if R is satisfiable, so is $\mathsf{rep}(R)$. On the other hand, if a configuration D satisfies $\mathsf{rep}(R)$ then D satisfies R, because every r in R is an ancestor of some r' in $\mathsf{rep}(R)$ and can be satisfied by the slot satisfying r'.

(iii) Suppose that R is satisfiable. If D is a configuration that satisfies R then it also satisfies $\mathsf{rep}(R)$, by (ii). By (i), for any requestable set S, all requests in $\mathsf{rep}(R)$ to subsets of S must be satisfied in D by different slots of S, so there can be at most $|S|$ such requests. To prove the reverse implication, assume that for any requestable set S there are at most $|S|$ requests in $\mathsf{rep}(R)$ to subsets of S. We construct D top-down. Let T be the root of the laminar hierarchy $\mathcal{S}$. (We could assume that $T = [k]$, but it's not necessary.) By our assumption, there are at most $|T|$ requests in $\mathsf{rep}(R)$. The children of T in $\mathcal{S}$ are disjoint, so we can distribute these requests to the children of T in such a way that each child Q is assigned at most $|Q|$ requests from $\mathsf{rep}(R)$, and each request assigned to Q is to a subset of Q. Continuing this recursively down the tree, we will end up with requests assigned to leaves. Then, for any leaf L we can satisfy its assigned requests by different slots in L. □

Algorithm REFSEARCH is given in Figure 7. It consists of phases. The first phase starts in time step 1, and each phase ends when adding the current request to the request set from this phase makes it unsatisfiable. Within a phase, redundant requests, that is those satisfied by the current configuration, are ignored (Step 1.3). To serve a non-redundant request $\sigma_t = \langle p, S \rangle$, the cache content is rearranged to free a slot in S. This rearrangement involves shifting the content of some slots that serve requests in $\mathsf{rep}(R)$ along the path from S to the root, to find a slot that is either unused or holds p (Step 1.4.2).

For technical reasons, in the analysis of Algorithm REFSEARCH it will be useful to introduce a slightly refined concept of configurations. Given a request set R, an *R-configuration* is a configuration D in which each request in $\mathsf{rep}(R)$ is served by exactly one slot. (By Lemma 5.2(i), each slot can serve only one request in $\mathsf{rep}(R)$, but in general in a configuration serving R there may be multiple slots that serve the same request in $\mathsf{rep}(R)$.) Slots in D that do not serve requests in $\mathsf{rep}(R)$ are called *free in D*. Observe that each configuration C_t of Algorithm REFSEARCH implicitly is an R_t-configuration – due to the assignment of slots in Step 1.4.2. Also, if the slot s_m chosen by the algorithm in Step 1.4.1 satisfies condition (ii.1) then s_m is a free slot of D, according to our definition.

The following helper claim, which characterizes when a particular request is not satisfied by a given configuration, follows directly from Lemma 5.2(iii).

Claim 5.3. *Let R be a set of requests and D be an R-configuration. Let also $r = \langle p, S\rangle$ be a request such that D does not satisfy r, yet $R \cup \{r\}$ is satisfiable. Then D has a slot s in S that is either free or satisfies a request $\langle p', S'\rangle \in \mathsf{rep}(R)$ where $S \subsetneq S'$.*

The following lemma establishes the validity of Steps 1.4.1 and 1.4.2 of Algorithm REFSEARCH.

Lemma 5.4. *Let R be a set of requests and D be an R-configuration. Let $r = \langle p_0, S_0\rangle$ be a request such that r is not satisfied by D and $R \cup \{r\}$ is satisfiable. Then there exist sequences $\langle s_1, \ldots, s_m\rangle$, $\langle S_0, S_1, \ldots, S_{m-1}\rangle$, and $\langle p_0, p_1, \ldots, p_{m-1}\rangle$ such that (i) $S_{i-1} \subsetneq S_i$ and $s_i \in S_{i-1}$ is currently satisfying request $\langle p_i, S_i\rangle \in \mathsf{rep}(R)$, for $1 \le i < m$, and (ii) $s_m \in S_{m-1}$ is either a free slot or is currently satisfying $\langle p_0, S'\rangle \in \mathsf{rep}(R)$ for some $S' \supsetneq S_{m-2}$. Furthermore, transforming D by moving page p_{i-1} to slot s_i (and modifying the slot assignment in D accordingly), for $1 \le i \le m$, yields an $(R \cup \{r\})$-configuration.*

Proof. The proof is by induction on the depth of S_0 in the laminar hierarchy. For the induction base, consider $S_0 = [k]$. Since r is not satisfied by D, $R \cup \{r\}$ is satisfiable, and every requestable slot set is subset of $[k]$, we obtain from Claim 5.3 that 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). Since s_1 is free, bringing page p_0 to slot s_1 yields a $(R \cup \{r\})$-configuration.

We now establish the induction step. Let R, D, and $r = \langle p_0, S_0\rangle$ be as given. By Claim 5.3 there are two cases. In the first case, there is a free slot $s_1 \in S_0$ in D. Then the desired claim holds with $m = 1$, and sequences $\langle s_1\rangle$, $\langle S_0\rangle$ and $\langle p_0\rangle$. Furthermore, as in the base case, since s_1 is free, bringing page p_0 to slot s_1 yields an $(R \cup \{r\})$-configuration.

The remainder of this proof concerns the second case, in which there is a slot $s_1 \in S_0$ currently satisfying a request $r' = (p_1, S_1)$ in $\mathsf{rep}(R)$ with $S_0 \subsetneq S_1$. Let D' denote the configuration that is identical to D except that D has p_0 in slot s_1. Since D is an R-configuration, no other slot satisfies r' in D; the same holds in D'. Hence, D' does not satisfy r'. Furthermore, D' satisfies every request in $\mathsf{rep}(R)$ other than r'. Let $R' = R \cup \{r\} \setminus \mathsf{anc}(r', R)$. In D', s_1 satisfies r. Consider any request x in $R \setminus \mathsf{anc}(r', R)$. By definition of $\mathsf{rep}(R)$, there exists a request x' in $\mathsf{rep}(R)$ that is a descendant of x. Since R' does not include any ancestors of r', x' is not r' and hence is satisfied by some slot in D'. We thus obtain that D' satisfies R' and, in fact D' is an R'-configuration. In D' slot s_1 is assigned to r, and if there is a request (p, S') in $\mathsf{rep}(R)$ then its assigned slot is designated as free in D'. At the same time, D' does not satisfy r'. Further, since $R' \cup \{r'\}$ is a subset of $R \cup \{r\}$, which is satisfiable, $R' \cup \{r'\}$ is also satisfiable. Since $S_1 \supsetneq S_0$, 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 $\langle p_1, p_2, \ldots, p_{m-1}\rangle$ such that (i) $S_{i-1} \subsetneq S_i$ and $s_i \in S_{i-1}$ is currently satisfying $(p_i, S_i) \in \mathsf{rep}(R')$, for $2 \le i < m$; and either (ii.1) s_m is a free slot in D' or (ii.2) is currently satisfying a request $(p_1, S') \in \mathsf{rep}(R')$ for some $S' \supsetneq S_1$. Note, however, that s_m has to be a free slot in D' since (ii.2) above cannot hold: any request (p_1, S') is in $\mathsf{anc}(r', R)$, all requests of which are excluded from R'. Furthermore, transforming D' to D'' by moving page p_{i-1} to s_i for $2 \le i \le m$, satisfies $R' \cup \{r'\}$.

We now establish the desired claim for D, 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 condition (i) follows from (i) of the induction hypothesis above and the fact that in D, $s_1 \in S_0$ is currently satisfying a request (p_1, S_1) in $\mathsf{rep}(R)$ with $S_0 \subsetneq S_1$. For (ii), note that since s_m is a free slot in D', either s_m is a free slot in D or (p_0, S') is in $\mathsf{rep}(R)$ for some $S' \supsetneq S_{m-2}$, thus establishing (ii). Finally, transforming D to D'' by moving p_{i-1} to s_i for $1 \le i \le m$, satisfies $R' \cup \{r'\}$. Since any request satisfying r' also satisfies all ancestors of r', we have $\mathsf{rep}(R \cup \{r\}) = \mathsf{rep}(R' \cup \{r'\})$, implying that D'' also satisfies $R \cup \{r\}$. This completes the induction step and the proof of the lemma. □

Proof of Theorem 5.2. We first argue that at any time t, configuration C_t of RefSearch satisfies the set R_t of requests from the current phase of the algorithm. The proof is by induction on the number of steps within a phase. When the phase is about to start at time t then R_{t-1} is set to $\emptyset$, so the claim holds. For the induction step, consider a step t within a phase and assume that C_{t-1} satisfies R_{t-1}. If C_{t-1} satisfies new request σ_t, then by Step 3.3, C_t satisfies R_t. Otherwise, $R_{t-1} \cup \{\sigma_t\}$ is satisfiable but C_{t-1} does not satisfy σ_t. Then, by Lemma 5.4, Steps 1.4.1 and 1.4.2 derive a configuration C_t satisfying R_t, completing the induction step and the argument that at any time t, C_t satisfies R_t.

We next analyze the competitive ratio. We first show that the number of page retrievals during a phase of RefSearch is at most $2 \cdot \mathsf{mass}(\mathcal{S})$. Let R denote the set of requests in the current phase. We charge the cost in this phase to the depths of the requests in $\mathsf{rep}(R)$. The cost of Step 1.4.2 is m. If s_m satisfies condition (ii.1), then $\mathsf{rep}(R \cup \{\sigma_t\}) = \mathsf{rep}(R) \cup \{\sigma_t\}$ and the depth of S is at least m, so the charge per unit depth is at most 1. Otherwise, condition (ii.2) holds and $\mathsf{rep}(R \cup \{\sigma_t\}) = \mathsf{rep}(R) \cup \{\sigma_t\} \setminus \{\langle p, S' \rangle\}$. In this case we have σ_t inherit the charges to $\langle p, S' \rangle$, and we charge the cost of m to the difference in depths of S and S', which is at least $m-1$ (because $S_{m-2} \subsetneq S'$), so the charge per unit of depth is at most $m/(m-1) \le 2$. (Note that in this case $m \ge 2$.) When the phase ends, a request at depth d was charged at most d times, and these charges include at least one unit charge, so its total charge is most $2d-1$. Thus the algorithm's cost per phase is at most $2 \cdot \mathsf{mass}(\mathcal{S}) - k \le (2h-1)k$. The optimal cost in a phase is at least 1 as no configuration satisfies all requests in the phase and the request that starts the next phase. The theorem follows. □

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, Section 6.1 proves that the optimal randomized ratio is at least $2H_k - O(1)$. (Compare it to the upper bound of $12H_k$, that follows from Theorem 5.1 for $h = 2$.) Then, Section 6.2 shows that the offline problem is $\mathbb{NP}$-hard.

6.1 Lower bound for randomized All-or-One Paging

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

Proof. We establish our lower bound by giving a probability distribution on the input sequences for which any deterministic algorithm $\mathbb{A}$ has expected cost at least $2H_k - 1$ times the optimum cost. Without loss of generality we can assume that $\mathbb{A}$ is *lazy*, in the sense that it retrieves a page only when it is necessary to satisfy a request.

We present the proof in terms of a game between algorithm $\mathbb{A}$ and an adversary who generates the request sequence and its solution. It is then sufficient to show that the expected cost of $\mathbb{A}$ is at least $2H_k - 1$ times the adversary's cost. We use some fixed $k+1$ pages $p_1, p_2, \ldots, p_{k+1}$ and the random input sequence generated by the adversary will consist of L phases, where L is an integer that can be made arbitrarily large.

Consider any phase. To ease notation, by symmetry, assume without loss of generality that when the phase starts the adversary has pages $p_1, p_2, \ldots, p_k$ in the cache, with each page p_i in slot i, for $i = 1, 2, \ldots, k$. To start the phase, the adversary chooses a random permutation $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ of these k pages, replaces p_{i_k} in its cache by p_{k+1}, at cost 1, then makes request $\langle p_{k+1}, * \rangle$, followed by $k-1$ stages. Each stage $s = 1, 2, \ldots, k-1$ consists of $L \cdot (2H_k - 1)$ repetitions of the request sequence

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

which costs the adversary nothing.

It remains to bound the expected cost of $\mathbb{A}$. Let $\mathcal{E}$ denote the event that for every phase and every stage s in the phase, the configuration of $\mathbb{A}$ at the end of the stage has each page p_{i_r}, for $r = 1, 2, \ldots, s$, in slot i_r and one of the slots in $[k] \setminus \{i_1, \ldots, i_s\}$ contains p_{k+1}. We will separately bound the expected cost of $\mathbb{A}$ conditioned on $\mathcal{E}$, and the expected cost of $\mathbb{A}$ conditioned on $\overline{\mathcal{E}}$ (the complement of $\mathcal{E}$).

We first analyze the expected cost of $\mathbb{A}$ conditioned on $\mathcal{E}$. Observe that when a phase starts $\mathbb{A}$ and the adversary are in the same configuration. For the first phase this holds because both $\mathbb{A}$ and the adversary are in the initial configuration. For any other phase it follows from condition $\mathcal{E}$ applied to stage $k-1$ of the previous phase.

Now, consider a stage s of a phase. At the beginning of this phase the configuration of $\mathbb{A}$ has each page p_{i_r}, for $r = 1, 2, \ldots, s-1$, in slot i_r, and one of the slots in $[k] \setminus \{i_1, \ldots, i_{s-1}\}$ contains p_{k+1}. Indeed, for $s \geq 2$ this follows directly from condition $\mathcal{E}$ applied to stage $s-1$. For $s = 1$ this follows from $\mathbb{A}$ and the adversary being in the same configuration when the phase starts, and from the adversary making the request $\langle p_{k+1}, * \rangle$ right before stage 1.

Since the probability distribution of i_s is uniform in $[k] \setminus \{i_1, i_2, \ldots, i_{s-1}\}$, the probability that $\mathbb{A}$ has p_{k+1} in slot i_s equals $1/(k-s+1)$. If it does, the cost of $\mathbb{A}$ is at least 2 in stage s, because p_{i_s} will need to be fetched into slot i_s and p_{k+1} will need to be moved to a different slot (to preserve condition $\mathcal{E}$). So the expected cost of $\mathbb{A}$ in this stage is at least $2/(k-s+1)$. Summing over all stages $s = 1, 2, \ldots, k-1$ and adding 1 for the first request, the expected cost of $\mathbb{A}$ for a phase, conditioned on event $\mathcal{E}$, will be at least $2(H_k - 1) + 1 = 2H_k - 1$. Therefore, the expected total cost of $\mathbb{A}$ over L phases, conditioned on event $\mathcal{E}$, is at least $L \cdot (2(H_k - 1) + 1) = L \cdot (2H_k - 1)$.

We next analyze the expected cost of $\mathbb{A}$ conditioned on $\overline{\mathcal{E}}$. The event $\overline{\mathcal{E}}$ implies that there is a stage s of a phase in which $\mathbb{A}$ does not end with a configuration in which each page p_{i_r}, for $r = 1, 2, \ldots, s$, is in slot i_r, and page p_{k+1} is in one of the slots in $[k] \setminus \{i_1, \ldots, i_s\}$. Since such a configuration satisfies all requests in the stage and $\mathbb{A}$ is lazy, this implies that $\mathbb{A}$ never reaches such a configuration in the stage. (Otherwise $\mathbb{A}$ would have stayed in this configuration through the rest of the phase.) There are no other configurations that satisfy all requests in this phase at no cost. Therefore, $\mathbb{A}$ incurs a cost of at least one during each of the $L \cdot (2H_k - 1)$ repetitions of the request sequence, yielding a total cost in this stage alone of at least $L \cdot (2H_k - 1)$.

Since the adversary pays 1 for each phase, the total cost of the adversary is L. We showed that, whether we condition on $\mathcal{E}$ or $\overline{\mathcal{E}}$, the expected cost of $\mathbb{A}$ is at least $L \cdot (2H_k - 1)$, so this will be true also without any conditioning. Therefore, the competitive ratio of $\mathbb{A}$ is at least $2H_k - 1$. □

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

The off-line version of Paging, where the request sequence is given upfront, can be solved in time $O(n \log n)$ using the classical algorithm by Belady [11]. All-or-One Paging differs from standard Paging only by inclusion of specific requests, which appear easy to handle because they don't give the algorithm any choice. In this section we show that this intuition is not valid:

Theorem 6.2. *Offline All-or-One Paging is* $\mathbb{NP}$*-complete.*

Proof of Theorem 6.2. Let $G = (V, E)$ be a graph with vertex set $V = \{0, 1, \ldots, n-1\}$. 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 solution for σ whose cost with a cache size $k+2$ is at most F.

At a fundamental level our proof resembles the argument in [31], where $\mathbb{NP}$-completeness of an interval-packing problem was proved. The basic idea of the 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, its corresponding set of vertices) 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 [31] will be useful:

Lemma 6.1. *Let $X_1, X_2, \ldots, X_p \subseteq V$ be sets of cardinality k such that $X_1 \prec X_2 \prec \ldots \prec X_p$. 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 $b = 0, 1, \ldots, B-1$ and $j = 0, 1, \ldots, n-1$. 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 $x_{b,0}, x_{b,1}, \ldots, x_{b,n-1}$. See Figures 8 and 9 for illustration.

- We have B pages y_b, for $b = 0, 1, \ldots, B-1$. 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 $\theta_b, \theta'_b \in [\tau_{b,j}, \tau'_{b,j}]$, 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 $1, 2, \ldots, k$ 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.

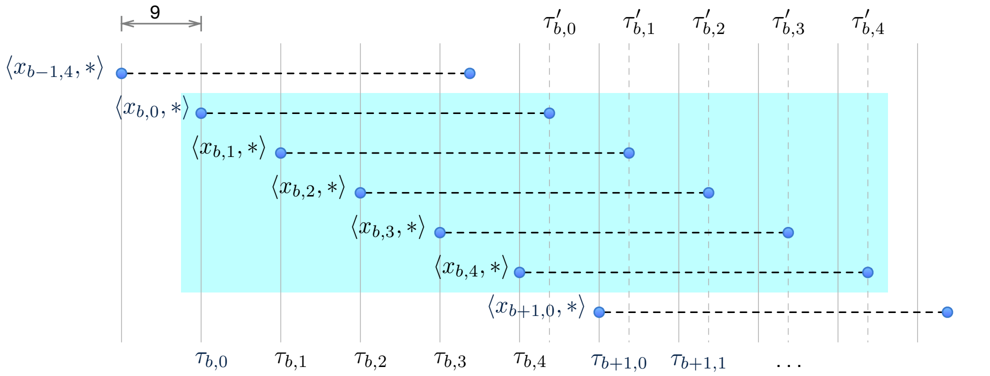

Figure 8: The sequence of vertex requests, for $n = 5$. The shaded region contains requests from bundle b.

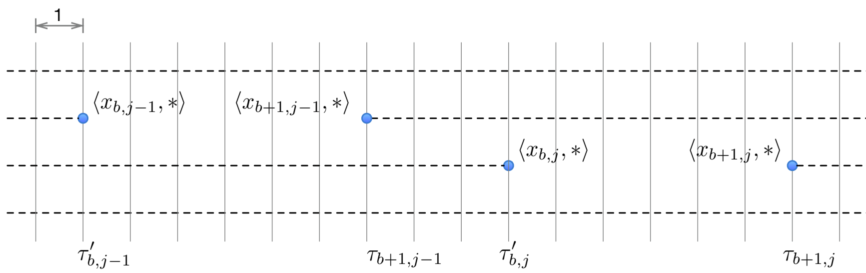

Figure 9: A more detailed picture showing relations between general requests to pages $x_{b,j-1}$, $x_{b,j}$, $x_{b+1,j-1}$, and $x_{b+1,j}$, where $1 \le j \le n-1$.

Let $F' = (2n - k + 2)B$. For any solution S of σ' and any bundle b, denote by $V_{S,b}$ the set of vertices $j \in V$ for which S does not fault in σ' 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 solution for σ' with at most F' faults, then for any $b = 0, 1, \ldots, B-2$ we have $V_{S,b} \preceq V_{S,b+1}$.*

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 does not fault 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 avoid faulting on at most k 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 two 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 solution 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 $m-1$ 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 $\boldsymbol{\omega}_{p,e}$ uses six new pages $z_{p,u}$, $z_{p,v}$, $g_{p,u}$, $g_{p,v}$, $h_{p,u}$ and $h_{p,v}$, and consists 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.

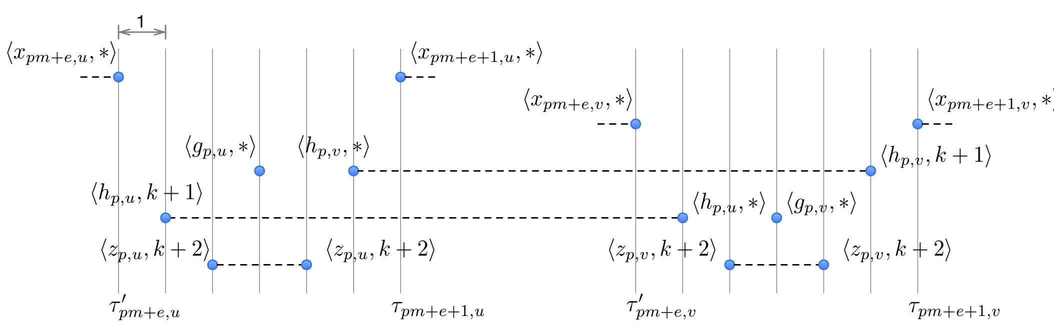

Figure 10: Gadget $\boldsymbol{\omega}_{p,e}$. Requests $\langle x_{pm+e,u}, *\rangle$, $\langle x_{pm+e+1,u}, *\rangle$, $\langle x_{pm+e,v}, *\rangle$, and $\langle x_{pm+e+1,v}, *\rangle$ are not part of this gadget; they are shown only to illustrate how gadget $\boldsymbol{\omega}_{p,e}$ fits into the overall request sequence.

Consider now possible solutions of gadget $\boldsymbol{\omega}_{p,e}$. Notice that this gadget will require 7 faults regardless of all other requests, 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 do not fault 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 solution that puts $\langle g_{p,u}, *\rangle$ in a vertex slot is called a u-*solution* of $\boldsymbol{\omega}_{p,e}$ and a solution that puts $\langle g_{p,v}, *\rangle$ in a vertex slot is called a v-*solution* of $\boldsymbol{\omega}_{p,e}$. (A solution of $\boldsymbol{\omega}_{p,e}$ can be both a u-solution and a v-solution.)

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 solution 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 construct a solution for 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-solution 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-solution 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 solution 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 $\langle x_{b,j}, *\rangle$, 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. Solution S makes 7 faults on gadget $\boldsymbol{\omega}_{p,e}$, so for this gadget it must be either a u-solution or a v-solution. If it is a u-solution 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-solution 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 completes the proof of the theorem. □

7 Weighted All-Or-One Paging

Section 1 introduced the generalization of the standard k-Server problem called Heterogenous k-Server, where each request, in addition to the request point, specifies also a subset of servers that are allowed to serve the request. The previous section focussed on the special case of Heterogenous k-Server in uniform metrics. Extending this work beyond uniform metrics, this section addresses Weighted All-Or-One Paging, the natural weighted variant 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 requestable-set family $\mathcal{S} = \{[k]\} \cup \{\{s\} : s \in [k]\}$. This section proves the following theorem:

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

The bound is optimal up to a constant factor, as the optimal ratio for standard Weighted Paging is k. Figure 11 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 $\mathbb{NP}$-hard special case of

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 that $\sigma_i = \langle \zeta, i \rangle$ for $i \in [k]$ — *k specific requests to artificial weight-0 page ζ*
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 $\langle p_t, s_t \rangle = \langle p_t, * \rangle$ is a general request not satisfied by any cached request t' (s.t. $p_{t'} = p_t$):

3.2.1. define
$$\begin{cases} \ell_t(s) := \max\{t' \le t : s_{t'} = s\} \text{ for } s \in [k] & \text{— most recent specific request to slot } s \\ A := \{s \in [k] : \mathsf{cap}[\ell_t(s)] \ge \tfrac{1}{2}\,\mathsf{wt}(p_t) \textit{ and s does not hold a specific request}\} \\ B := \{s \in [k] : \textit{slot s holds a general request of weight at least } \tfrac{1}{2}\,\mathsf{wt}(p_t)\} \end{cases}$$

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[a]

[a] And elsewhere (e.g. the definition of $\ell_t(s)$ in Step 3.2.1) restrict to previous times t' that were not ignored in this way.

Figure 11: An $O(k)$-competitive online algorithm for Weighted All-Or-One Paging. Following our convention, we present the algorithm as caching request times rather than pages, with the understanding that request t actually represents page p_t.

Multicommodity Flow. As a small example that illustrates the challenge, consider a cache of size $k = 2$, and repeatedly make three requests: a general request to a weight-1 page, and specific requests to different weight-zero pages in slots 1 and 2. The weight-zero requests force the weight-1 page to be evicted with each round, so the optimal cost is the number of rounds. But the solution of the classical linear-program relaxation will have value 1. Thus this linear program cannot be used to bound the competitive ratio.

Let $\sigma = (\sigma_t = \langle p_t, s_t \rangle)_{t=1}^T$ be the request sequence. For convenience, in the proof we identify requests with the time when they are issued; that is, by "request t" we mean request σ_t. Also, our algorithm needs to keep track not only of the cache content, but also of the times when each page in the cache was retrieved. For this reasons, it's convenient to think about the algorithm and the optimal solution as caching *requests* or *request times* rather than pages, with the understanding that "request t" in the cache actually represents page p_t. We adopt this convention throughout the proof.

Here is a sketch of the proof of Theorem 7.1, then the detailed proof. Fix an optimal solution C, that is $\mathsf{opt}(\sigma) = \mathsf{cost}(C)$. For each $t \in [T]$, let $x_t \in \{0, 1\}$ be an indicator variable for the event that C evicts request t before satisfying another request $t' > t$ with the same page/slot pair that satisfied t. Let $R \subseteq [T]$ be the set of all specific requests, and for each $t \in R$, let y_t be the amount C 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 $2\,\mathsf{opt}(\sigma)$. One can show (Lemma 7.1) that 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 $t' = \ell_t(s) \in R$. It follows that the residual cost is decreasing at least at unit rate in Step 3.2.2.1.

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 $2k$. So, the final value of ϕ is at most $4k\,\mathsf{opt}(\sigma)$. To finish, we show by a charging argument that the algorithm's cost is at most $6\,\phi + 3\,\mathsf{opt}(\sigma) \le (24k+3)\,\mathsf{opt}(\sigma)$.

Here is the detailed proof. Consider any execution of the algorithm on a k-slot instance σ. To ease notation and streamline the analysis, without loss of generality we make the following assumptions:

- The first k requests are specific requests for an artificial weight-zero page ζ in each of the k slots.
- Each request is not redundant (per Step 3.3).
- The last k requests are specific requests for an artificial weight-zero page ζ in each of the k slots.

These assumptions can indeed be made without loss of generality, as the zero-weight requests do not have any cost, the algorithm ignores redundant requests, and removing redundant requests doesn't increase the optimum cost.

We first prove a key lemma used in the proof of the theorem.

Lemma 7.1. *Suppose that, while responding to a general request t, the algorithm is executing Step 3.2.2.1 (that is, the loop condition in Step 3.2.2 is satisfied). Then, just after C has responded to request t, either*

(i) some request t' currently cached by the algorithm is not in C's cache, or

(ii) for some slot $s \in [k]$, after the most recent specific request $\ell_t(s)$ to slot s solution C has incurred cost more than $\mathsf{cap}[\ell_t(s)]$ for retrievals into s.

Proof. If C satisfies property (i), we are done. So assume that it doesn't. We will show that then property (ii) holds. If (i) doesn't hold then, just after responding to request t, in addition to the current general request p_t, solution C caches every request t' that is cached by the algorithm. This, together with the loop condition, implies that C has at least $|B| + 1 \ge |A| + 1$ generally requested pages of weight at least $\frac{1}{2}\,\mathsf{wt}(p_t)$ in its cache. Thus one of these pages, say $p_{t'}$, is in a slot $s \notin A$. The choice of $p_{t'}$ and the definition of A imply then that the cost of C for retrievals into s after time $\ell_t(s)$ is at least $\mathsf{wt}(p_{t'}) \ge \frac{1}{2}\,\mathsf{wt}(p_t) > \mathsf{cap}[\ell_t(s)]$, so property (ii) holds. ☐

Proof of Theorem 7.1. Fix an optimal solution C, that is $\mathsf{opt}(\sigma) = \mathsf{cost}(C)$. For each $t \in [T]$, let $x_t \in \{0, 1\}$ be an indicator variable for the event that C evicts request t before satisfying another request $t' > t$ with the same page/slot pair that satisfied t. Let $R \subseteq [T]$ be the set of all specific requests, and for each $t \in R$, let y_t be the amount C 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 $t' = \ell_t(s) \in R$. It follows that the residual cost is decreasing at least at unit rate in Step 3.2.2.1.

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 $2k$. So, the final value of ϕ is at most $4k\,\mathsf{opt}(\sigma)$.

[3]This interpretation of the problem via covering constraints handled via residual costs follows [41]. It can be recast as a linear-programming primal-dual argument, or as (a generalization of) the local-ratio method [41, §5 & §6].

To finish, we show that the algorithm's cost is at most $6\,\phi + 3\,\mathsf{opt}(\sigma) \leq (24k+3)\,\mathsf{opt}(\sigma)$.[4] 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.

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 C 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 C, 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 $2\,\sum_{t' \in R} \mathsf{cap}[t']$.

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 request t. Since the slot holding $p_{t'}$ is not in B, $\mathsf{wt}(p_{t'}) < \frac{1}{2}\,\mathsf{wt}(p_t)$.

Each general request t receives at most one charge in Item 6, from a request t' of at most half the weight of t; this general request t' may also receive such charges, forming a chain of charges, but since the weights of the requests in this chain decrease geometrically, t is charged at most its weight. In Item 5, each specific request t is charged by at most one general request t' of the same weight, that may also carry the chain charge not exceeding its weight. So this specific request is charged at most twice its weight. Overall, the charge of each request from Items 5 and 6 is at most twice its weight.

The total weight of evictions considered in Items 1, 2, 3, and 4 is at most $2\,\phi + \mathsf{opt}(\sigma)$. Adding also the charges to these items by evictions considered in Items 5 and 6, we obtain that the total cost of the algorithm is bounded by $3\,(2\,\phi + \mathsf{opt}(\sigma)) = 6\,\phi + 3\,\mathsf{opt}(\sigma)$. □

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. In particular:

- 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 is the best randomized competitive ratio for Slot-Heterogenous Paging, for arbitrary $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$? Is it possible to achieve ratio that is a poly-logarithmic function of $\mathcal{S}$ and k?

- For Page-Laminar Paging it is easy to show a lower bound of $\Omega(h)$, even for $k = 1$ and for randomized algorithms. But it still may be possible to eliminate or reduce the multiplicative dependence on h. For

[4]This constant can be reduced with more careful analysis.

example, is it possible to achieve ratio $O(h + k)$ with a deterministic algorithm and $O(h + H_k)$ with a randomized algorithm? Similarly, does Slot-Laminar Paging (where $h \leq k$) admit an $O(k)$ deterministic ratio and $O(\log k)$ randomized ratio?

- For deterministic All-or-One Paging, we conjecture that the optimal ratio is $2k - 1$. (For $k = 2$ we can show an upper bound of 3.) In the randomized case, can ratio $2H_k - 1$ be achieved?
- For Weighted All-Or-One Paging, is the optimal randomized ratio $O(\text{polylog}(k))$?
- The status of Heterogenous k-Server in arbitrary metric spaces is wide open. Can ratio dependent only on k be achieved? This question, while challenging, could still be easier to resolve for Heterogenous k-Server than for Generalized k-Server.

Conflict of interest statement. The authors have no relevant financial or non-financial interests to disclose.

References

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000. doi:10.1016/S0304-3975(98)00116-9.

[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. doi:10.1145/3450349.

[3] Nikhil Ayyadevara and Ashish Chiplunkar. The randomized competitive ratio of weighted k-server is at least exponential. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 9:1–9:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.ESA.2021.9, doi:10.4230/LIPICS.ESA.2021.9.

[4] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph (Seffi) Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5), nov 2015. doi:10.1145/2783434.

[5] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012. doi:10.1145/2339123.2339126.

[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. doi:10.1109/FOCS.2017.52.

[7] Nikhil Bansal, Marek Eliás, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized *k*-server in uniform metrics. *ACM Trans. Algorithms*, 19(1):8:1–8:15, 2023. doi:10.1145/3568677.

[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. doi:10.1145/3409964.3461801.

[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. doi:10.1137/1.9781611976021.1.

[10] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966. doi:10.1147/sj.52.0078.

[11] Laszlo A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Syst. J.*, 5(2):78–101, 1966. doi:10.1147/sj.52.0078.

[12] Marcin Bienkowski, Łukasz Jeż, 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. doi:10.4230/LIPIcs.ISAAC.2019.14.

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

[14] Allan Borodin and Ran El-Yaniv. On randomization in on-line computation. *Inf. Comput.*, 150(2):244–267, 1999. doi:10.1006/inco.1998.2775.

[15] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.

[16] Mark Brehob, Richard J. Enbody, Eric Torng, and Stephen Wagner. On-line restricted caching. *J. Sched.*, 6(2):149–166, 2003. doi:10.1023/A:1022989909868.

[17] 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. doi:10.1109/TC.2004.1255792.

[18] Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 581–594. ACM, 2023. doi:10.1145/3564246.3585132.

[19] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Mądry. 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. doi:10.1145/3188745.3188798.

[20] 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.

[21] 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. doi:10.1007/978-3-662-44777-2_18.

[22] Niv Buchbinder, Christian Coester, and Joseph Naor. Online k-taxi via double coverage and time-reverse primal-dual. *Math. Program.*, 197(2):499–527, 2023. URL: https://doi.org/10.1007/s10107-022-01815-6, doi:10.1007/S10107-022-01815-6.

[23] 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. URL: https://arxiv.org/abs/2205.11102.

[24] Ashish Chiplunkar and Sundar Vishwanathan. Metrical service systems with multiple servers. *Algorithmica*, 71(1):219–231, 2015. doi:10.1007/s00453-014-9903-7.

[25] Ashish Chiplunkar and Sundar Vishwanathan. Randomized memoryless algorithms for the weighted and the generalized k-server problems. *ACM Trans. Algorithms*, 16(1), December 2019. doi:10.1145/3365002.

[26] Marek Chrobak, Samuel Haney, Mehraneh Liaee, Debmalya Panigrahi, Rajmohan Rajaraman, Ravi Sundaram, and Neal E. Young. Online paging with heterogeneous cache slots. In *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, 2023.

[27] Marek Chrobak, Samuel Haney, Mehraneh Liaee, Debmalya Panigrahi, Rajmohan Rajaraman, Ravi Sundaram, and Neal E. Young. Online paging with heterogeneous cache slots. *Algorithmica*, 2024. doi:10.1007/s00453-024-01270-z.

[28] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

[29] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

[30] Marek Chrobak and John Noga. Competitive algorithms for relaxed list update and multilevel caching. *J. Algorithms*, 34(2):282–308, 2000. doi:10.1006/jagm.1999.1061.

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

[32] 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. doi:10.1145/3313276.3316370.

[33] 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. doi:10.4230/LIPIcs.ICALP.2021.57.

[34] 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.

[35] 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. doi:10.1007/BFb0054307.

[36] 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. doi:10.1016/0196-6774(91)90041-V.

[37] Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theor. Comput. Sci.*, 130(1):85–99, 1994. doi:10.1016/0304-3975(94)90154-6.

[38] Samuel Haney. *Algorithms for Networks With Uncertainty*. PhD thesis, Duke University, 2019. URL: https://dukespace.lib.duke.edu/dspace/handle/10161/18661.

[39] 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.

[40] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988. doi:10.1007/BF01762111.

[41] Christos Koufogiannakis and Neal E. Young. Greedy δ-approximation algorithm for covering with arbitrary constraints and submodular cost. *Algorithmica*, 66(1):113–152, 2013. doi:10.1007/s00453-012-9629-3.

[42] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

[43] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theor. Comput. Sci.*, 324(2-3):347–359, 2004. doi:10.1016/j.tcs.2004.06.002.

[44] James R. Lee. Erratum:fusible HSTs and the randomized k-server conjecture, 2018. URL: https://homes.cs.washington.edu/~jrl/papers/kserver-erratum.html.

[45] 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.

[46] Xiuhua Li, Xiaofei Wang, Keqiu Li, Zhu Han, and Victor CM Leung. Collaborative multi-tier caching in heterogeneous networks: Modeling, analysis, and design. *IEEE Transactions on Wireless Communications*, 16(10):6926–6939, 2017.

[47] 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. doi:10.1109/HPCA.2016.7446082.

[48] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

[49] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991. doi:10.1007/BF01759073.

[50] M. Mendel and Steven S. Seiden. Online companion caching. *Theoretical Computer Science*, 324(2):183–200, 2004. Online Algorithms: In Memoriam, Steve Seiden. doi:10.1016/j.tcs.2004.05.015.

[51] Jignesh Patel. Restricted k-server problem. Master's thesis, Michigan State University, 2004. URL: https://d.lib.msu.edu/etd/32678.

[52] 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. doi:10.1137/1.9781611975994.92.

[53] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

[54] Aditya Sundarrajan, Mingdong Feng, Mangesh Kasbekar, and Ramesh K Sitaraman. Footprint descriptors: Theory and practice of cache provisioning in a global cdn. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 55–67, 2017.

[55] 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. doi:10.1145/1250662.1250723.

[56] Yaocheng Xiang, Xiaolin Wang, Zihui Huang, Zeyu Wang, Yingwei Luo, and Zhenlin Wang. Dcaps: dynamic cache allocation with partial sharing. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3190508.3190511.

[57] 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. doi:10.1145/2628071.2628104.

[58] 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. doi:10.1007/s10617-015-9168-7.