# Online Paging with Heterogeneous Cache Slots

**Marek Chrobak**
University of California at Riverside, CA, USA

**Samuel Haney**
Tumult Labs, Durham, NC, USA

**Mehraneh Liaee**
Northeastern University, Boston, MA, USA

**Debmalya Panigrahi**
Duke University, Durham, NC, USA

**Rajmohan Rajaraman**
Northeastern University, Boston, MA, USA

**Ravi Sundaram**
Northeastern University, Boston, MA, USA

**Neal E. Young**
University of California at Riverside, CA, USA

## 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 initiate a systematic study of this generalization, we focus on uniform and star metrics. For uniform metrics, the problem is equivalent to a generalization of Paging in which each request specifies not only a page $p$, but also a subset $S$ of cache slots, and is satisfied by having a copy of $p$ in some slot in $S$. We call this problem *Slot-Heterogenous Paging*.

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 ($\mathcal{S} = 2^{[k]}$), 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 that we call *All-or-One Paging* extends standard Paging by allowing each request to specify a specific slot to put the requested page in. For All-or-One Paging the optimal competitive ratios are $\Theta(k)$ (deterministic) and $\Theta(\log k)$ (randomized), while the offline problem is $\mathbb{NP}$-hard. We extend the deterministic upper bound to the *weighted* variant of All-or-One Paging (a generalization of standard Weighted Paging), showing that it is also $\Theta(k)$.

Some 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).

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Caching and paging algorithms, $k$-server, weighted paging, laminar family

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2023.23

**Related Version** *Full Version*: `https://arxiv.org/abs/2206.05579` [23]

**Funding** *Marek Chrobak*: Supported by NSF grants CCF-1536026 and CCF-2153723.
*Samuel Haney*: Supported by NSF grants CCF-1527084 and CCF-1535972.
*Mehraneh Liaee*: Supported by NSF grants CCF-1535929 and CCF-1909363.
*Debmalya Panigrahi*: Supported by NSF grants CCF-1527084, CCF-1535972, CCF-1750140, CCF-1955703, ARO grant W911NF2110230, and Indo-US Joint Center for Algorithms under Uncertainty.
*Rajmohan Rajaraman*: Supported by NSF grants CCF-1535929 and CCF-1909363.
*Ravi Sundaram*: Supported by NSF grants CCF-1535929 and IIS-2039945.
*Neal E. Young*: Supported by NSF 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, nor the fact that modern cache systems often partition the slots, sometimes dynamically, with some parts exclusively accessible by specific processors, cores, processes, threads, or page sets (e.g., [27, 37, 44–46]).

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 [20, 41]. Here, following the strategy taken for other hard generalizations of $k$-Server [6, 7, 11, 21, 28, 36], we initiate a systematic study of this problem by focusing on its restriction to uniform and star metrics. For uniform metrics, the problem is equivalent to a variant of Paging in which each request specifies a page $p$ and a subset $S$ of $k$ 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.) 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, 11].

- 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 1 (i) and 3).

Hence, the optimal ratios for Slot-Heterogenous Paging are exponentially worse than for standard Paging. The proofs of Theorems 1 and 3 employ some novel ideas that may be useful for other problems: the lower bound in Theorems 1 (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 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 $S$. And in realistic settings only certain subsets of cache slots or servers would 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 5). Theorem 1 (ii) shows a complementary lower bound: for infinitely many families $\mathcal{S}$, every deterministic online algorithm has competitive ratio $\Omega(\sqrt{|\mathcal{S}|})$.

Together Theorems 5 and 1 (ii) imply that, as a function of $|\mathcal{S}|$ and $k$, the optimal deterministic ratio for Slot-Heterogenous Paging is polynomial.

**Slot-Laminar Paging (Section 4).** We then consider the specific structure of $\mathcal{S}$, showing better bounds when $\mathcal{S}$ is *laminar*. This case, which we call *Slot-Laminar Paging*, models applications where slot (or server) capabilities are hierarchical. Laminarity implies that $|\mathcal{S}| < 2k$, so (per Theorem 5 above) the optimal deterministic ratio is $O(k^3)$.

- We show that the optimal deterministic and randomized ratios for Slot-Laminar Paging are $O(h^2k)$ and $O(h^2 \log k)$, where $h \leq k$ is the height of $\mathcal{S}$ (Theorem 8). We next tighten the deterministic bound to $O(hk)$ (Theorem 10).

The proof of Theorem 8 is via a reduction to an intermediate problem that we call *Page-Laminar Paging* (introduced below), while the proof of Theorem 10 refines the generic algorithm from Theorem 5. The dependence on $k$ in these bounds is asymptotically tight, as Slot-Laminar Paging generalizes standard Paging.

*Page-Laminar Paging (Section 4.1).* Page-Laminar Paging is a natural generalization of Paging in which each request is a set $P$ of *pages* from an arbitrary but fixed laminar family $\mathcal{P}$, and is satisfiable by fetching any page from $P$ into any slot in the cache.

- 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 6).

The proof is by a reduction, which replaces each set request $P$ by a request to one carefully chosen page in $P$, yielding an instance of Paging, while increasing the optimal cost by at most a factor of $h$.

*Reducing Slot-Laminar Paging to Page-Laminar Paging.* The reduction of Slot-Laminar Paging to Page-Laminar Paging in Theorem 8 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.

**Weighted All-Or-One Paging (Section 5).** *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 [20] has shown that the optimal deterministic ratio is twice that of Paging, to within an additive constant. The optimal randomized ratio of All-or-One Paging is also at least twice that for Paging, as we show in the full paper [23]. Note that Theorem 8 upper bounds the optimal randomized ratio to within a constant factor of that for Paging. The full paper also has a proof that the offline problem is $\mathbb{NP}$-hard [23], in sharp contrast to even $k$-Server, which can be solved in polynomial time for arbitrary metrics.

We initiate a study of Heterogenous $k$-Server in non-uniform metrics through *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 small constant factor (Theorem 14).

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 5 has an example that illustrates the challenge.) We show how to augment the linear program to partially model the slot constraints.

**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 $\mathcal{S}^* = \bigcup_{S\in\mathcal{S}} 2^S$. The lower bound for One-of-$m$ Paging holds for some but not all $m$ and $k$—see Theorem 1(ii). The upper bound for Slot-Laminar Paging in the deterministic case (Theorem 8) 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 [23], as is offline Page-Subset Paging ([21]).

| Paging problem | set family | deterministic | randomized | where |
|---|---|---|---|---|
| Slot-Heterogeneous | $2^{[k]}\setminus\{\emptyset\}$ | $\leq k2^k$ | $\leq O(k^2\log k)$ | via [7, 11] |
| " | arbitrary $\mathcal{S}$ | $\leq k\min(\lvert\mathcal{S}^*\rvert, \mathsf{mass}(\mathcal{S}))$ | | Thm. 5 |
| One-of-$m$, $m\approx k/2$ | $\binom{[k]}{m}$ | $\geq \Omega(2^k/\sqrt{k})$ | $\geq\Omega(k)$ | Thms. 1(i), 3 |
| One-of-$m$, any $m$ | $\binom{[k]}{m}$ | $\gtrsim \Omega((4k/m)^{m/2}/\sqrt{m})$ | | Thm. 1(ii) |
| Slot-Laminar | laminar $\mathcal{S}$, height $h$ | $\leq (2h-1)k$ | $\leq 3h^2H_k$ | Thms. 8, 10 |
| All-or-One | $\{[k]\}\cup$ $\{\{s\}:s\in[k]\}$ | $\geq 2k-1$ | $\geq 2H_k-1$ | [20, 23, 32] |
| " | " | $\leq 2k+14$ | | [20] |
| Weighted All-or-One | $\{[k]\}\cup$ $\{\{s\}:s\in[k]\}$ | $\leq O(k)$ | | Thm. 14 |
| Page-Subset | $\mathcal{P}=\binom{\text{all pages}}{m}$ | $\geq \binom{k+m}{k}-1$ | | [28] |
| " | | $\leq k(\binom{k+m}{k}-1)$ | $\leq O(k^3\log m)$ | [21] |
| Page-Laminar | $\mathcal{P}$ laminar height $h$ | $\leq hk$ | $\leq hH_k$ | Thm. 6 |

**Related work.** Paging and $k$-Server have played a central role in the theory of online computation since their introduction in the 1980s [12, 38, 43]. For $k$-Server, the optimal deterministic ratio is between $k$ and $2k-1$ [35]. Recent work [26] offers hope for closing this gap, and substantial progress towards resolving the randomized case has been reported in [4, 16]. For Weighted Paging the optimal ratios are $k$ and $\Theta(\log k)$ [1, 5, 29, 39, 43].

*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 $O(\log^2 k)$ [18]. 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 [14, 15, 30, 40]. 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 [15] 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 [34], *Multi-Level Caching*—where the cost to access a slot depends on the slot [24], and *Writeback-Aware Caching*—where each page has multiple copies, each with a distinct level and weight, and each request specifies a page and a level, and can be satisfied by fetching a copy of this page at the given or a higher level [8, 9]. (This is a special case of weighted Page-Laminar Paging where $\mathcal{P}$ consists of pairwise-disjoint chains.) *Multi-Core Caching* models the fact that faults can change the request sequence (e.g. [33]).

Patel's master thesis [41] 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 [20]. Earlier work in [32] 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 [36]. For uniform metrics, the optimal competitive ratios for this problem are between $2^k$ and $k2^k$ (deterministic) and between $\Omega(k)$ and $O(k^2 \log k)$ (randomized) [7, 11]. These ratios are exponentially worse than the ratios for standard $k$-Server. Heterogenous $k$-Server, parameterized by $\mathcal{S}$, provides a spectrum of problems that bridges the two extremes.

*Weighted* $k$-Server is a restriction of Generalized $k$-Server in which servers move in the same metric space but have different weights, and the cost is the weighted distance [31]. For this problem (in non-uniform metrics) the deterministic and randomized ratios are at least (respectively) doubly exponential [6, 7] and exponential [3, 22].

Page-Subset Paging, restricted to $m$-element sets of pages, has been studied as (uniform) *Metrical Service Systems with Multiple Servers* [21, 28]. For this problem the deterministic ratio is at least $\binom{k+m}{k} - 1$ [28], while the randomized ratio is $O(k^3 \log m)$ [21]. 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 $P$ [17]. For $k$-Chasing, no online algorithm is competitive even for $d = k = 2$ [17], while for $k = 1$ the ratios grow with $d$ [2, 42].

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 [19, 25].

## 2 Formal Definitions

**Slot-Heterogenous Paging.** A problem instance consists of a set $[k] = \{1, 2, \ldots, k\}$ of cache slots, a family $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$ of requestable slot sets, and a request sequence $\sigma = \{\sigma_t\}_{t=1}^{T}$, where each request has the form $\sigma_t = \langle p_t, S_t \rangle$ for some *page* $p_t$ and set $S_t \in \mathcal{S}$. A *cache configuration* $C$ is a function that specifies the content of each slot $s \in [k]$; this content is either a single page 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 $\sigma$ is a sequence $\{C_t\}_{t=1}^{T}$ of cache configurations such that, for each $t \in [T]$, $C_t$ satisfies request $\sigma_t$. 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 trees), with a set $R$ being a descendant of $R'$ if $R \subseteq R'$. When discussing Slot-Laminar Paging we will routinely use tree-related terminology; for example, we will refer to some sets in $\mathcal{S}$

as leaves, roots, or internal nodes. The height $h$ of a laminar family $\mathcal{S}$ is one less 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 [36].

**Approximation algorithms.** An algorithm $\mathbb{A}$ for a given cost minimization problem is called a *$c$-approximation algorithm* if, for each instance $\sigma$, $\mathbb{A}$ 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 $\sigma$, $\mathsf{opt}(\sigma)$ is the optimum cost of $\sigma$, and $b$ is a constant independent of $\sigma$.

**Online algorithms and competitive ratio.** In the online variants of the paging problems studied in this paper the requests arrive online, one per time step, and an online algorithm needs to satisfy each request before the next one is revealed. To simplify presentation we assume that the algorithm knows the underlying set family $\mathcal{S}$ (or $\mathcal{P}$), but many of our algorithms work (or can be adapted to work) without knowing the set family in advance. An online algorithm $\mathbb{A}$ is called *$c$-competitive* if $\mathbb{A}$ is a $c$-approximation algorithm. As common in the literature, we will 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.

# 3 Slot-Heterogenous Paging

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

Theorems 1 (i) and 3 (Section 3.1) show that these bounds are tight within 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 5 (Section 3.2) shows an upper bound of $k^2|\mathcal{S}|$ on the optimal deterministic ratio for any family $\mathcal{S}$. For One-of-$m$ Paging, Theorems 5 and 1 (ii) (Section 3.1) imply that the optimal deterministic ratio is $O(k^{m+1})$ and $\Omega((4k/m)^{m/2}/\sqrt{m})$.

## 3.1 Lower bounds for Slot-Heterogenous Paging

We establish our lower bounds for Slot-Heterogenous Paging and One-of-$m$ Paging given in Table 1.

▶ **Theorem 1.**

**(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 1, we prove Lemma 2. It states that for any $\mathcal{S}$ the existence of a family $\mathcal{Z} \subseteq 2^{[k]}$ with certain properties implies a lower bound of $|\mathcal{Z}|$ on the competitive ratio. The proof of the theorem then constructs such families $\mathcal{Z}$ for appropriate families $\mathcal{S}$ of requestable sets. Throughout this section $\overline{X}$ denotes the complement of set $X \subseteq [k]$, that is $\overline{X} = [k] \setminus X$.

▶ **Lemma 2.** *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 $S \not\subseteq Z$ and $S \not\subseteq \overline{Z}$ 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. At each step, the adversary chooses a request that forces the algorithm to fault but causes at most two faults total among a fixed set of $2|\mathcal{Z}|$ other solutions. At the end, the algorithm's total cost is at least $|\mathcal{Z}|$ times the average cost of these other solutions, so its competitive ratio is at least $|\mathcal{Z}|$. This general approach is common for lower bounds on deterministic online algorithms (see e.g. lower bounds on the optimal ratios for $k$-Server [38], for Metrical Task Systems [13] and for Generalized $k$-Server on uniform metrics [36]).

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

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

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

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

Thus the combined total cost for all $Z$-strategies (not counting the cost of 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 1.** *Part (i).* Recall that $m = \lfloor (k+1)/2 \rfloor$. First consider the case when $k$ is odd. Apply Lemma 2, 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 2, 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.

*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 $\ell$. 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 $\ell$ edges.

First we define $\mathcal{G} \subseteq \mathcal{S}$ for Lemma 2. 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 1 show one $S$ in $\mathcal{G}$.) This set of $m/2$ edges uniquely determines $S$, and vice versa.

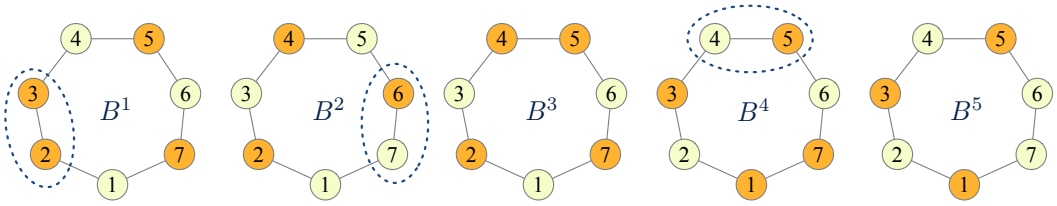


**Figure 1** Illustration of the proof of Theorem 1 Part (ii) for $k = 35$, $m = 6$, and $\ell = 7$. The figure shows the partition of all slots into $m - 1 = 5$ sets $B^1, \ldots, B^5$, each represented by a cycle. To avoid clutter, each slot $b^e_c$ is represented by its index $c$ within $B^e$. The 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 verify that $\mathcal{G}$ has Property (gz0) from Lemma 2. 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 2. 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 1 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 2. 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 $\ell$ ways to choose one edge. Lemma 2 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

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

**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 $\ell \leftarrow 1$ — *$\ell$ 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 $\sigma_t$: 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 $\sigma_t$ — *start the next phase*

**Figure 2** Online algorithm ExhSearch for Slot-Heterogenous Paging.

This gives us one estimate on the competitive ratio in Theorem 1(ii). To obtain a second estimate, squaring both sides of (2), we obtain

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

Therefore $f(m,k) = \Omega(\sqrt{|\mathcal{S}|})$, as claimed, completing the proof of Theorem 1(ii). ◀

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

▶ **Theorem 3.** *The optimal randomized ratio for One-of-$m$ Paging with $m = \lfloor k/2 \rfloor$ is $\Omega(k)$.*

The proof is by a reduction from standard Paging with some $N$ pages and a cache of size $N-1$. For any $N$, this problem has optimal randomized competitive ratio $H_{N-1} = \Theta(\log N)$ [29]. This and the next lemma, whose proof is deferred to the full paper [23], imply the theorem.

▶ **Lemma 4.** *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 $N = 2^{\Theta(k)}$.*

## 3.2 Upper bounds for deterministic Slot-Heterogenous Paging

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

▶ **Theorem 5.** *Fix any $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$. The competitive ratio of Algorithm* ExhSearch *in Figure 2 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. Theorem 5 can be strengthened slightly by making the algorithm retrieve at most $\min(t,k)$ pages for the $t$th request in each phase.

**Proof of Theorem 5.** By inspection of the algorithm, the length of each phase does not depend on previous phases. Focus on any one phase. We first bound the length, say $L$, of the phase. To ease notation and without loss of generality, assume the phase is the first (with $\ell = 1$) and the algorithm faults in each step $t$, that is $C_{t-1}$ does not satisfy $\sigma_t = \langle p_t, S_t\rangle$. (Otherwise first remove such requests; this doesn't change the algorithm's cost or increase the optimal cost.) So the following holds:

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

The final configuration $C_L$ in the phase satisfies all requests in the phase. In particular, for each $S \in \mathcal{S}$, for each request $\langle p, S\rangle$ in the phase, $C_L$ has $p$ in some slot in $S$, so *(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 $\sigma_t$ in this phase is distinct* (indeed, for any $t' < t$, $C_{t-1}$ satisfies $\sigma_{t'}$ but not $\sigma_t$). Observations (i) and (ii) imply the following 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 $\sigma$ 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 $\sigma$ is satisfied by a single configuration, 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. For each cache configuration $C_t$, let $C_t^i \in \mathbb{R}$ denote the page in slot $i$, if any, else 0. Define matrix $M \in \mathbb{R}^{L\times L}$ by

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

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

Expanding the formula for $M_{st}$, we obtain

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

where 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 $[t'+1, t''+1]$. 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 two upper bounds on $L$ imply Theorem 5. ◀

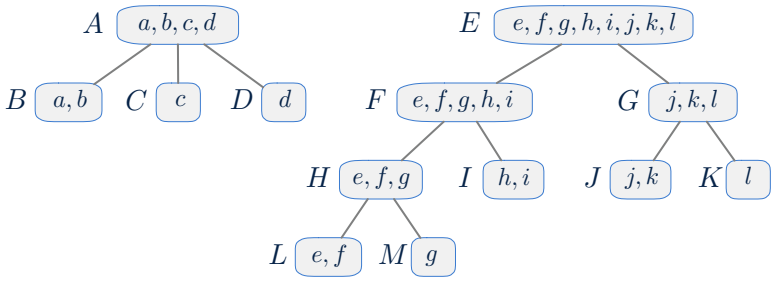


**Figure 3** An example of a laminar family $\mathcal{P}$ of height 4.

## 4 Slot-Laminar Paging

In this section we prove upper bounds for Slot-Laminar Paging given in Table 1. Recall that in Slot-Laminar Paging family $\mathcal{S}$ is assumed to be a laminar family of slot sets whose height we denote by $h$. Theorem 8 bounds the optimal ratios by $3h^2k$ (deterministic), $3h^2H_k$ (randomized) and $3h^2$ (offline polynomial-time approximation). The proof of Theorem 8 (Section 4.2) is by a reduction of Slot-Laminar Paging to Page-Laminar Paging, which we study in Section 4.1. Theorem 10, presented in Section 4.3, tightens the deterministic upper bound to $2hk$.

### 4.1 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. (If the number of non-leaf sets in $\mathcal{P}$ is constant, the problem can be solved in polynomial time using dynamic programming and Belady-like rules for choosing pages to fetch and evict.)

▶ **Theorem 6.** *Page-Laminar Paging admits the following polynomial-time algorithms: an $hk$-competitive deterministic online algorithm, an $hH_k$-competitive randomized online algorithm, and an offline $h$-approximation algorithm.*

The proof is by reduction to standard Paging. Known polynomial-time algorithms for standard Paging include an optimal offline algorithm [10], a deterministic $k$-competitive online algorithm [43] and a randomized $H_k$-competitive online algorithm [1]. Theorem 7 follows directly from composing these known results with the following lemma, whose proof is deferred to the full paper [23].

▶ **Lemma 7.** *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.*

### 4.2 Upper bounds for randomized and offline Slot-Laminar Paging

▶ **Theorem 8.** *Slot-Laminar Paging admits the following polynomial-time algorithms: a deterministic $3h^2k$-competitive online algorithm, a randomized $3h^2H_k$-competitive online algorithm, and an offline $3h^2$-approximation algorithm.*

Our focus here is on uniform treatment of the three variants of Slot-Laminar Paging in the above theorem, and the ratios in this theorem have not been optimized. For example, in Section 4.3 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 [20].

The proof of Theorem 8 is by a reduction of Slot-Laminar Paging to Page-Laminar Paging, in Lemma 9 (proved in Appendix A). 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 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 9 and Theorem 6 (for Page-Laminar Paging) immediately imply Theorem 8.

▶ **Lemma 9.** *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.*

## 4.3 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 8. 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 10.** *For Slot-Laminar Paging, Algorithm* RefSearch *(Fig. 4) has competitive ratio at most* $2 \cdot \mathsf{mass}(\mathcal{S}) - k \le (2h-1)k$.

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 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 11 (proved in the full paper [23]) establishes some basic properties of $\mathsf{rep}(R)$.

▶ **Lemma 11.** *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 $S$.*

Algorithm RefSearch is given in Figure 4. 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

**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 $1 \leq i < m$, 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 $1 \leq i \leq m$

**Figure 4** Deterministic online Slot-Laminar Paging algorithm RefSearch. Note that in Step 1.4.1 we have $m \leq k + 1 - |S|$, and that in (ii), if $s_m$ satisfies $\langle p, S' \rangle \in \mathsf{rep}(R_{t-1})$ then $m \geq 2$ (because $C_{t-1}$ does not satisfy $\sigma_t$); thus $S_{m-2}$ is well-defined.

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 11(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 11(iii).

▷ Claim 12. 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. We defer the proof of Lemma 13 and the full proof of Theorem 10 to Appendix A.

▶ **Lemma 13.** *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 $1 \leq i < m$, 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 $1 \leq i \leq m$, yields an $(R \cup \{r\})$-configuration.*

**input:** Weighted All-Or-One Paging instance $(k, \sigma)$, 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 $\langle p_t, s_t \rangle = \langle 0, t \rangle$ for $t \in [k]$ — *$k$ specific requests to artificial weight-0 page in each slot*
3. for $t \leftarrow k+1, k+2, \ldots, T$:
3.1. if $\langle p_t, s_t \rangle$ is a specific request with no equivalent request $t'$ (s.t. $\langle p_{t'}, s_{t'} \rangle = \langle p_t, s_t \rangle$) in the cache:
3.1.1. evict any cached general request to page $p_t$, and any cached request in slot $s_t$
3.1.2. put $t$ in slot $s_t$ — *note* $\mathsf{cap}[t] = \mathsf{credit}[t] = 0$
3.2. else if $\langle p_t, s_t \rangle$ is a general request not satisfied by any cached request $t'$ (s.t. $p_{t'} = p_t$):
3.2.1. let
$\ell_t(s) \coloneqq \max\{t' \le t : s_{t'} = s\}$ for $s \in [k]$ — *most recent specific request to $s$*
$A \coloneqq \{s \in [k] : \mathsf{cap}[\ell_t(s)] \ge \frac{1}{2}\mathsf{wt}(p_t)$, *$s$ does not hold a specific request*$\}$
$B \coloneqq \{s \in [k] :$ *slot $s$ holds a general request of weight at least* $\frac{1}{2}\mathsf{wt}(p_t)\}$
3.2.2. while $|A| \le |B|$:
3.2.2.1. continuously raise $\mathsf{cap}[\ell_t(s)]$ for $s \in [k]$ and $\mathsf{credit}[t']$ for each cached request $t'$, at unit rate,
3.2.2.2. evicting each request $t'$ such that $\mathsf{credit}[t'] = \mathsf{wt}(p_{t'})$, and updating $A$ and $B$ continuously
3.2.3. choose a slot $s \in A \setminus B$; evict the request $t'$ currently in slot $s$ (if any)
3.2.4. put $t$ in slot $s$ — *note* $\mathsf{credit}[t] = 0$
3.3. else: classify the (already satisfied) request as *redundant* and ignore it

**Figure 5** An $O(k)$-competitive online algorithm for Weighted All-Or-One Paging. For technical convenience, we present the algorithm as caching request times rather than pages, with the understanding that request $t$ represents page $p_t$.

## 5 Weighted All-Or-One Paging

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

▶ **Theorem 14.** *Weighted All-Or-One Paging has a deterministic $O(k)$-competitive online algorithm.*

The bound is optimal up to a small constant factor, as the optimal ratio for standard Weighted Paging is $k$. Figure 5 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 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 the classical linear-program relaxation will have value 1. Thus this linear program cannot be used to bound the competitive ratio.

**Proof sketch of Theorem 14.** 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 $2\,\mathsf{opt}(\sigma)$. In Appendix B, we show in Lemma 15 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 $\phi$ is at most $4k\,\mathsf{opt}(\sigma)$. To finish, we show by a charging argument that the algorithm's cost is at most $6\,\phi + 3\,\mathsf{opt}(\sigma) \le (24k+3)\,\mathsf{opt}(\sigma)$. We defer the full proof to Appendix B. ◀

## 6 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 5 tight for every $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$, within $\mathrm{poly}(k)$ factors?
- 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 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 $h \le k$) 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 $O(\mathrm{polylog}(k))$?
- The status of Heterogenous $k$-Server in arbitrary metric spaces is widely open. Can ratio dependent only on $k$ be achieved? This question, while challenging, could still be easier to resolve for Heterogenous $k$-Server than for Generalized $k$-Server.

## References


1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000. 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. *CoRR*, abs/2102.11119, 2021. arXiv:2102.11119.

4 Nikhil Bansal, Niv Buchbinder, Aleksander Mądry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 267–276. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.63.

5 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 507–517. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.7.

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 Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 992–1001, 2018. doi:10.1137/1.9781611975031.64.

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 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.

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

13 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.

14 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.

15 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.

16 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.

17 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.

18 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.

**19** Niv Buchbinder, Christian Coester, and Joseph (Seffi) Naor. Online k-taxi via double coverage and time-reverse primal-dual. In Mohit Singh and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization – 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2021. doi:10.1007/978-3-030-73879-2_2.

**20** 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.

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

**22** 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.

**23** Marek Chrobak, Samuel Haney, Mehraneh Liaee, Debmalya Panigrahi, Rajmohan Rajaraman, Ravi Sundaram, and Neal E. Young. Online paging with heterogeneous cache slots, 2022. arXiv:2206.05579.

**24** 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.

**25** 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.

**26** 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.

**27** 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.

**28** 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.

**29** 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.

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

**31** 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.

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

**33** 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.

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

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

36 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.

37 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.

38 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.

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

40 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.

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

42 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.

43 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.

44 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, pages 494–505, New York, NY, USA, 2007. doi:10.1145/1250662.1250723.

45 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.

46 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.


# A Proofs for Slot-Laminar Paging

## A.1 General upper bounds via reduction to page-laminar paging

**Proof of Lemma 9.** 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 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 $p$ being in slot $s$.

Formally, this relaxation is defined as follows. Fix any $k$-slot Slot-Heterogenous Paging instance $\sigma = (\sigma_1, \ldots, \sigma_T)$ with requestable slot-set family $\mathcal{S}$. For any page $p$ and $S \in \mathcal{S}$, define $V(p,S) = \{v(p,s) : s \in S\}$, where $v(p,s)$ is a *virtual page* for the pair $(p,s)$. Define the *relaxation* of $\sigma$ 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 $\pi$ 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 $\pi$ is a relaxation of $\sigma$ in the sense that for any solution $C$ for $\sigma$ there is a solution $D$ for $\pi$ with $\mathsf{cost}(D) \le \mathsf{cost}(C)$. (Namely, have $D$ keep in its cache the virtual pages $v(p,s)$ such that $C$ has page $p$ cached in slot $s$.) It follows that $\mathsf{opt}(\pi) \le \mathsf{opt}(\sigma)$.

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

Assume that the family $\mathcal{S}$ has just one root $R$ with $|R| \leq k$. (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* $\sigma_S$ to be obtained from $\sigma$ by deleting all requests that are not subsets of $S$. Let $\pi_S$ denote the (Page-Laminar Paging) relaxation of $\sigma_S$. Algorithm $\mathbb{B}$ on input $\sigma$ executes, simultaneously, $\mathbb{A}(\pi_S)$ for every requestable set $S \in \mathcal{S}$, giving each execution $\mathbb{A}(\pi_S)$ its own independent cache of size $|S|$ composed of copies of the slots in $S$.

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

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

**Correctness.** the following invariant over time. For each requestable set $S$, for each virtual page $v(p, s)$ currently cached by $\mathbb{A}(\pi_S)$:
1. the solution $\mathbb{B}(\sigma_S)$ caches $v(p, s)$ in some slot in $S$, and
2. if $S$ has a child $c$ with $s \in c$, and $\mathbb{B}(\sigma_c)$ has $v(p, s)$ in its cache $c$, then in $\mathbb{B}(\sigma_S)$ copy $v(p, s)$ is in the same slot as in $\mathbb{B}(\sigma_c)$.

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

To maintain the invariant $\mathbb{B}$ does the following for each requestable set $S$. Whenever the relaxed solution $\mathbb{A}(\pi_S)$ evicts a page $v(p, s)$, the solution $\mathbb{B}(\sigma_S)$ also evicts $v(p, s)$. After this eviction both Conditions 1 and 2 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 2 of the invariant: one at $\mathbb{B}(\sigma_S)$, because $v(p, s)$ is already cached by a child $\mathbb{B}(\sigma_c)$ but in some slot $s_1 \neq s'$; 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

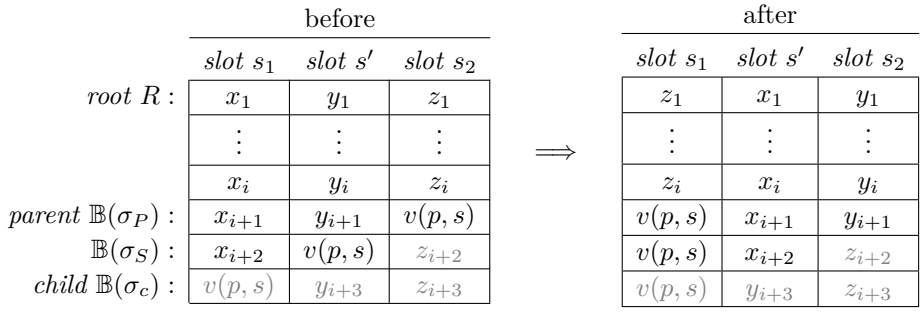


**Figure 6** "Rotating" slots in $\mathbb{B}(\sigma_S)$ and ancestors to preserve the invariant. Pages in grey are not moved.

create two 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 $\mathsf{cost}(\mathbb{B}(\sigma_R))$

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

The second step uses that $\mathbb{A}(\pi_S)$ is $f_h(|S|)$-competitive for $\pi_S$. The third step uses that $\pi_S$ is a relaxation of $\sigma_S$ so $\mathsf{opt}(\pi_S) \leq \mathsf{opt}(\sigma_S)$, and that $|S| \leq k$ so $f_h(|S|) \leq f_h(k)$.[1] 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. ◀

## A.2 Improved upper bound for deterministic case

**Proof of Lemma 13.** 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 12 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 12 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.

[1] 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.

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 $2 \leq i < m$; 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 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 $2 \leq i \leq m$, 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 step 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 $1 \leq i \leq m$, 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 10.** 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 $\sigma_t$, 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 $\sigma_t$. Then, by Lemma 13, 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 $\sigma_t$ 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 $m/(m-1) \leq 2$. (Note that in this case $m \geq 2$.) When the phase ends, a request at depth $d$ was charged at most $d$ times, and these charges include at least a unit charge, so its total charge is most $2d-1$. So, the algorithm's cost per phase is at most $2 \cdot \mathsf{mass}(\mathcal{S}) - k \leq (2h-1)k$. 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. ◀

## B Proofs for Weighted All-Or-One Paging

Consider any execution of the algorithm on a $k$-slot instance $\sigma$. To ease notation and streamline the analysis, without loss of generality we will make the following assumptions:

- The first $k$ requests are specific requests for an artificial 0-wt page 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 0-wt page in each of the $k$ slots.

These assumptions can 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. For technical convenience, we think of the algorithm and the optimal solution as caching request *times* rather than pages, with the understanding that request $t$ represents page $p_t$.

▶ **Lemma 15.** *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, in any solution $C$, just after $C$ has responded to request $t$, either*
*(i) $C$ has evicted some request $t'$ currently cached by the algorithm, 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. 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 $|B| + 1 \geq |A| + 1$ 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 $\mathsf{wt}(p_{t'}) \geq \frac{1}{2}\,\mathsf{wt}(p_t) > \mathsf{cap}[\ell_t(s)]$, so property (ii) holds. ◀

**Proof of Theorem 14.** 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 15, 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 $\phi$ is at most $4k\,\mathsf{opt}(\sigma)$. To finish, we show that the algorithm's cost is at most $6\,\phi + 3\,\mathsf{opt}(\sigma) \leq (24k + 3)\,\mathsf{opt}(\sigma)$. 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 $\sigma$ 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 $3\,(2\,\phi + \mathsf{opt}(\sigma)) = 6\,\phi + 3\,\mathsf{opt}(\sigma)$. ◀