SIAM J. COMPUT.
Vol. 51, No. 4, pp. 975–1017

CACHING WITH TIME WINDOWS AND DELAYS*

ANUPAM GUPTA†, AMIT KUMAR‡, AND DEBMALYA PANIGRAHI§

Abstract. We consider two generalizations of the classical weighted paging problem that incorporate the notion of delayed service of page requests. The first is the *(weighted) paging with time windows* (PageTW) problem, which is like the classical weighted paging problem except that each page request only needs to be served before a given deadline. This problem arises in many practical applications of online caching, such as the "deadline" I/O scheduler in the Linux kernel and video-on-demand streaming. The second, and more general, problem is the *(weighted) paging with delay* (PageD) problem, where the delay in serving a page request results in a penalty being added to the objective. This problem generalizes the caching problem to allow delayed service, a line of work that has recently gained traction in online algorithms (e.g., [Y. Emek, S. Kutten, and R. Wattenhofer, *Proceedings of the* 48*th Annual ACM SIGACT Symposium on Theory of Computing,* 2016, pp. 333–344; Y. Azar et al., *Proceedings of the* 49*th Annual ACM SIGACT Symposium on Theory of Computing,* 2017, pp. 551–563; Y. Azar and N. Touitou, *Proceedings of the* 60*th IEEE Annual Symposium on Foundations of Computer Science*, 2019, pp. 60–71]). We give $O(\log k \log n)$-competitive algorithms for both the PageTW and PageD problems on n pages with a cache of size k. This significantly improves on the previous best bounds of $O(k)$ for both problems [Y. Azar et al., *Proceedings of the* 49*th Annual ACM SIGACT Symposium on Theory of Computing,* 2017, pp. 551–563]. We also consider the *offline* PageTW and PageD problems, for which we give $O(1)$-approximation algorithms and prove APX-hardness. These are the first results for the offline problems; even NP-hardness was not known before our work. At the heart of our algorithms is a novel "hitting-set" LP relaxation of the PageTW problem that overcomes the $\Omega(k)$ integrality gap of the natural LP for the problem. To the best of our knowledge, this is the first example of an LP-based algorithm for an online problem with delays/deadlines.

Key words. online algorithms, caching, approximation algorithms

MSC code. 68

DOI. 10.1137/20M1346286

1. Introduction.

In the caching/paging problem, page requests from a universe of n pages arrive over time. They have to be served by swapping pages in and out of a cache that can hold only $k < n$ pages at a time. In weighted paging, each page p has a weight w_p, which is the cost of fetching the page into the cache. The goal is to minimize the total cost of fetching pages over all requests. In this paper we consider situations where page requests do not need to be served immediately, but can be delayed for some time. For instance, in mixed-workload environments such as those arising in cloud computing or operating systems, requests from time-sensitive applications (such as interactive ones) have short deadlines, but batch processes can

*Received by the editors June 17, 2020; accepted for publication (in revised form) March 31, 2022; published electronically July 7, 2022. A preliminary version of this paper appeared in *Proceedings of the* 52*nd Annual ACM Symposium on Theory of Computing*, 2020.

https://doi.org/10.1137/20M1346286

Funding: This research was done under the auspices of the Indo-US Virtual Networked Joint Center IUSSTF/JC-017/2017. The first author was supported in part by NSF award CCF-1907820. The third author was supported in part by NSF award CCF-1535972 and NSF CAREER award CCF-1750140.

†Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA (anupamg@cs.cmu.edu).

‡Department of Computer Science and Engineering, IIT Delhi, New Delhi, India (amitk@cse.iitd.ac.in).

§Department of Computer Science, Duke University, Durham, NC 27708 USA (debmalya@cs.duke.edu).

tolerate longer wait times. (Indeed, the "deadline" I/O scheduler in the Linux kernel is precisely for this purpose, although the way it currently handles deadlines is not very sophisticated [Lin].) A different application arises in network streaming, e.g., in video-on-demand, where a server needs to cache segments appearing in multiple video streams (see, e.g., [CBD+15, DDH+12]). Depending on when these segments are required, various streams set different deadlines for each of these segments. In all these applications, the key feature is that individual page requests can be delayed, but only until a given deadline. Specifically, the request $r_t = (p, d)$ at time t for a page p includes a deadline d, and the algorithm must ensure that the page is in the cache at some time in the interval $[t, d]$. (We consider time to be in discrete steps, and the page must be in the cache at some time instant in the interval $[t, d]$. We also assume that the insertion/eviction of pages from the cache is an instantaneous process.) We call this the *(weighted) paging with time windows* (PageTW) problem; if the deadline is the same as the time of the request, we get back the weighted paging problem.

A more general setting is one where the page requests do not have specific deadlines, but the algorithm incurs a cost that is monotonically nondecreasing with the delay in serving individual requests. This is related to the recent line of work in online algorithms with delay, where problems such as online matching [EKW16, ACK17, AAC+17, AF20] and online network design [AT19, AT20] have been considered. In particular, our work relates to the "online service with delays" problem [AGGP17, BKS18, AT19] and can be interpreted as a generalization of this problem to k servers but for the special case of a star metric. In our problem, each request is specified by a triple (p, t, F), where p is the requested page, t is the time at which this request is made, and $F : \{t, t+1, \dots, \} \to \mathbb{R}_{\geq 0}$ denotes the nondecreasing loss function associated with it. The objective is to minimize the sum of two quantities: the sum of weights of pages evicted from the cache (the usual objective in weighted paging) and *the total delay losses incurred over all the individual page requests.* (Note that as is the case in other online problems with delay, we also assume that the two cost functions, that of fetching pages into the cache and the penalty associated with delayed service of page requests, are calibrated to the same scale so that they can be added to obtain the overall cost.) We call this the *(weighted) paging with delay* (PageD) problem. Note that PageTW is a special case of this problem where the delay loss is 0 till the deadline and ∞ thereafter.

THEOREM 1.1 (main results: online algorithm). *There is an $O(\log k \log n)$-competitive randomized algorithm for the PageD problem in the online setting, where n is the number of pages and k is the size of the cache. As a consequence, there is also an $O(\log k \log n)$-competitive randomized algorithm for the special case of the PageTW problem in the online setting.*

Previously, an $O(k)$-competitive deterministic algorithm was given by Azar et al. [AGGP17] for both problems. PageD and PageTW inherit an $\Omega(\log k)$-competitiveness lower bound from the classical paging problem; closing the gap between our upper bound and this lower bound remains open.

While we stated the above theorem for the more general PageD problem, and derived the bound for PageTW as a corollary, we will actually prove the theorem for the special case of the PageTW problem first, and then show that we can reduce the PageD problem to the PageTW problem. More precisely, we extend our PageTW algorithm to a generalization that we call the PageTWPenalties problem, where every page request has a nonnegative penalty that the algorithm can choose to incur instead of satisfying the request. Then, we give a reduction from the PageD problem to the

PageTWPenalties problem in section 6 without changing the objective; moreover, this reduction can be performed online. So, the rest of this section, and much of the subsequent sections, focuses on the PageTW and PageTWPenalties problems.

At the heart of our algorithm is a novel "hitting-set" LP relaxation of the PageTW problem that overcomes the $\Omega(k)$ integrality gap of the natural LP relaxation for this problem (see Appendix B.2). From a theoretical perspective, the PageTW problem is in the category of online optimization problems with delays/deadlines that has attracted significant interest recently (e.g., [EKW16, AGGP17, ACK17, BFNT17, AT19, BBB$^+$16]). To the best of our knowledge, our work is the first example of an LP-based algorithm in this line of research. Given the great success of LP-based techniques in online algorithms in general, we hope that our work spurs further progress in this area.

We also study the *offline* versions of the PageTW and PageD problems, where the request sequence is given up front. Here, the first question is tractability: since weighted paging is solvable in polynomial time offline, it is conceivable that so are PageTW and PageD. We show that the PageTW problem (and therefore, by generalization, the PageD problem) is APX-hard. We complement this lower bound with an $O(1)$-approximation for the offline PageTW and PageTWPenalties problems, which again by our reduction from the PageD problem to the PageTWPenalties problem implies an $O(1)$-approximation for the offline PageD problem.

THEOREM 1.2 (main results: offline algorithm). *The PageTW problem is NP-hard (and APX-hard) even when the cache size k is equal to 1, and the pages have unit weight. As a consequence, the PageD problem is also NP-hard (and APX-hard) under these restrictions. Moreover, there are $O(1)$-approximation deterministic algorithms for the PageTW and PageD problems, based on rounding a linear program to show a constant integrality gap.*

1.1. Our techniques. The weighted paging problem has an "interval covering" IP formulation [BBN12, You91] (here, p_t represents the page requested in time-step t):

$$\min \left\{ \sum_{p,j} w_p x_{p,j} : \sum_{p \neq p_t} x_{p,j(p,t)} \geq n - k \ \forall t, x_{p,j} \in \{0,1\} \ \forall p, j \right\}.$$

For every page p, define an interval starting at each request for it, and ending just before the next request. (These requests are indexed by j, i.e., the interval starts at the jth request and ends in the time-step immediately before the $(j+1)$st request. Inverting the notation, we use $j(p,t)$ to indicate the value of j for page p at time t. In other words, $j(p,t) = j$ if t is at or after the jth request and before the $(j+1)$st request for page p.) Because of the jth request, this page p must be present in the cache at the start of each such interval, but may be evicted at some subsequent point: the IP variable $x_{p,j} \in \{0,1\}$ indicates if a page is evicted before its next, i.e., the $(j+1)$st, request. While this IP does not explicitly indicate *when* a page is evicted, any online algorithm solving it must raise a variable $x_{p,j}$ from 0 to 1 at a specific time between the jth and $(j+1)$st request for page p. We visualize this using a two-dimensional picture indexed by the pages and time, recording the eviction of page p at time t by putting a star at location (p,t) (see Figure 1). In classical paging, the intervals for any page partition its row into disjoint, tightly fitting segments. The capacity constraint of the cache forces the following property: of the n intervals (for different pages) containing time t (these are indexed $j(p,t)$ for page p), at least $n-k$ contain a

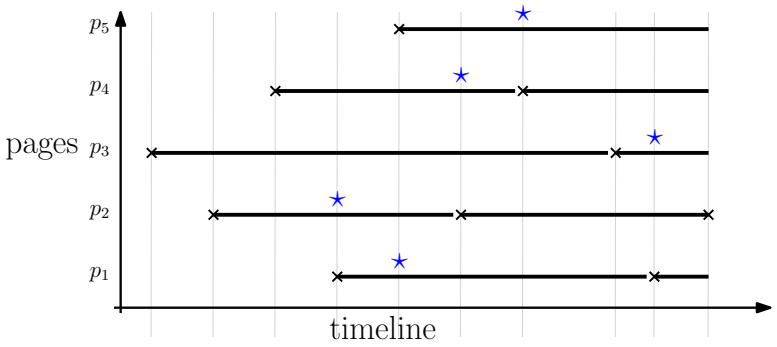

FIG. 1. *A two-dimensional view of page requests and evictions. The crosses represent page requests and the stars represent page evictions. This illustration is for a cache of size* 3.

star at some time $\leq t$. In other words, at least $n-k$ pages must have been evicted from the cache since their last request.

The situation is more complex in PageTW. Previously, it sufficed to record page evictions, because page insertions were entirely dictated by the requests: whenever a page is requested, it must be inserted in the cache if it were evicted after its previous request. So, for an insertion to be feasible, it suffices to just ensure that sufficiently many pages are evicted since their previous request. In PageTW, however, page requests can be fulfilled at a later time, so evictions alone do not completely describe the state of the cache. One option is to explicitly encode page insertions via IP variables, but then we need packing constraints on these variables to enforce the size of the cache. Handling such packing constraints in online IPs seems beyond the scope of current techniques in online algorithms. Another idea is to reduce the ambiguity of when pages are inserted in the cache, e.g., by enforcing that all page insertions are done at the end of their request intervals (if the page is not in the cache at the beginning of the interval). This would be a useful property, because the state of the cache could then be completely described by variables for page evictions. This property, however, is false: forcing a page request to be satisfied at the start/end of its request interval can be much costlier than doing it somewhere in the middle. For example, if a heavy page is being evicted, we should serve some outstanding light requests while there is an empty slot in the cache (see Appendix B.1).

The hitting set IP relaxation. To overcome these challenges, we first reinterpret the interval covering IP for classical paging. We use $x_{p,t}$ variables which denote whether page p is evicted at time t. The cache-size constraint at any time t' insists that at least $n-k$ pages are evicted at times $\leq t'$ since their last request. To implement this, we define an interval for each page p starting at the last request for p and ending at t', and write a covering constraint saying at least $n-k$ of these intervals have a star within them (i.e., $x_{p,t}=1$ for times t within these intervals). Note that there is nothing special about the *last* request for a page before t'—we could have written these constraints for every choice of request of every page before t'. The additional constraints would be redundant given the one containing the last requests and would unnecessarily lead to an exponential-sized IP.

In the PageTW problem, however, the request intervals for a page might overlap, or may even be nested, so it is easier to write constraints for *every* request, rather than to identify some (noncanonical) *last* request before time t'. Extending the previous intuition, we define the following intervals for time t': corresponding to a request

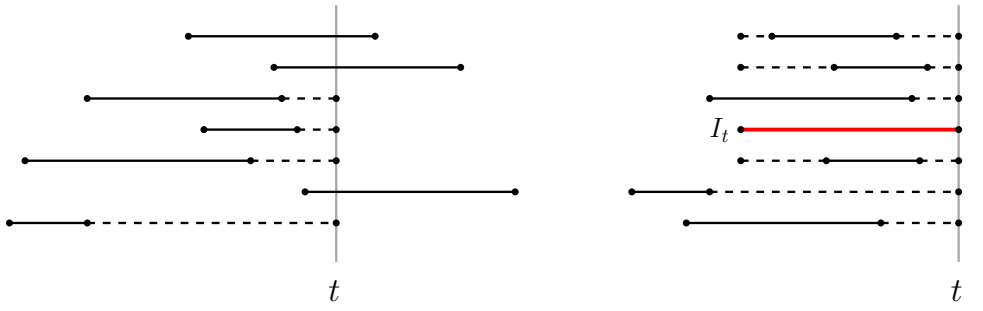

FIG. 2. *The left figure illustrates right extensions* $\mathsf{Rext}(I,t)$ *for request intervals* I *shown by solid lines. The dotted lines show how these intervals are extended. The figure on the right shows double extensions* $\mathsf{Dext}(I,t)$*, with the critical interval* I_t *in red.*

interval $I = (s(I), e(I))$ for a page with $s(I) < t'$, there is a constraint interval $(s(I), \max(e(I), t'))$. Note that if the request interval extends beyond t', i.e., $e(I) > t'$, then we must extend the constraint interval rightward to $e(I)$, since the page might be served *after* t'. Now, we enforce the same constraint as earlier: for any choice of such constraint intervals, one for each distinct page, at least $n - k$ must have a star in them. We call these constraint intervals the *right extensions* of their respective request intervals at time t' (see Figure 2 for an example).

In classical paging, these constraints are valid even if we exclude the page currently requested at time t'. In other words, of the remaining $n - 1$ pages, the constraint ensures that at least $n - k$ have been evicted ensuring a cache slot for the currently requested page. All feasible solutions satisfy this constraint since the requested page *must* be in the cache at time t'. This stronger constraint, however, does not hold for PageTW. If we write the above constraints for $n-1$ pages, then it would reserve a cache slot for the remaining page at the current time, thereby excluding feasible solutions that do not satisfy this property. Conversely, the (weaker) constraints summing over all n (and not $n - 1$) pages are not sufficient: they do not reserve a cache slot for a requested page at any time during the request interval.

So we need a new set of constraints. These reserve a cache slot for a page p within each request interval $I_t = (s_p, t_p)$ for it. Let us exclude this page p and choose a request $I = (s(I), e(I))$, where $e(I) \le t_p$, for each of the remaining $n - 1$ pages. For each such request (say, for a page p'), consider a different extended constraint interval $(\min(s_p, s(I)), t_p)$. Now we are guaranteed that in any feasible solution, one of two things happens: either page p' resides in the cache for the entire extended interval $(\min(s_p, s(I)), t_p)$ and therefore also for the subinterval (s_p, t_p), or it is "hit" (inserted or evicted) during the constraint interval. Since page p must be served in its request interval (s_p, t_p), at most $k - 1$ pages can be resident in the cache during (s_p, t_p), i.e., at least $n - k$ of the $n - 1$ pages are hit during these extended constraint intervals. We call these extended intervals *double extensions* of their request intervals for time t_p (again, see Figure 2). Our "hitting set" IP comprises these two sets of requests, for right extensions and double extensions. We give details of this formulation in section 2.

Solving the hitting set IP online. Loosely, we extend ideas from Bansal, Buchbinder, and Naor [BBN12] for solving the weighted paging IP online to our hitting set IP. There are some challenges, however. First, the hitting set IP is of exponential size, since we wrote covering constraints for every choice of request interval for every page.

Second, unlike in weighted paging, there are two sets of constraints, one on $n-1$ pages and the other on n pages; the weighted paging IP only has the first set. Third, the decision variables are for (p,t) pairs and do not uniquely correspond to constraint intervals. Nevertheless, we show that, as long as the request intervals for the pages are "nonnested," the techniques of [BBN12] can be adapted to our hitting set IP. When the request intervals are nested, we solve the problem on two carefully selected subsets of the input where the request intervals are nonnested; then we show, somewhat surprisingly, that the combined solution satisfies the general instance. The competitive ratio of this algorithm is $O(\log k)$, asymptotically the same as weighted paging. This algorithm appears in section 5.

Converting IP solution to cache schedule online. As described earlier, the IP solution only gives us a set of stars, indicating "hits" for each page where each hit might represent either insertion or eviction of the page from the cache. Moreover, the IP solution does not necessarily give *all* the insertions and evictions. For example, in the case of instantaneous request intervals representing classical weighted paging, our IP is identical to the standard interval covering IP and only gives page evictions. Indeed, the bulk of our technical work is in converting a feasible IP solution to an actual cache schedule that satisfies all requests. This is further complicated by the fact that this translation has to be done online.

The main difficulty is the following: when a request interval I for a page p arrives, we don't know how long to wait before serving it. For instance, suppose the IP has a "hit" for a heavy page. The example in Appendix B.1 shows that we must use this opportunity to serve requests for light pages that are currently waiting. But, which pages should we serve? Suppose we serve a page p at some time $t \in I$ by loading p in the cache, and evict it soon after to serve other light pages. Now if another interval I' for p arrives after t and I' overlaps with (or is even nested in) I, it is clear that we should have waited to load p till I' arrives. In the offline case, we can use a reverse-delete step where we undo such mistakes. But, in the online setting, we must find a careful balance between waiting "long enough" and servicing outstanding requests. Specifically, we build a tree structure over the request intervals (which may not be laminar in general), and use the structural properties to argue that our algorithm can find a balance between these two competing goals. The online conversion algorithm appears in section 3.

While we cannot show that our algorithm achieves the ultimate goal of being $O(\log k)$-competitive, we do not know any worse gaps for our approach. Indeed, the fact that the integrality gap of the hitting set formulation is constant, as evidenced by our offline solution, gives us hope that the ideas here will lead to further improvements.

1.2. Related work. Azar et al. [AGGP17] study the online service problem with delays, where a single server services requests in a metric space. Each request has an associated monotone delay function that gives the cost of serving requests at each time after its arrival. The server pays for the total movement plus delay costs. They give an $O(h^3)$-competitive algorithm for hierarchically separated trees (HSTs) of height h. They extend the result to k servers at a loss of a factor of k, which gives an $O(k)$-competitiveness for PageTW. (Progress on related problems appears in [AT19].) A related problem is *online multilevel aggregation* [BBB+16] where a single server sits at the root of a tree, requests arrive at the leaves, and the server occasionally goes to service some subset of requests and returns to the root. The cost is again the sum of movement and delay costs. Buchbinder et al. gave an $O(h)$-competitive algorithm for h-level HSTs [BFNT17], improving on [BBB+16]; the model itself combines elements

of TCP acknowledgment [KKR03] and online joint replenishment [BKL+13]. Online problems with delays were first proposed by Emek, Kutten, and Wattenhofer [EKW16] for online matching; see [ACK17, AAC+17] for other work.

In the classical paging/caching problem with instantaneous requests, each interval is of length zero and must be satisfied immediately. Belady's offline algorithm (Farthest in Future) is optimal for the number of evictions [Bel66]; in contrast, the offline PageTW problem is APX-hard. We know deterministic k-competitive and randomized $O(\ln k)$-competitive algorithms for the classical caching problem; both are optimal [ST85, FKL+91]. *Weighted paging* is equivalent to the k-server problem on a weighted star, so deterministic k-competitiveness follows from the algorithm k-server on trees [CKPV91]. Bansal, Buchbinder, and Naor [BBN12] gave a randomized $O(\ln k)$-competitive algorithm for weighted paging, illustrating the power of the primal-dual technique for these problems. They used an interval covering IP give by [BBF+01, CK99], which we extend in our work.

Paper outline. In section 2, we describe the new IP formulation for PageTW. In section 3, we show how a solution to this IP can be used to generate a caching schedule online. We give the corresponding offline algorithm in section 4. We show how to (approximately) solve the IP, both offline and online, in section 5. The reduction from the PageD problem to the PageTW problem that changes the objective by at most an $O(1)$-factor appears in section 6. We prove APX-hardness of PageTW in Appendix A and give some illustrative examples in Appendix B.

2. The IP relaxations for PageTW and PageTWPenalties. There is a universe of n pages, and the cache can hold k pages at any time. Each page p incurs a cost when we fetch it into the cache, which is denoted by its *weight* $w(p) \geq 0$. We assume that the problem instance starts with an empty cache, which ensures that we can switch freely between assigning cost to fetching or evicting pages (this adds at most one to the competitive ratio). Therefore, in the rest of the paper, we will pay for either fetching or evicting pages, making sure that we pay for either of these two steps for every page that enters the cache. Moreover, for the optimal solution, we will charge it for both fetching and evicting the same page, since this only double charges the optimum.

In the *(weighted) paging with time windows* (PageTW) problem, each request specifies a page p and an interval $I = [s(I), t(I)]$: the page p must be in the cache at some time during this interval I. Since the only times of interest in the problem are the start and end times of intervals, we assume without loss of generality (wlog) that $s(I), e(I) \in \mathbb{Z}$, so the interval $I := \{s(I), s(I)+1, \ldots, e(I)\}$. Note that in the traditional paging problem, each interval I contains a single time-step, i.e., $I = \{t\}$ for some t. In the online setting, a request comprising the identity of the page and the end time of the interval $e(I)$ (i.e., the *deadline*) is revealed at its start time $s(I)$. This is known as the *clairvoyant* setting in the literature; strong lower bounds are known for the nonclairvoyant setting where the deadline is only revealed at time $t(I)$ [AGGP17].

We write a "hitting set" *integer* programming relaxation for this problem: this IP does not capture the PageTW problem exactly, but we show that (a) it contains only valid constraints, and hence provides a lower bound on the optimal cost (assuming we count both evictions and insertions of pages), (b) it can be solved approximately in polynomial time, and (c) the "relaxation" gap is small, i.e., a solution to this IP can be used to obtain a feasible solution to the original PageTW problem.

The IP has Boolean variables x_{pt} for each page-time pair (p, t), with this variable being set if page p is "hit" at time t, i.e., it is either brought into or evicted from the

cache at time t. We assume that for each time $t \in \mathbb{Z}$, there is exactly one request interval I having $e(I) = t$. This incurs no loss of generality because of the following transformation, which can be done online:

(a) For a time-step t that is not the end of any interval, we eliminate the time-step t from the IP formulation and move the start time for any interval(s) starting at t to $t+1$. (This is done online for each time-step in chronological order; i.e., if no interval ends at time-step $t+1$ either, then we apply the same rule to $t+1$ and move the starting time of all intervals starting at $t+1$ to $t+2$. This includes intervals whose start time was moved from t as well.)

(b) For a time-step t where multiple (say, ℓ) intervals end, we create ℓ time-steps between t and $t+1$ (including t itself) and set the end times for each of the ℓ intervals to these ℓ times after ordering the intervals by increasing start times (breaking ties arbitrarily). For instance, if $\ell = 2$ and if the two intervals are $[s_1, t]$ and $[s_2, t]$, then we create two time-steps t and t' where $t < t' < t+1$ and the new intervals are now defined as $[s_1, t]$ and $[s_2, t']$.

Note that the transformation above can be done online. Moreover, although this changes the index of the time-steps in the instance, we can reindex all the time-steps in chronological order (this can also be done online). Moreover, this transformation is purely cosmetic in that it does not change the problem instance: namely, no start/end times for intervals are inverted in terms of chronological order. This implies that solutions between the transformed and the original instance have a bijective map with equal cost. This allows us to make this transformation online and assume wlog that each time-step has exactly one interval ending at it.

Hence each request interval I corresponds to a unique page $\mathsf{page}(I) \in [n]$. For time t, let I_t and p_t be the unique interval ending at time t and its corresponding page; we call these the *critical interval and page* for time t.

As described in the introduction, we use two sets of extensions for request intervals to define the covering constraints of our IP:

(2.1)
$$\text{if } s(I) \le t \quad \Longrightarrow \quad \text{right extension of } I \quad \mathsf{Rext}(I,t) := [s(I), \ldots, \max(t, e(I))].$$

(2.2)
$$\text{if } e(I) \le t \quad \Longrightarrow \quad \text{double extension of } I \quad \mathsf{Dext}(I,t) := [\min(s(I_t), s(I)), \ldots, t].$$

(See Figure 2.) The "hitting set IP" below has variables $x_{pt} \in \{0,1\}$ (here $\mathcal{C}$ refers to a collection of request intervals):

(IP)
$$\min \sum_{p,t} w(p)\, x_{p,t},$$

(R1)
$$\sum_{I \in \mathcal{C}} \sum_{t' \in \mathsf{Rext}(I,t)} x_{\mathsf{page}(I),t'} \ge 1$$
$$\forall t \forall \mathcal{C} \text{ with } k+1 \text{ requests for distinct pages starting before } t,$$

(D1)
$$\sum_{I \in \mathcal{C}} \sum_{t' \in \mathsf{Dext}(I,t)} x_{\mathsf{page}(I),t'} \ge 1$$
$$\forall t\, \forall \mathcal{C} \text{ with } k \text{ requests for distinct pages (excluding } p_t) \text{ ending before } t.$$

We now show that these sets of constraints are valid.

CLAIM 2.1. *Any solution to* PageTW *yields a feasible solution to* (IP) *of equal cost.*

Proof. Fix a solution $\mathcal{S}$ for PageTW. Set $x_{p,t}$ to 1 if this page p is evicted at time t or loaded into the cache at time t in this solution. Consider the constraint (R1) for a collection $\mathcal{C}$ and time t. At time t, one of the $k+1$ pages corresponding to $\mathcal{C}$ is not in the cache—let the corresponding request interval be $I \in \mathcal{C}$ for page $p = \mathsf{page}(I)$. Two cases arise: in the solution the page p is in the cache either (a) at some time during $[s(I), t)$, or (b) at some time during $[t, e(I)]$. (The latter case arises only if $t \le e(I)$.) In the first case, p must have been evicted during $[s(I), t]$, whereas in the second case it must have been brought into the cache during $[t, e(I)]$. In either case the $x_{p,t'}$ variables sum to at least 1 over the right-extended interval for I with respect to t.

Now consider the constraint (D1) for a collection $\mathcal{C}$ and time t, where the critical request at time t is I_t with $p_t := \mathsf{page}(I_t)$. In the solution $\mathcal{S}$, let page p_t be in the cache at some time $\tau \in I_t$. At this time, at least one of the k pages corresponding to the intervals in $\mathcal{C}$ is not in the cache; say this interval is I for page $p := \mathsf{page}(I)$. Again two cases arise: in the solution $\mathcal{S}$, this page is in the cache either (i) at some time during $[s(I), \tau]$ (where this case arises only if $s(I) \le \tau$), or (ii) at some time t during $[\tau, e(I)]$ (again, this case arises only if $\tau \le e(I)$—note that at least one of these two cases must happen). If the former case happens, then p must have been evicted during $[s(I), \tau]$, whereas if the second case happens, then p must have been brought in the cache during $[\tau, e(I)]$. Since $\tau \in I_t \in [s(I_t), e(I_t) = t]$, in both cases the $x_{p,t'}$ variables sum to at least 1 for the doubly extended interval for I with respect to t. ∎

In the *(weighted) paging with time windows and penalties* (PageTWPenalties) problem, each request interval I has an associated penalty value $\ell(I) \ge 0$, which is the penalty for not satisfying the request associated with interval I. Not all requests must be satisfied, but if some request is not satisfied we must pay the penalty for it. The IP for the PageTWPenalties problem is very similar, where y_I is the indicator for whether we choose to take the penalty.

(IPp)
$$\min \sum_{p,t} w(p)\, x_{p,t} + \sum_{I} \ell(I)\; y_I,$$

(R1)
$$\sum_{I \in \mathcal{C}} \Big(y_I + \sum_{t' \in \mathsf{Rext}(I,t)} x_{\mathsf{page}(I),t'} \Big) \ge 1$$
$$\forall t\, \forall \mathcal{C} \text{ with } k+1 \text{ requests for distinct pages starting before } t,$$

(D1)
$$\sum_{I \in \mathcal{C}} \Big(y_I + \sum_{t' \in \mathsf{Dext}(I,t)} x_{\mathsf{page}(I),t'} \Big) \ge 1 - y_{I_t}$$
$$\forall t\, \forall \mathcal{C} \text{ with } k \text{ requests for distinct pages (excluding } p_t) \text{ ending before } t.$$

This IP is a strict generalization of (IP), since we can set $\ell(I) = \infty$ to force the $y_I = 0$. The proof of validity of (IPp) for the PageTWPenalties problem is identical to Claim 2.1 above and is omitted.

3. Solving PageTW and PageTWPenalties online using online solutions to (IP) and (IPp). Now that we have the IPs, we need to solve them online and

also show how to convert a solution into one for the PageTW or PageTWPenalties problem. Indeed, (IP) and (IPp) do not have any explicit capacity constraints, so we need to extract a "schedule" from the IP solution in an online manner. In this section we show the latter step; we discuss solving the IPs in section 5.

THEOREM 3.1. *There is an online algorithm that converts an α-competitive integral solution to* (IPp) *into a valid $O(\alpha \log n)$-competitive solution for the PageTWPenalties instance. As a special case, there is an online algorithm that converts an α-competitive integral solution to* (IP) *into a valid $O(\alpha \log n)$-competitive solution for the PageTW instance.*

Open problem. An intriguing open question is whether the above theorem can be improved by eliminating the $\log n$ terms in the competitive ratio. Note that our offline algorithm shows that there is no (superconstant) gap between the cost of an integral solution to (IPp) and a valid solution for the corresponding PageTWPenalties instance. This suggests that it might be possible to remove the extra $\log n$ term even in the online setting.

Although we stated this theorem in terms of the more general PageTWPenalties problem, we will actually prove it for the simpler PageTW problem. This is wlog since an integer solution to (IPp) for an instance of the PageTWPenalties problem already specifies the page requests that are being satisfied by the solution, and the ones where the solution incurs the penalty. Given an instance of the PageTWPenalties problem, we simply remove the requests in the latter set to create an equivalent instance of the PageTW problem. On this instance, we apply the above theorem for the PageTW problem to recover the theorem for the PageTWPenalties instance.

In the rest of the paper, we move between solutions x to (IP) and their characteristic set $A^\star := \{(p,t) \mid x_{p,t} = 1\}$. Visually, thinking of time as the x-axis and the n pages as the y-axis, the solution $A^\star$ corresponds to a set of "stars" in the two-dimensional plane. Let us list some properties of the online solution $A_t^\star$ to (IP) (which should satisfy all the constraints corresponding to times t and earlier) that are maintained by the algorithm in section 5.

(A1) *Monotonicity*: $A_t^\star \subseteq A_{t+1}^\star$ for all t.

(A2) *Past-preservation*: At time t the algorithm only adds stars corresponding to times t or later. Ideally, at time t, it should only add stars at time t, with the following exception.

(A3) *Sparsity*: For every page p, $A_t^\star$ contains at most one star (p,t') with $t' > t$. Furthermore, if $A_t^\star$ has such a star, then this star hits all the request intervals for p which contain time t. In fact, our online algorithm for PageTW does not need to know the exact location of the stars after time t—it just needs to know the set of pages p for which the solution $A_t^\star$ contains such a star.

The main idea of the algorithm is that if the cache is full and we need to evict a heavy page p, we should spend about $w(p)$ amount of weight in serving other outstanding requests at time t. The requests that need to be serviced need to be carefully chosen, because there are conflicting goals: (i) we want to service the cheaper requests, because this way we can service many of these, (ii) we want to go by EDF (earliest deadline first) order because the ones ending soon are more critical, and finally (iii) we prefer to service the requests which are hit by the solution $A_t^\star$ because we can directly pay for these service costs. Interestingly, we show that we can simultaneously take care of all three requirements. Moreover, we can identify a weight w such that we can take care of *all* outstanding requests which are cheaper than w and are not hit by $A_t^\star$.

3.1. The online algorithm for nonoverlapping requests. In this section, we assume that no two request intervals for the same page overlap—that is, for any pair of requests I, I' for the same page, $I \cap I' = \varnothing$. This gives a simpler algorithm than for the general case, which follows the same approach but has to deal with the case that multiple request intervals for the same page may try to charge to the same star in $A_t^\star$. (See section 3.2 for the online algorithm for the general case and section 4 for the offline algorithm).

Algorithm 1 (see Figure 3) shows how to convert an online solution $A_t^\star$ to (IP) into a feasible solution to the underlying PageTW instance. At each time t, we begin with some pages $C(t)$ in the cache. If the unique request I_t ending at time t is not already satisfied, and the cache is full, we evict the cheapest page $p_{\min}$ in the cache. We then potentially serve some other pending requests by bringing in and then evicting them, and also potentially remove some other pages from the current cache. (These services and removals help pay for evicting $p_{\min}$.)

Specifically, for every page p in $C(t)$, define I_t^p to be the most recent request for p which ends before t—this is well-defined because requests don't overlap. Define $Z^\star$ to be the pages in $C(t)$ for which the interval $\mathsf{Dext}(I_t^p, t)$ is hit by $A_t^\star$. (Since $\mathsf{Dext}(I_t^p, t)$ ends at time t, this only requires the knowledge of stars in $A_t^\star$ at or before time t.) We can directly pay for evicting these pages from the cache. But the situation is tricky—some $\mathsf{Dext}(I_{t'}^p, t')$ for a future time t' may also be hit by the same star in $A_t^\star$.

Algorithm 1: ConvertOnline(Online (IP) solution $A_t^\star$)

foreach $t = 0, 1, \ldots$ **do**
  let I_t be the interval with deadline t, and let $p_t \leftarrow \mathsf{page}(I_t)$
  if *cache $C(t)$ is full and I_t not satisfied* **then**
    evict the least-weight page $p_{\min}$ in $C(t)$
    if $w(p_t) \le 2\, w(p_{\min})$ **then**
      $Z^\star \leftarrow \varnothing$.
      for *every page p in $C(t)$* **do**
        $I_t^p \leftarrow$ the request interval I with $\mathsf{page}(I) = p$ and largest ending time $e(I) < t$.
        if $\mathsf{Dext}(I_t^p, t)$ is hit by $A_t^\star$ then add p to $Z^\star$.
      $U \leftarrow$ unsatisfied request intervals active at time t (one per page, page requests are disjoint).
      $U^\circ \leftarrow \{I \in U \mid \nexists t' \in I$ with $(\mathsf{page}(I), t') \in A_t^\star\}$ be intervals in U **not** hit by $A_t^\star$
      serve and **evict** all requests in $U \setminus U^\circ$.
      let $U^\circ_{\le w}$ and $Z^\star_{\le w}$ denote pages in U° and $Z^\star$ respectively with weight at most w.
      let $p^\star$ be a page in $Z^\star$ such that $w(U^\circ_{\le 2w(p^\star)}) \le 2 \cdot w(Z^\star_{\le w(p^\star)})$.
      evict all pages in $Z^\star_{\le w(p^\star)}$.
      serve and **evict** all requests in $U^\circ_{\le 2w(p^\star)}$.
  if I_t not satisfied **then** bring page p_t into cache.

FIG. 3. *Online algorithm to service request intervals.*

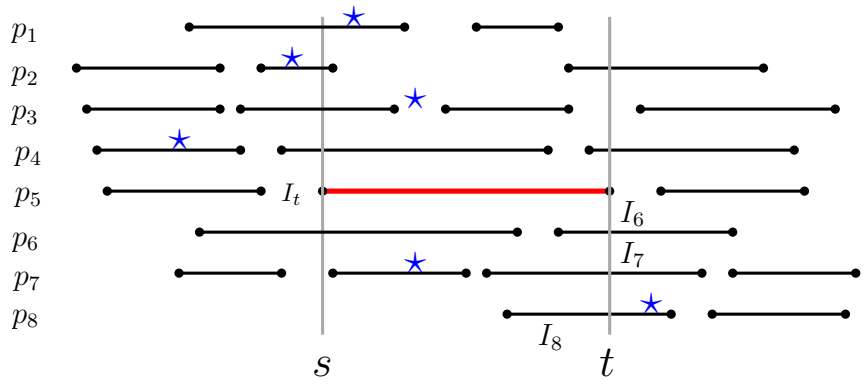

FIG. 4. *Illustration of the definitions used in Algorithm* 1. *The request intervals for each page are shown in one horizontal line (these do not overlap in our example). Focus on time t: the cache has pages $C(t) = \{p_1, \ldots, p_4\}$. The page $p_5 = \mathsf{page}(I_5)$ is critical at time t, with critical interval $I_t = [s, t]$. The solution $A^\star$ is given by the stars, each of which corresponds to a star (p, t) in the natural manner. The set $Z^\star = \{p_1, p_2, p_3\}$ and assuming I_6, I_7, I_8 are unsatisfied at time t, the set $U^\circ = \{I_6, I_7\}$.*

So we evict a subset of $Z^\star$—ones for which we are sure that the corresponding stars of $A^\star_t$ won't be charged again in the future.

To do this, the first simple observation is that we need to do this charging only when the critical page is not much heavier than the cheapest page in the cache; otherwise we can charge the eviction to this much heavier page in the cache. We define U to be the set of outstanding requests at time t and U° to be the subset of U which are not hit by $A^\star_t$ (lines 10–11)—by the sparsity property, these are the pages p for which $A^\star_t$ does not currently have a star beyond time t. We service all the requests in $U \setminus U^\circ$ immediately (we service a request by loading the corresponding page in the cache and "evict a request" by evicting the corresponding page from the cache)—these request intervals are hit by $A^\star_t$ and can be directly paid for (because of the nonoverlapping intervals). It is trickier to decide which requests in U° to service. In Lemma 3.4 we show there is a page $p^\star$ in $Z^\star$ such that $w(U^\circ_{\le 2w(p^\star)}) \le 2 \cdot w(Z^\star_{\le w(p^\star)})$, where the notation $X_{\le a}$ denotes all the pages in X of weight at most a. We service all the requests in $U^\circ_{\le 2w(p^\star)}$ and evict the pages in $Z^\star_{\le w(p^\star)}$. This ensures that all the remaining unsatisfied requests are much heavier than the current pages remaining in the cache. Finally, we show that the stars in $A^\star_t$ which are being charged for the eviction of $Z^\star_{\le w(p^\star)}$ are not going to be charged again (Lemma 3.7).

Finally, we serve I_t by bringing $p_t = \mathsf{page}(I_t)$ into the cache if still needed. Observe that the cache $C(t+1)$ at the start of time $t+1$ is contained within $C(t) \cup \{p_t\}$, since all other pages we satisfy at time t are also evicted. Moreover, if $C(t)$ was full, the cheapest page in $C(t)$ is evicted, and other pages from $C(t)$ may be evicted too (see Figure 4 for an example).

3.1.1. The analysis. We first need some supporting claims to show that the algorithm is well-defined, and then bound the cost.

CLAIM 3.2. *Suppose a page p is evicted from the cache at time t_1 but is in the cache at the end of time $t_2 > t_1$. Then there must exist a request interval I for page p with $t_1 < s(I) \le e(I) \le t_2$.*

Proof. The only step in the algorithm when a page is brought into the cache (and not evicted immediately afterward) is line 17. Hence, there must be an unsatisfied request interval I ending at some time $e(I) \le t_2$ which brought in (and kept) p in the cache after it had been evicted at time t_1. If $s(I) \le t_1$, then I would have been satisfied at time t_1, which is not true. ∎

We now show that the algorithm is well-defined.

CLAIM 3.3. *The set $Z^\star$ defined in lines 6–9 is nonempty.*

Proof. For each page $p \in C(t)$ (including the page $p_{\min}$), let I_t^p be the request interval defined in line 8—such a request interval exists because of Claim 3.2 (we assume that the cache is empty initially). Applying the IP constraint (D1) to time t and these k request intervals implies that at least one of their doubly extended intervals is hit by $A_t^\star$, and hence the corresponding page belongs to $Z^\star$. ∎

LEMMA 3.4. *There exists a page $p^\star \in Z^\star$ such that*

$$w(U^\circ_{\le 2w(p^\star)}) \le 2\, w(Z^\star_{\le w(p^\star)}).$$

Proof. We first claim that $|U^\circ| \le |Z^\star|$. Indeed, define a set of $k+1$ request intervals as follows. For each page $p \in C(t) \setminus Z^\star$, consider the request interval I_t^p for page p as defined in line 8. Since $p \notin Z^\star$, we must have $(p, t') \notin A_t^\star$ for all times $t' \in \mathsf{Dext}(I_t^p, t)$. But since the interval ends before t, we get $\mathsf{Rext}(I_t^p, t) \subseteq \mathsf{Dext}(I_t^p, t)$ and so $A_t^\star$ does not hit the right-extended interval for I_t^p either. To this collection of $k - |Z^\star|$ intervals, add the request intervals corresponding to U°—all these request intervals contain t, so the right-extension operation does not extend them. Moreover, we have at most one interval per page, and they are all unsatisfied, so the collection now has $|U^\circ| + k - |Z^\star|$ many intervals for distinct pages. And none of their right extensions are hit by $A_t^\star$, so by constraint (R1) this collection has size at most k. This proves that $|U^\circ| \le |Z^\star|$.

Let A be the set of pages in $Z^\star$ and B the set of pages in U°. We set up a bipartite graph on (A, B) with an edge between $p \in A$ and $p' \in B$ if $w(p') \le 2\, w(p)$. If this graph has a perfect matching, then $w(B) \le 2\, w(A)$. We choose $p^\star$ to be the highest-weight page in $Z^\star$.

Otherwise such a perfect matching does not exist. Let $A' \subseteq A$ be a minimal Hall set and B' be the neighborhood of A'. Let a be any page in A'. The pages in $A' \setminus \{a\}$ can be matched with B'. Therefore, $w(B') \le 2\, w(A' \setminus \{a\}) \le 2w(A')$. Now choose $p^\star$ to be the highest-weight page in A', to get $w(U^\circ_{\le 2w(p^\star)}) \le 2\, w(Z^\star_{\le w(p^\star)})$. ∎

Therefore when we reach line 14, a page $p^\star$ of the desired form exists, and the algorithm is well-defined. Finally the next claim shows that the request interval I_t gets served.

CLAIM 3.5. *Suppose I_t has not been satisfied before time t. Then the request for page p_t is satisfied at time t in Algorithm* 1.

Proof. We evict the page $p_{\min}$ from the cache in line 4. Till line 17, we do not retain any other page in the cache. Therefore, there is a vacancy in the cache at the beginning of line 17, and so we can bring the page p_t into the cache (if it is not in the cache already). ∎

3.1.2. The cost guarantee. We want to bound the total cost incurred till time T. The high-level cost analysis goes as follows. If the cache has room we can just satisfy I_t, so suppose the cache is full and we need to pay to evict $p_{\min}$. If the page p_t

is twice as heavy as $p_{\min}$, we can charge $p_{\min}$ to p_t and pay when p_t is subsequently evicted. Otherwise, if the unsatisfied intervals crossing time t which are hit by $A_t^\star$ have large weight, i.e., if $w(U \setminus U^\circ) \geq w(p_{\min})$, we can serve and evict them and then charge to them—this can pay for $p_{\min}$. Finally, we evict some pages from the current cache that are hit by $A_t^\star$: they pay for both evicting $p_{\min}$ and for serving some more of the outstanding requests. These pages are evicted from the cache to ensure they are not charged again.

We now show how to pay for the possible evictions in lines 4, 12, and 15–16. Recall there are two cases: $w(p_t) > 2w(p_{\min})$ (in which case the only eviction is that of $p_{\min}$) and $w(p_t) \leq 2w(p_{\min})$ (in which case there are multiple evictions). We handle the first case by amortization, namely we make the page p_t responsible for the cost of evicting $p_{\min}$. Formally, we introduce an accounting mechanism using a "load" on every page. Initially, the load on every page is 0. If $w(p_t) > 2w(p_{\min})$, then the load on $p_{\min}$ is transferred to p_t; additionally, p_t incurs a "load" of $w(p_{\min})$ to account for the eviction cost of $p_{\min}$. On the other hand, if $w(p_t) \leq 2w(p_{\min})$, i.e., we are in the second case, then the "load" on $p_{\min}$ and all other pages evicted from the cache is reset to 0. In this case, the eviction cost of these pages, and their respective loads, will be directly charged to the IP solution.

The following invariants on "load" are maintained by this accounting framework.

LEMMA 3.6. *The "load" of pages satisfies the following invariants:*
(a) *The load on any page outside the cache is* 0.
(b) *The load on any page in the cache is at most its weight.*

Proof. The lemma holds by induction over time. Note that whenever a page is evicted, its load becomes 0 in both cases; this preserves the first invariant. For the second invariant, the only way that a page gains load is when $p_{\min}$ is evicted and p_t gains load because $w(p_t) > 2w(p_{\min})$. But, this can happen only once for a page before it gets evicted, namely at the end of its request interval. The total "load" that the page gains at this time is at most its own weight, because of the condition $w(p_t) > 2w(p_{\min})$, and because the second invariant holds inductively for $p_{\min}$. This establishes that the second invariant is also maintained by this accounting framework. ∎

At the end of the algorithm, the total load over all pages is at most the weight of the pages in the cache, which is at most the optimum cost by the second invariant in the above lemma. This adds one to the competitive ratio. So, it suffices to only account for the eviction cost of pages when $w(p_t) \leq 2w(p_{\min})$. (We will not explicitly account for the load on these evicted pages since by the second invariant above, it only doubles the cost of the evicting the page.)

First, we charge the cost of evicting $p_{\min}$. In this case, we note that at least one page in $Z^\star$ is evicted (by Claim 3.3 and Lemma 3.4). So, we can charge evicting $p_{\min}$ to the eviction of that page (since $p_{\min}$ is the minimum weight page in the cache). This leaves us to account for the eviction of pages in $Z^\star$ and serving the requests in U.

First, we consider the requests in $U \setminus U^\circ$. Since each interval in $U \setminus U^\circ$, say, for page p, contains some star at (p, t') in $A_t^\star$, we charge the page to $w(A_T^\star)$ using these stars.

Finally, we charge the eviction cost for lines 15–16. This cost is $O(w(Z^\star_{\leq w(p^\star)}))$ by our choice of $p^\star$ in line 14. Observe that for each page p in $Z^\star_{\leq w(p^\star)}$, the doubly extended interval $\mathsf{Dext}(I_t^p, t)$ is hit by a star at $(\mathsf{page}(I), t') \in A_t^\star$ for some $t' \leq t$. We charge page p to this star of $A_t^\star$.

We finally show that no star of $A_T^\star$ can be charged twice in this manner.

LEMMA 3.7. *No star in $A^\star_T$ can be charged twice because of evictions in lines* 12*,* 15*, and* 16*.*

Proof. We first consider the evictions in line 12. If page p is evicted in line 12 at time t because of a star (p, t'), then (p, t') lies in the request interval for p containing t. Moreover, (p, t') cannot lie in any other request interval for p (recall that by our simplifying assumption, the request intervals are disjoint) and will not be charged by line 12 again.

Now, we consider the evictions in lines 15 and 16. For contradiction, suppose a star at $(q, t_q) \in A^\star_T$ is charged twice, at time t_1 and time t_2. Hence, at both these times q was in the cache and was evicted in line 15, so all unsatisfied request intervals that were active at these times and had weight $\leq 2w(q)$ were definitely served by line 16. (We will contradict this implication of our assumption.)

Let $I^q_{t_1}$ and $I^q_{t_2}$ be the corresponding intervals defined in line 8 for the page q. Claim 3.2 shows that $I^q_{t_2}$ starts after t_1. But we know that $t_q \leq t_1$, since (q, t_q) was charged at time t_1, and so $t_q \notin I^q_{t_2}$. So, in order for (q, t_q) to hit the doubly extended interval $\mathsf{Dext}(I^q_{t_2}, t_2)$, it must be the case that the critical interval I_{t_2} contained the time t_q (and hence time $t_1 \in [t_q, t_2]$). Let p_{t_2} denote $\mathsf{page}(I_{t_2})$. Then $w(p_{t_2}) \leq 2w(q)$; otherwise we would merely have evicted the cheapest page at time t_2 and not reached lines 15–16 again. This means p_{t_2} had weight at most $2w(q)$, and the request I_{t_2} was active and remained unsatisfied at the end of time t_1, which contradicts the implication above. ∎

We now summarize the overall eviction cost of the algorithm.

LEMMA 3.8. *For nonoverlapping requests, the overall eviction cost of the algorithm is at most $O(1)$ times $w(A^\star_T)$.*

Proof. We first consider the eviction cost of the page $p_{\min}$ in line 4. If $w(p_t) > 2w(p_{\min})$, then we have shown that the total eviction cost of $p_{\min}$ at such times t is at most the total eviction cost incurred by the algorithm in other steps plus the weight of all the pages. Thus, it is enough to consider only those times t when $w(p_t) \leq 2w(p_{\min})$. The total cost of evicting $U \setminus U^o$ during line 12 is at most $w(A^\star_T)$, and that incurred during lines 15–16 is at most $3w(A^\star_T)$, by Lemma 3.7. Moreover, there is an additional $w(A^\star_T)$ for the eviction of $p_{\min}$ for such times t. Thus, the total eviction cost incurred during times t when $w(p_t) \leq 2w(p_{\min})$ is at most $5w(A^\star_T)$. This implies that the overall eviction cost of the algorithm is at most $11w(A^\star_T)$. ∎

This proves Theorem 3.1 (without losing the extra $\log n$ factor) in the case of nonoverlapping requests for any page p.

3.2. Online algorithm for the general setting. The algorithm from subsection 3.1 assumes the requests for a page are nonoverlapping. We now extend it to handle overlapping requests in an online fashion. To understand the difficulty of the general case, consider the example with a page p having request intervals $[t_1, t], [t_2, t], \cdots, [t_k, t]$, where $t_1 < t_2 < \cdots < t_k < t$. Suppose we have a star $(p, t) \in A^\star_{t_1}$. Consider a time $t' \in [t_1, t]$ when the algorithm reaches line 10. If any of these intervals is not satisfied at t', then they will get counted in $U \setminus U^\circ$, and so we will charge the star at (p, t) for servicing p at time t'. But this can happen for multiple values of t', and we have only one star in $A^\star$ to charge to. Moreover, we cannot say that we will take care of all these requests at the ending time t—since all the pages in the cache may be very expensive at that time. In the offline case (which appears in section 4), one can add a *reverse delete* step, where we look at all these times when we service some of these

Algorithm 2: ConvertOnline((IP) solution $A_t^\star$ appearing online)

1 **foreach** $t = 0, 1, \ldots$ **do**
2 **let** I_t be the interval with deadline t, and let $p_t \leftarrow \mathsf{page}(I_t)$
3 **if** *cache* $C(t)$ *is full and* I_t *not satisfied* **then**
4 **evict** the least-weight page $p_{\min}$ in $C(t)$
5 **if** $w(p_t) \le 2\, w(p_{\min})$ **then**
6 $Z^\star \leftarrow \varnothing$.
7 **for** *every page* p *in* $C(t)$ **do**
8 $I_t^p \leftarrow$ non-dominating request interval I with $\mathsf{page}(I) = p$ and largest ending time $e(I) < t$.
9 **if** $\mathsf{Dext}(I_t^p, t)$ is hit by $A_t^\star$ then add I_t^p to $Z^\star$.
10 $U \leftarrow$ unsatisfied request intervals active at time t (one per page, if there are multiple choose one with earliest deadline).
11 $U^\circ \leftarrow \{I \in U \mid \nexists t' \in I$ with $(\mathsf{page}(I), t') \in A_t^\star\}$ be intervals in U **not** hit by $A_t^\star$.
12 **if** $I_t \notin U^\circ$ **then**
13 $U_t^\star \leftarrow$ request intervals I in $(U \setminus U^\circ)$ which are hit by $A_t^\star$ at some time $\le t$.
14 **serve** and **evict** all requests in $U_t^\star$.
15 **evict** the cheapest page $p^\dagger$ in $Z^\star$ (this may be the same as $p_{\min}$)
16 **sort** intervals in $U \setminus (U^\circ \cup U_t^\star)$ with weights $\le 2w_{p^\dagger}$ in ascending order of end-times.
17 **serve** and **evict** a maximal prefix of these intervals with total weight at most $4w_{p^\dagger}$.
18 **let** $U^\circ_{\le w}$ and $Z^\star_{\le w}$ denote pages in U° and $Z^\star$ with weight at most w.
19 **let** $p^\star$ be a page in $Z^\star$ such that $w(U^\circ_{\le 2w(p^\star)}) \le 2 \cdot w(Z^\star_{\le w(p^\star)})$.
20 **evict** all pages in $Z^\star_{\le w(p^\star)}$.
21 **serve** and **evict** all requests in $U^\circ_{\le 2w(p^\star)}$.
22 **if** I_t not satisfied **then** bring page p_t into cache.

FIG. 5. *Online algorithm to service request intervals.*

requests, and realize that a subset of them would suffice. However, the online setting, which we discuss below, is more complicated.

Algorithm 2 (see Figure 5) gives the online algorithm—the lines changed from Algorithm 1 are highlighted. We call a request interval I *nondominating* if it does not contain another request interval for $page(I)$—we know whether I is nondominating only at time $e(I)$. Notice that the definition of I_t^p in line 8 looks only at nondominating intervals.

Since the request intervals for a particular page are no longer disjoint, we do not serve and evict all the intervals in $U \setminus U^\circ$ when we create space at time t (as Algorithm 1 would do in line 12). Instead we only serve the requests hit by $A_t^\star$ before time t and some small set of requests that are hit by $A_t^\star$ after time t. As shown in the example at the beginning of this section, serving all such requests may lead to an

unbounded amount of charging to a star in $A^\star_T$. These requests are considered in the earliest deadline order and their total weight is a constant times the weight of the cheapest page in $Z^\star$ (denoted by $p^\dagger$). It is also worth noting that we perform these steps only if I_t is hit by $A^\star_t$ (line 12).

Also, note that line 16 is the only place in the algorithm where we need to know the right end-point of an existing request for a page.

By Lemma 3.4 the page $p^\star$ in line 19 exists, and hence the algorithm is well-defined. For the correctness we need to show that each page is served. This follows from the same arguments as in the proof of Claim 3.5. It remains to estimate the total eviction cost of this algorithm.

3.2.1. The cost analysis. Consider the run of the algorithm until time T; we bound the total cost incurred until this time, in terms of $w(A^\star_T)$. For technical reasons, we assume that $A^\star_T$ contains at least one star for each page—otherwise, we add such a star. Since the optimum cost is at least $\sum_p w(p)$, this raises $w(A^\star_T)$ by at most the optimal cost, i.e., adds one to the competitive ratio. Observe that evictions can only happen on lines 4, 14–17, and 20–21. The first and the last of these can be dealt with as in subsection 3.1. Indeed, paying for $p_{\min}$ and its load is done by putting a load on p_t if $w(p_t) \geq 2w(p_{\min})$ or else at least one other page from $C(t)$ is evicted and charged for, and we can handle $p_{\min}$ by charging a constant factor more. The total cost incurred during lines 20–21 is at most $3w(A^\star_T)$, since the proof for Lemma 3.7 remains unchanged. Indeed, at each time t when we perform those evictions, we charge the stars in $A^\star_T$ which hit the intervals in $Z^\star_{\leq w(p^\star)}$, and these stars are never charged again due to Lemma 3.7.

It remains to bound the cost incurred during lines 14–17. Let $\mathfrak{T} \subseteq [T]$ contain the times when we reach those lines. Let $Z^\star_t$, U_t, U°_t, and $U^\star_t$ denote the corresponding sets at time t, and let $p^\dagger_t, p^\star_t$ denote the pages chosen in line 15 and line 19. The evictions in line 14 are easy to pay for—see the next claim.

CLAIM 3.9. $\sum_{t\in\mathfrak{T}} w(U^\star_t) \leq w(A^\star_T)$.

Proof. We charge the weight of evicting $p = \mathsf{page}(I)$ for some request interval I in $U^\star_t$ to the element $(p, t') \in A^\star_t$ where $t' \in [0, t] \cap I$. We claim that no element in $A^\star_T$ will get charged twice this way. Indeed, if we charge to (p, t') at time $t \geq t'$, we have satisfied all existing request intervals for page p containing the time t'—and no future requests can arrive that contain it. ∎

We introduce some more notation. Let $U^\dagger_t$ be the prefix of request intervals serviced in line 17. For a time $t \in \mathfrak{T}$, define the *effective cost* at time t to be $w(p^\dagger_t) + w(U^\dagger_t)$—this is the remaining cost incurred at time t in lines 15 and 17. For an interval $[a, b]$, let $A^\star_t[a, b]$ denote the set of (p, t') in $A^\star_t$ where $t' \in [a, b]$. Let $P^\star_t[a, b]$ be the set of pages corresponding to which there is at least one star in $A^\star_t[a, b]$. Note that $w(P^\star_t[a, b]) \leq w(A^\star_t[a, b])$ for any time t and interval $[a, b]$.

CLAIM 3.10. *Suppose times $t_1, t_2 \in \mathfrak{T}$ are such that $t_1 < t_2$ and I_{t_2} contains time t_1. Then, $w(p^\dagger_{t_1}) \leq \frac{1}{2} w(P^\star_{t_2}[t_1, t_2])$ and as a consequence, the effective cost at time t_1 is at most $\frac{5}{2} w(P^\star_{t_2}[t_1, t_2])$.*

Proof. By design, $w(U^\dagger_{t_1}) \leq 4w(p^\dagger_{t_1})$, so the effective cost at time t_1 is at most $5w(p^\dagger_{t_1})$. Thus it suffices to show $w(p^\dagger_{t_1}) < w(p_{t_2})/2 \leq w(P^\star_{t_2}[t_1, t_2])/2$. Since $t_2 \in \mathfrak{T}$, the interval I_{t_2} was not served at time t_1. The following are possible reasons:

1. The interval $I_{t_2} \in U^\circ_{t_1}$ and $w(p_{t_2}) > 2\, w(p^\star_{t_1})$. Since $w(p^\dagger_{t_1}) \leq w(p^\star_{t_1})$ by the choice of $p^\dagger_{t_1}$, so $w(p^\dagger_{t_1}) < \frac{1}{2} w(p_{t_2})$. Since I_{t_2} is hit by $A^\star_{t_2}$ (because $t_2 \in \mathfrak{T}$), the past-preserving property of the online solution $A^\star$ implies that there must be a

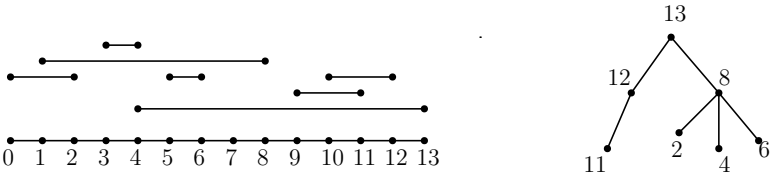

FIG. 6. *Illustration of F: the intervals on the left are I_t for $t \in \mathfrak{T}$ (note that these intervals are identified using their right end-points). The corresponding forest F is shown on the right.*

star at (p_{t_2}, t') in $A^\star_{t_2}$ for some $t' \in [t_1, t_2]$. Therefore, $w(p_{t_2}) \le w(P^\star_{t_2}[t_1, t_2])$. Thus, $w(p^\dagger_{t_1}) < \frac{1}{2} w(p_{t_2}) \le \frac{1}{2} w(P^\star_{t_2}[t_1, t_2])$.

2. $I_{t_2} \notin U^\circ_{t_1}$ but $w(p_{t_2}) > 2w(p^\dagger_{t_1})$, so it was not considered in the sorted ordering (in line 16). However, I_{t_2} was not in $U^\star_{t_1}$, so it was hit by $A^\star_{t_1}$ at some time after t_1—this means $w(p_{t_2})$ is counted in $w(P^\star_{t_1}[t_1, t_2])$. So $w(p^\dagger_{t_1}) \le \frac{1}{2} w(P^\star_{t_1}[t_1, t_2])$.
3. $I_{t_2} \notin U^\circ_{t_1}$ and $w(p_{t_2}) \le 2w(p^\dagger_{t_1})$ but we did not add I_{t_2} to $U^\dagger_{t_1}$ at time t_1, so $w(U^\dagger_{t_1})$ must have been more than $4\, w(p^\dagger_{t_1}) - w(p_{t_2}) \ge 2\, w(p^\dagger_{t_1})$. We added intervals to $U^\dagger_{t_1}$ in the EDF order, so all the weight added to $w(U^\dagger_{t_1})$ before considering I_{t_2} belongs to $w(A^\star_{t_1}[t_1, t_2])$. Chaining these inequalities, $w(p^\dagger_{t_1}) \le \frac{1}{2} w(U^\dagger_{t_1}) \le \frac{1}{2} w(P^\star_{t_1}[t_1, t_2])$.

Since $A^\star_{t_1} \subseteq A^\star_{t_2}$, we get the desired result. ∎

CLAIM 3.11. *Suppose $t_1, t_2 \in \mathfrak{T}$ are such that $t_1 < t_2$, and I_{t_2}* does not *contain t_1. Suppose $p^\dagger_{t_1} = p^\dagger_{t_2}$—call this page $p^\dagger$; then there is a star for $(p^\dagger, t')$ in $A^\star_{t_2}$ for some $t' \in (t_1, t_2]$.*

Proof. By Claim 3.2, since page $p^\dagger$ is evicted at time t_1, it must have been brought in by a request I that starts after t_1 and ends before t_2 (since $p^\dagger$ is in the cache at time t_2). We claim that $I^{p^\dagger}_{t_2}$ is also contained in $(t_1, t_2]$. Suppose not. So $s(I^{p^\dagger}_{t_2}) \le t_1$.

The interval I either is itself nondominating or contains a nondominating request for $p^\dagger$. In either case, there is a nondominating request interval for $p^\dagger$ which is contained in $(t_1, t_2]$—call this I' (it could be same as I). Now, I' is not designated as $I^{p^\dagger}_{t_2}$. It must be the case that $t(I^{p^\dagger}_{t_2}) \ge t(I')$. But then $I^{p^\dagger}_{t_2}$ contains I', which contradicts the fact that it is nondominating.

Since I_{t_2} is also contained in $(t_1, t_2]$, we see that $\mathsf{Dext}(I^{p^\dagger}_{t_2}, t_2)$ is also contained in $(t_1, t_2]$. Since $p^\dagger \in Z^\star$ at time t_2, $\mathsf{Dext}(I^{p^\dagger}_{t_2}, t_2)$ is hit by $A^\star_{t_2}$. This proves the claim. ∎

The charging forest. Motivated by Claims 3.10 and 3.11, we define a directed forest $F = (\mathfrak{T}, E)$ as follows. For time $t \in \mathfrak{T}$, if time t' is the smallest time such that $t' > t$ and the critical interval $I_{t'}$ for t' contains t, we define t' to be the parent of t, i.e., we add an arc (t', t). If no such time $t' > t$ exists, then t has no parent (i.e., zero in-degree). The following lemma gives some natural properties of the forest F.

LEMMA 3.12. *Suppose $t, t' \in \mathfrak{T}$ and $t < t'$.*

(a) *If $I_{t'}$ contains t, then t' is an ancestor of t in F.*
(b) *If t' is not an ancestor of t in F, then any node t'' in the subtree rooted at t' satisfies $t'' > t$.*

Proof. The first property follows by a simple induction, which we omit. For the second property, suppose for a contradiction that $t'' < t$, and let $t' = t_0, t_1, \ldots, t_r = t''$ be the path from t' to t'' in the forest F. Since $t' > t$, there is an i for which $t \in [t_{i+1}, t_i]$.

Since the interval $I_{t_{i+1}}$ contains t_i, it also contains t. But then, t_{i+1} should be an ancestor of t by property (a), which gives a contradiction. ∎

We now divide pages and times into classes. For each class c, we will consider a subforest F_c of F. We say that a page p is of *class* c if $w(p)$ lies in the range $[2^c, 2^{c+1})$. We say that a node t in the charging forest F is of class c if the corresponding page $p_t^\dagger$ is of class c. Let V^c be the vertices of class c in F. Let F^c be the minimal subgraph of F which preserves the connectivity between V^c (as in F). So the leaves of F^c belong to V^c, but there could be internal vertices belonging to other classes. We now show how to account for the cost incurred for the vertices in V^c. Let $A_T^\star(c)$ be the total weight of the stars in $A_T^\star$ corresponding to pages of class c. We say that a node in F^c is a *lone-child* if it is the only child of its parent.

CLAIM 3.13. *The total effective cost incurred during the leaf nodes in F^c and the internal nodes of class c in F^c which are not lone-children is $O(A_T^\star(c))$.*

Proof. The effective cost incurred during each time of class c is a constant times 2^c. Since the number of internal nodes which are not lone-children is bounded above by the number of leaf nodes, it is enough to bound the effective cost incurred at the leaf nodes. For a page p of class c, let $F^c(p)$ be the leaf nodes t in F^c for which $p_t^\dagger = p$. Let the times in $F^c(p)$ in increasing order be $t_1, t_2, \ldots, t_k$. Note that I_{t_i} does not contain t_{i-1} for $i = 2, \ldots, k$—otherwise t_i would be an ancestor of t_{i-1} (Lemma 3.12).

Claim 3.11 now implies that $A_T^\star$ contains a star for page p during $(t_{i-1}, t_i]$. Thus, the total effective cost incurred during $F^c(p)$ can be charged to the stars in $A_T^\star$ corresponding to page p (in case $k = 1$, we use the fact that $A_T^\star$ contains at least one star for p). Since all the leaf nodes in F^c belong to class c, the result follows. ∎

It remains to account for the times in V^c which have only one child in F^c.

CLAIM 3.14. *Let t_1 and t_2 be two distinct times of class c which are lone-child nodes in F^c. Let t_1' and t_2' be the parents of t_1 and t_2, respectively. Then the intervals $[t_1, t_1']$ and $[t_2, t_2']$ are internally disjoint.*

Proof. Suppose not. Say $t_1 < t_2 \le t_1'$. First assume $t_2' > t_1'$. Then $I_{t_2'}$ contains t_1' and so t_2' must be an ancestor of t_1'. If t_1' is the same as t_2, then the result follows easily; otherwise t_1' is a descendant of t_2 (since t_2' has only one child). But then $t_1' < t_2$, a contradiction.

The other case happens when $t_2' < t_1'$. In this case $I_{t_1'}$ contains t_2' (since it contains t_1 and $t_1 < t_2'$). If $t_1 = t_2'$, the result again follows trivially. Otherwise t_2' is a descendant of t_1, a contradiction. ∎

The above claim along with Claims 3.10 and 3.13 shows that the total cost incurred by times of class c can be charged to $w(A_T^\star)$. Thus, if there are K different classes, we get $O(K)$ approximation. To convert this into $O(\log n)$ approximation, we observe the following refinement of Claim 3.10. For a class c, times $t_1 < t_2$, let $A_T^\star(c, [t_1, t_2])$ be the stars of $A_T^\star[t_1, t_2]$ which are of class c.

CLAIM 3.15. *Suppose times $t_1, t_2 \in \mathfrak{T}$ are such that $t_1 < t_2$ and I_{t_2} contains time t_1. Let $p_{t_1}^\dagger$ be of class c. Then the effective cost at time t_1 is at most*

$$5\left(\sum_{c'=c-\lg n}^{c} w(A_T^\star(c', [t_1, t_2])) + \sum_{c'>c} \frac{w(A_T^\star(c', [t_1, t_2]))}{2^{c'-c}}\right).$$

Proof. Note that the total effective cost at time t is at most $5w(p_{t_1}^\dagger)$. Thus, we need to show that the expression in parentheses above is at least $w(p_{t_1}^\dagger)$. We have

two cases. First, suppose $P^\star_T[t_1, t_2]$ contains at least one page p of class $c' > c$. In this case, $w(p^\dagger_{t_1}) = \frac{w(p)}{2^{c'-c}}$ and therefore, the second term is at least $w(p^\dagger_{t_1})$. In the second case, all pages in $P^\star_T[t_1, t_2]$ are of class c or lower. Now, Claim 3.10 shows that $w(P^\star_T[t_1, t_2]) \geq 2w(p^\dagger_{t_1})$. Let P' be the pages of weight at most $w(p^\dagger)/n$ in $P^\star_T[t_1, t_2]$—the total weight of these pages is at most $w(p^\dagger)$. Thus the pages of class $c - \lg n$ and higher contribute at least half of $w(P^\star_T[t_1, t_2])$, i.e., they contribute at least $w(p^\dagger_{t_1})$. Since all pages in $P^\star_T[t_1, t_2]$ are of class c or lower, it follows that the first term is at least $w(p^\dagger_{t_1})$. This completes the proof. ∎

Claims 3.13 and 3.14 along with Claim 3.15 imply that the total cost incurred during times of class c is a constant times

$$\sum_{c'=c-\lg n}^{c} w(A^\star_T(c')) + \sum_{c'>c} \frac{w(A^\star_T(c'))}{2^{c'-c}}.$$

Summing over all classes yields Theorem 3.1.

4. Offline algorithm for the PageTW and PageTWPenalties problems. In section 3, we gave an online algorithm for PageTW and PageTWPenalties using online (integer) solutions to (IP) and (IPp). We shall now prove the following offline version of Theorem 3.1.

THEOREM 4.1. *There is a polynomial time algorithm that converts an α-approximate integral solution to* (IPp) *into a solution for the PageTWPenalties instance and has approximation ratio of $O(\alpha)$. As a consequence, there is a polynomial time algorithm that converts an α-approximate integral solution to* (IP) *into a solution for the PageTW instance and has approximation ratio of $O(\alpha)$.*

As in Theorem 3.1, we will actually prove the above theorem for the more restricted PageTW problem. This is sufficient for the more general PageTWPenalties problem as well, by the same reduction as the one we used in Theorem 3.1. Namely, the requests that are satisfied by the integer solution to an instance of PageTWPenalties are used to create an (equivalent) instance of the PageTW problem, and then the above theorem for the PageTW problem is applied to this instance to derive a valid solution for the original PageTWPenalties instance.

In the offline setting, we can assume that a request interval for a page p does not contain another interval for the same page—otherwise we can remove the outer interval wlog. Let $A^\star$ be an integral solution to (IP), and we want to convert it to a feasible solution to the underlying PageTW instance. As discussed in section 3, this will be done by adding a reverse delete step to Algorithm 1 (which considered the special case when all the request intervals for a particular page were mutually disjoint).

The algorithm is shown in Algorithm 3 (Figure 7). The first part of the algorithm until line 3 is same as in Algorithm 1. In line 10, if there are multiple active requests for a page p at time t, then we add the one with the earliest deadline to U. However, we cannot pay for all the evictions in line 12. Therefore, we *remove* some of these evictions in lines 18–23. We describe the details of this process now. We use Sat to denote the set of requests serviced during line 3. Let Sat_p be the requests in Sat that correspond to p and T_p be the times at which they are served. Let Sat'_p be any maximal collection of disjoint intervals in Sat_p. Note that since every interval in Sat'_p is hit by a distinct element of $A^\star$, we can pay for the service of Sat'_p. We define T'_p to be the time instances in T_p which are closest on each side to the end-points of the intervals in Sat'_p. In other words, if an interval $[s, t]$ is in Sat'_p, then we add the

Algorithm 3: ConvertOffline(IP solution $A^\star$)

foreach $t = 0, 1, \ldots$ **do**
  let I_t be the interval with deadline t, and let $p_t \leftarrow \mathsf{page}(I_t)$
  if *cache* $C(t)$ *full and* I_t *not satisfied* **then**
    evict the least-weight page $p_{\min}$ in $C(t)$
    if $w(p_t) \le 2\, w(p_{\min})$ **then**
      $Z^\star \leftarrow \varnothing$.
      for *every page* p *in* $C(t)$ **do**
        $I_t^p \leftarrow$ the request interval I with $\mathsf{page}(I) = p$ and largest ending time $e(I) < t$.
        if $\mathsf{Dext}(I_t^p, t)$ is hit by $A^\star$ then add p to $Z^\star$.
      $U \leftarrow$ unsatisfied request intervals active at time t (one per page).
      $U^\circ \leftarrow \{I \in U \mid \nexists t' \in I$ with $(\mathsf{page}(I), t') \in A^\star\}$ be intervals in U **not** hit by $A^\star$
      serve and **evict** all requests in $U \setminus U^\circ$.
      let $U^\circ_{\le w}$ and $Z^*_{\le w}$ denote pages in U° and $Z^\star$ respectively with weight at most w.
      let $p^\star$ be a page in $Z^\star$ such that $w(U^\circ_{\le 2w(p^\star)}) \le 2 \cdot w(Z^\star_{\le w(p^\star)})$.
      evict all pages in $Z^\star_{\le w(p^\star)}$.
      serve and **evict** all requests in $U^\circ_{\le 2w(p^\star)}$.
  if I_t not satisfied **then** bring page p_t into cache.

$\mathsf{Sat} \leftarrow$ set of requests serviced in line 12
for *every page* p **do**
  $T_p \leftarrow$ set of times when request intervals in Sat for page p are serviced (in line 12)
  $\mathsf{Sat}'_p \leftarrow$ maximal disjoint collection of request intervals for page p in Sat.
  $T'_p \leftarrow$ set of times in T_p closest (on either side) to the two end-points of intervals in Sat'_p
  cancel all movements of p into cache at times in $T_p \setminus T'_p$ (during line 12).

FIG. 7. *Offline algorithm to service request intervals in the general case.*

following time-steps to T'_p: (a) time-steps $s^*, s^{**} \in T_p$ such that there is no time-step in T_p that is after s^* but before s, or after s and before s^{**}, and (b) similarly, time-steps $t^*, t^{**} \in T_p$ such that there is no time-step in T_p that is after t^* but before t, or after t and before t^{**}. Clearly, $|T'_p| \le 4|\mathsf{Sat}'_p|$. It is not difficult to show that each interval in Sat_p has nonempty intersection with T'_p, and so it suffices to service p only during the times in T'_p. This is why the algorithm is correct and services all requests; we prove these facts formally below.

For the analysis, we again give some supporting claims to show that the algorithm is well-defined, and then bound the cost. The proofs of Claims 3.2–3.3 and Lemma 3.4 remain unchanged. We restate these here for the sake of completeness.

CLAIM 4.2. *Suppose a page p is evicted from the cache at time t_1 but is in the cache at the end of time $t_2 > t_1$. Then there must exist a request interval I for page p*

with $t_1 < s(I) \le e(I) \le t_2$.

CLAIM 4.3. *The set* $Z^\star$ *defined in lines* 6–9 *is nonempty.*

LEMMA 4.4. *There exists a page* $p^\star \in Z^\star$ *such that*

$$w(U^\circ_{\le 2w(p^\star)}) \le 2\, w(Z^\star_{\le 2w(p^\star)}).$$

Lemma 4.4 shows the existence of page $p^\star$ in line 3. The proof of the following claim is the same as that of Claim 3.5.

CLAIM 4.5. *Let* I_t *be unsatisfied at time* t. *If* $w(p_t) \le 2w(p_{\min})$, *then the page* p_t *belongs either to* $U \setminus U^\circ$ *in line* 12 *or to* $U^\circ_{\le 2w(p^\star)}$ *in line* 16 *and is served and evicted. Otherwise* p_t *is served by line* 17 *and remains in the cache.*

We now show that even after removing some of the services for a page p in lines 20–23, the algorithm services all the requests in Sat_p. In the claim below, we use the notation in lines 20–23.

CLAIM 4.6. *Every request interval in* Sat_p *has nonempty intersection with* T'_p.

Proof. Let I be a request interval in Sat_p. First assume that it lies in Sat'_p, and let $t \in T_p$ be the time at which it is serviced in line 12. Since t lies between $s(I)$ and $e(I)$, the closest time in T_p to the right of $s(I)$, call it t', must lie between $s(I)$ and t. Since $t' \in T'_p$, the result follows.

Now assume $I \notin \mathsf{Sat}'_p$. So there must be an interval $I' \in \mathsf{Sat}'_p$ which overlaps with I. Since any two requests for the same page are nonnested, I' contains either $s(I)$ or $t(I)$. Suppose it contains $s(I)$ (the other case is similar). Then $s(I') < s(I) < e(I') < e(I)$. Let t be the time at which I is serviced in line 12. Say t lies to the right of $e(I')$. Then the time in T_p which is closest to $e(I')$ on the right side, call it t', lies in T'_p. But $t' \in [e(I'), t]$ and so it lies in I. The case when t is to the left of $e(I')$ is similar—there will be a time in T'_p which lies in the interval $[s(I), e(I')]$ and so belongs to I as well. □

The above claim proves that the algorithm services all the request intervals.

We now analyze the cost incurred by the algorithm. The analysis is again very similar to that in section 3.1.2. Again, we introduce an accounting mechanism using a "load" on every page. Initially, the load on every page is 0. If $w(p_t) > 2w(p_{\min})$, then the load on $p_{\min}$ is transferred to p_t; additionally, p_t incurs a "load" of $w(p_{\min})$ to account for the eviction cost of $p_{\min}$. On the other hand, if $w(p_t) \le 2w(p_{\min})$, then the "load" on $p_{\min}$ and all other pages evicted from the cache are reset to 0. In this case, the eviction cost of these pages, and their respective loads, will be directly charged to the IP solution.

The following lemma has the same proof as Lemma 3.6.

LEMMA 4.7. *The "load" of pages satisfies the following invariants:*
(a) *The load on any page outside the cache is* 0.
(b) *The load on any page in the cache is at most its weight.*

At the end of the algorithm, the total load over all pages is at most the weight of the pages in the cache, which is at most the optimum cost by the second invariant in the above lemma. This adds one to the competitive ratio. So, it suffices to only account for the eviction cost of pages when $w(p_t) \le 2w(p_{\min})$. (As earlier, we will not explicitly account for the load on these evicted pages since by the second invariant above, it only doubles the cost of the evicting the page.)

First, we charge the cost of evicting $p_{\min}$. In this case, we note that at least one page in $Z^\star$ is evicted (by Claim 4.3 and Lemma 4.4). So, we can charge evicting $p_{\min}$

to the eviction of that page (since $p_{\min}$ is the minimum weight page in the cache). This leaves us to account for the eviction of pages in $Z^\star$ and serving the requests in U.

Now we consider the cost incurred during line 12. Because of lines 18–23, we do not pay for all of these request intervals. Instead, we have the following lemma.

LEMMA 4.8. *The total cost incurred in line* 12 *(after executing lines* 18–23*) is* $O(1)$ *times the cost of* $A^\star$.

Proof. Recall that we defined T'_p as the nearest time-steps in T_p before and after the start time $s(I)$ and end time $t(I)$ of each interval $I \in \mathsf{Sat}'_p$, where Sat_p is a maximal collection of disjoint intervals in Sat_p. This means that for a particular page p, we serve at most $4|\mathsf{Sat}'_p|$ requests for p in line 12. Since the requests in Sat'_p are disjoint and each of them is hit by $A^\star$, we can charge the service cost to the cost of $A^\star$. ∎

Finally, we charge the eviction cost for lines 15-16. This cost is $O(w(Z^\star_{\le w(p^\star)}))$ by our choice of $p^\star$ in line 14. Observe that for each page p in $Z^\star_{\le w(p^\star)}$, the doubly extended interval $\mathsf{Dext}(I^p_t, t)$ is hit by an element $(p, t') \in A^\star$, so we want to charge to this element of $A^\star$. Moreover, each page in $Z^\star_{\le w(p^\star)}$ is at least as heavy as $p_{\min}$, so any of these elements of $A^\star$ can pay to evict $p_{\min}$ (and its load).

We finally show that no element of $A^\star$ can be charged twice in this manner. The proof is identical to that of Lemma 3.7.

LEMMA 4.9. *No element in* $A^\star$ *can be charged twice because of evictions in lines* 15–16

This completes the proof of Theorem 4.1.

5. Solving the integer program (IPp) for PageTWPenalties. We now give algorithms to solve the integer program (IPp), in both the offline and online settings. The main challenge in the offline case is that the LP relaxation has an unbounded integrality gap, so just relaxing the integrality constraints and then rounding will not suffice. Instead, we write a compact IP that has a smaller gap, and also has fewer constraints. Let us consider an example problem, that of picking $n - k$ out of n items and the objective being the total weight of the picked items. All items have unit weight, so any feasible solution has cost at least $n - k$. If variable $x_i \in [0, 1]$ indicates that we should pick item i, we can write an integer linear constraint for every choice $\mathcal{C}$ of $k + 1$ items saying that $\sum_{i \in \mathcal{C}} x_i \ge 1$. But the LP relaxation of this IP admits the fractional solution where $x_i := \frac{1}{k+1}$ for all i, and hence total cost $\frac{n}{k+1} \ll n - k$, showing a large integrality gap. However, replacing these $\binom{n}{k+1}$ linear constraints by the compact form $\{\sum_i x_i \ge n - k, x \in [0, 1]^n\}$ gives a formulation having no integrality gap; we use analogous ideas to address both the challenges above. In the online setting, we need to solve and round the resulting LPs online, which will require us to refine the primal-dual algorithms of Bansal, Buchbinder, and Naor [BBN10].

We handle the constraints for the right and double extensions separately, in sections 5.1 and 5.2, respectively; this at most doubles the cost of the solution. Indeed, if we have two solutions, where the first one satisfies the constraints for right extensions and the second one satisfies those for double extensions, the maximum of these two solutions satisfies both sets of constraints. In both cases, we reduce to the following interval covering problem.

DEFINITION 5.1 (tiled interval cover). *In the* tiled interval cover *problem (TiledIC), for each page* $p \in [n]$*, we are given a collection* $\mathcal{I}_p$ *of disjoint intervals that partition the entire timeline. All intervals in* $\mathcal{I}_p$ *have the same weight* $w(p)$*. The goal is to select a minimum-weight subset of intervals from* $\mathcal{I} := \cup_p \mathcal{I}_p$ *such that for every time* t*, at*

least $n-k$ of these selected intervals contain t.

The offline algorithms to solve TiledIC will rely on total-unimodularity, and the online ones will reduce to primal-dual algorithms for the classical paging problem. The details of these solutions to TiledIC appear in Appendix D.

5.1. IP solution for right-extension constraints. In this section, the focus is only on the right-extension constraints, i.e., the following IP:

$$\text{(IP-Rp)} \qquad \min_{x,y \text{ Boolean}} \quad \sum_{p,t} w(p)\, x_{p,t} + \sum_{I} \ell(I)\, y_I,$$

$$\text{(R1p)} \qquad \sum_{I \in \mathcal{C}} \min\left(1, y_I + \sum_{t' \in \mathsf{Rext}(I,t)} x_{\mathsf{page}(I),t'}\right) \geq 1 \qquad \forall t\, \forall \mathcal{C},$$

where we again have $\mathcal{C}$ consisting of $k+1$ requests, each for a distinct page, and each starting before time t. The discussion about getting a compact IP above can be used to show that constraint (R1p) is equivalent (for integral solutions) to the following constraint:

$$\text{(R2p)} \qquad \sum_{I \in \mathcal{C}} \min\left(1, y_I + \sum_{t' \in \mathsf{Rext}(I,t)} x_{\mathsf{page}(I),t'}\right) \geq n-k \quad \forall t\, \forall \mathcal{C},$$

where $\mathcal{C}$ now consists of n requests, one for each of the pages, and each starting before time t.

We now show how to approximately solve (IP-Rp) using an algorithm for TiledIC. Consider an instance $\mathcal{I}$ of (IP-Rp). We assume that at time 0, there is a request interval $[0,0]$ with infinite penalty for every page; this only changes the optimum value by $\sum_p w(p)$ and therefore adds at most one to the competitive ratio. We now create an instance $\mathcal{I}'$ of TiledIC by creating a collection of intervals $\mathcal{K}_p$ for each page p using the procedure in Figure 9; each of these intervals will have weight $w(p)$ (see also Figure 8 for an example). Essentially each such interval is obtained by a minimal collection of original request intervals corresponding to p in $\mathcal{I}$ such that their total penalty becomes at least $w(p)$.

For the next two results, let $\mathcal{I}$ and $\mathcal{I}'$ be the instances as defined above.

LEMMA 5.2 (forward direction). *Consider an integral solution (x,y) to $\mathcal{I}$. Then there is a solution $\mathcal{S}$ to $\mathcal{I}'$ of cost at most $2(\sum_{p,t} w(p)x_{p,t} + \sum_I \ell(I)y_I)$.*

Proof. For an interval I' in $\mathcal{K}_p$, let $\mathtt{rt}(I')$ to be the interval in $\mathcal{K}_p$ which lies immediately to the right of I'. For every page p and interval $I' \in \mathcal{K}_p$, we add both I'

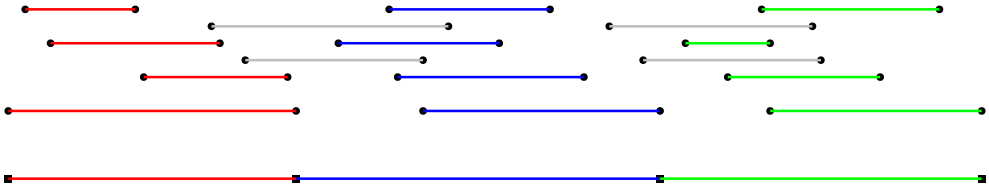

FIG. 8. *Constructing $\mathcal{K}_p$ for page p: assume each request interval has the same penalty $\ell(I)=1$, and $w(p)=4$. The request intervals in gray intersect with the previous intervals in $\mathcal{K}_p$, so we increase t until we see four nonintersecting intervals, and create the interval in $\mathcal{K}_p$ based on the end-point of this fourth interval (shown in the last row with end-points denoted by squares).*

```
Initialize K_p ← ∅, t* ← 0.
for t = 1, 2, ... do
    I'_p ← set of request intervals for p which are contained in [t*, t].
    if the total penalty of the intervals in I'_p becomes at least w(p) then
        Add [t*, t) to K_p.
        Update t* ← t.
```

FIG. 9. *The online procedure to construct the partition $\mathcal{K}_p$ for page p.*

and $\mathtt{rt}(I')$ to the solution $\mathcal{S}$ if either of these conditions is satisfied: (i) $x_{p,t} = 1$ for some time $t \in I'$, or (ii) $y_I = 1$ for every request interval $I \in \mathcal{I}$ for page p such that the interval I is contained within the interval I'. The cost guarantee is easy to see. We charge the cost of I' and $\mathtt{rt}(I')$ to $w(p)x_{p,t}$ in case (i) and to the total penalty of request intervals for page p contained within I' in case (ii).

Now to prove the feasibility of this solution $\mathcal{S}$: fix a time t. For each page p, let $I'(p) \in \mathcal{K}_p$ be the rightmost interval which ends before t. Classify the set of pages into two classes—let P_1 be the set of pages p such that every request interval $I \in \mathcal{I}$ for p which is contained within $I'(p)$ has $y_I = 1$; define $I(p)$ to be any of these request intervals. Let $P_2 := [n] \setminus P_1$ be the remaining set of pages, i.e., pages p such that there is at least one request interval for it (call this request interval $I(p)$) contained within $I'(p)$ such that $y_{I(p)} = 0$.

Let $\mathcal{C}$ be the collection of the intervals $I(p)$ defined above, one for each page p. Applying constraint (R2p) to $\mathcal{C}$, we get

$$|P_1| + \sum_{p \in P_2} \sum_{t' \in \mathsf{Rext}(I(p),t)} \min(1, x_{p,t'}) \geq n - k.$$

Hence, there is a set $P_2' \subseteq P_2$ of cardinality $n - k - |P_1|$ such that for any page $p \in P_2'$, there is a time $t' \in \mathsf{Rext}(I(p), t) \subseteq I'(p) \cup \mathtt{rt}(I'(p))$ with $x_{p,t'} = 1$. In either case (whether $t' \in I'(p)$ or $t' \in \mathtt{rt}(I'(p))$), it follows that we will pick the interval $\mathtt{rt}(I(p))$ in the solution $\mathcal{S}$. Moreover, the collection $\mathcal{S}$ contains the intervals $I(p)$ and $\mathtt{rt}(I(p))$ for each page $p \in P_1$. Since all these intervals $\mathtt{rt}(I(p))$ contain t, we have chosen $|P_1 \cup P_2'| \geq n - k$ intervals containing t, and hence $\mathcal{S}$ is a feasible solution to $\mathcal{I}'$. ∎

LEMMA 5.3 (reverse direction). *Let $\mathcal{S}$ be a integral solution to the instance $\mathcal{I}'$. Then there is a solution (x, y) to $\mathcal{I}$ of cost at most $3w(\mathcal{S})$.*

Proof. The solution (x, y) is as follows: for each page p and interval $I' = [t_1', t_2'] \in \mathcal{K}_p \cap \mathcal{S}$, (i) set $x_{p,t_1'} = x_{p,t_2'} = 1$, and also (ii) for every request interval $I = [t_1, t_2]$ for page p having $t_1' \leq t_1 \leq t_2 < t_2'$ (i.e., intervals contained within such a chosen interval I' and ending strictly earlier) set $y_I = 1$. Since t_2 is strictly smaller than t_2', the construction of $\mathcal{K}_p$ ensures that the total penalty cost of such intervals is at most $w(p)$. The cost guarantee for (x, y) follows immediately. It remains to show feasibility.

Fix a time t. For each page p, let $I'(p) \in \mathcal{K}_p$ be the interval containing t. By the feasibility of $\mathcal{S}$, there is a set $P_1 \subseteq \mathcal{S}$ of $n - k$ pages such that their corresponding intervals $I'(p)$ contain t. Consider some constraint (R1p) corresponding to a set $\mathcal{C}$ of requests for instance $\mathcal{I}$. For each page p, let $I(p) \in \mathcal{C}$ be the request interval for p. We know that $I(p)$ starts before time t, but it may end after time t. For a page $p \in P_1$, two cases arise: (i) $I(p)$ is strictly contained in $I'(p)$, or (ii) $\mathsf{Rext}(I(p), t)$

contains one of the two end-points of $I'(p)$. In the first case we must have set $y_{I(p)}$ to 1, whereas in the second case there is a time t' (which is one of the two end-points of $I'(p)$) in $\mathsf{Rext}(I(p), t)$ such that $x_{p,t'}$ is 1. Since $|P_1| = n - k$, we infer that (x, y) satisfies equation R1p. ∎

Combining these with Lemma D.1, we see that there is a 6-approximation offline algorithm for SolveRextP.

5.1.1. Implementing the solution online. For the *online setting*, we can construct the set $\mathcal{K}_p$ online, and also approximately solve the TiledIC problem in an online fashion using the algorithm from Appendix D. However, the translation given in Lemma 5.3 cannot be implemented online. Specifically, it sets the x variables for start and end times of the chosen intervals $I(p)$ and also y variables for all intervals strictly contained within $I(p)$, but (i) setting $x_{p,t}$ variables for times $t_1 < t$ and y_I variables for past intervals violates the past-preserving property required in section 3, and (ii) the online construction of $\mathcal{K}_p$ means the end time t_2 of the interval $I(p) \in \mathcal{K}_p$ is not known at time t.

The first issue is easy to fix: if we change the proof of Lemma 5.3 so that when interval $I'(p) = [t'_1, t'_2]$ is added to $\mathcal{S}$ at some time t, we set $x_{p,t} = 1$ instead of x_{p,t'_1}, and also we set $y_I = 1$ for interval I that are contained within $I'(p)$ only from now on. This change makes it past-preserving and maintains correctness. As for the second issue, that the online algorithm may not know the right end-point t'_2 of $I'(p) \in \mathcal{K}_p$ at time t, and hence cannot add the star in the future, there is at most one such "to-be-set" variable for each page p; moreover, all intervals for page p containing the current time t are already hit variables corresponding to past times, or by this single to-be-set variable. This satisfies the *sparsity* property of section 3.

Hence, combining the above reductions with the algorithmic results on solving TiledIC in Appendix D, and implementing these changes in the online setting, we get the following.

LEMMA 5.4 (right-extension algorithms). *There is an online $O(\log k)$-competitive algorithm to solve the right-extension constraints with penalties* (IP-Rp). *The solution satisfies the monotonicity, past-preserving, and sparsity properties required in section* 3. *Finally, there is an offline* 6*-approximation algorithm for this problem.*

5.2. IP solution for double extension constraints. We now want to solve the double-extension constraints (D1):

$$\text{(IP-D1p)} \qquad \min_{x,y \text{ Boolean}} \quad \sum_{p,t} w(p)\, x_{p,t} + \sum_{I} \ell(I)\, y_I,$$

$$\text{(D1p)} \qquad \sum_{I \in \mathcal{C}} \min\left(1, y_I + \sum_{t' \in \mathsf{Dext}(I,t)} x_{\mathsf{page}(I),t'}\right) \geq 1 - y_{I_t} \qquad \forall t\, \forall \mathcal{C},$$

where $\mathcal{C}$ consists of k requests, each for a distinct page (not equal to page p_t), and each request interval ending before time t. As in the previous section, we can replace (D1p) by the following constraints and get exactly the same integer solutions:

$$\text{(D2p)} \qquad \sum_{I \in \mathcal{C}} \min\left(1, y_I + \sum_{t' \in \mathsf{Dext}(I,t)} x_{\mathsf{page}(I),t'}\right) \geq (n-k)(1 - y_{I_t}) \quad \forall t\, \forall \mathcal{C}.$$

Here the set $\mathcal{C}$ consists of $n - 1$ requests, one for each distinct page different from page p_t, where each of these request intervals ends before time t. The variable y_I

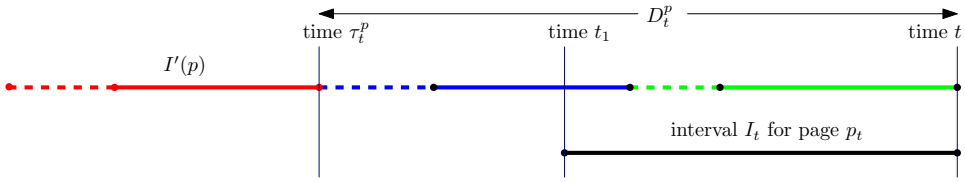

FIG. 10. *The definition of time* τ_t^p.

has two roles—the variable y_{I_t} on the right denotes whether we need to consider the constraints (D2p) corresponding to time t, whereas the ones on the left denote whether those requests were satisfied, or if their penalty is paid instead. It turns out that we can drop the occurrences of the y_I variables on the left and get a polynomial-sized covering IP instead.

For each page p, let $\mathcal{K}_p$ be as defined in subsection 5.1, and let $\mathcal{T}_p$ denote the right end-points of the intervals in $\mathcal{K}_p$. For time t and page p, define time τ_t^p as follows (see Figure 10): let $I_t = [t_1, t]$ be the request interval which ends at time t, and let p_t denote $\mathsf{page}(I_t)$. If $I'(p)$ is the last interval in $\mathcal{K}_p$ which ends before t_1, then τ_t^p is the right end-point of this interval $I'(p)$. (Of course, the time τ_t^p lies in the set $\mathcal{T}_p$.) Define the interval $D_t^p := [\tau_t^p, t]$.

Consider the following compact IP, which is equivalent to (IP-D1p) up to constant factors, as we show next:

$$\text{(IP-D3p)} \qquad \min_{x,y \text{ Boolean}} \quad \sum_{p,t} w(p)\, x_{p,t} + \sum_{I} \ell(I)\, y_I,$$

$$\text{(D3p)} \qquad \sum_{p: p \neq p_t} \min\Big(1, \sum_{t' \in D_t^p} x_{p,t'}\Big) \geq (n-k)(1-y_{I_t}) \qquad \forall t.$$

LEMMA 5.5. *Given an integer solution* (x, y) *to* (IP-D1p)*, there is a solution* (x', y') *to* (IP-D3p) *with cost at most three times as much. Conversely, if* (x', y') *is an integer solution to* (IP-D3p)*, there is a corresponding solution to* (IP-D1p) *of cost at most twice that of* (x', y').

Proof. For the first part, define $y'_I = y_I$ for all I, and $x'_{p,t} = x_{p,t}$ for all p and t. For an interval $I \in \mathcal{K}_p$, define $\mathtt{rt}(I)$ to be the interval immediately to its right in $\mathcal{K}_p$. For each page p and interval $I \in \mathcal{K}_p$, we perform the following steps: let t_1 and t_2 be the right end-points of I and $\mathtt{rt}(I)$, respectively. We set $x'_{p,t_1} = x'_{p,t_2} = 1$ if either of these conditions holds: (i) there is a time $t \in I$ such that $x_{p,t} = 1$, or (ii) $y_{I'} = 1$ for every request interval I' for p which is contained within I. The cost guarantee for (x', y') follows easily.

It remains to show that (x', y') is feasible, consider a time t, and assume $y'_{I_t} = y_{I_t} = 0$ (otherwise (D3p) follows immediately). Let $I_t = [t_0, t]$, and for page $p \neq p_t$, let $I'(p)$ be the last interval in $\mathcal{K}_p$ ending before t_0, so that τ_t^p is the right end-point of $I'(p)$. If there is a request interval I for page p which is contained in $I'(p)$ and $y_I = 0$, we define $I(p)$ to be such an interval I; otherwise $I(p)$ is any request interval for p contained within $I'(p)$. Let P_1 denote the pages for which the first case holds and P_2 be the second set of pages. Feasibility of (D1p) for these set of intervals implies that

$$|P_2| + \sum_{p \in P_1} \min\Big(1, \sum_{t' \in \mathsf{Dext}(I(p),t)} x_{p,t'}\Big) \geq n - k.$$

For pages $p \in P_2$, we would have set $x'_{p,\tau_t^p} = 1$ due to condition (ii). Furthermore, the interval $\mathsf{Dext}(I(p), t)$ contains D_t^p and is contained in $I'(p) \cup D_t^p$. Now for page $p \in P_1$, suppose $t' \in \mathsf{Dext}(I(p), t)$ is such that $x_{p,t'} = 1$. Then either $t' \in D_t^p$, in which case $x'_{p,t'} = x_{p,t'}$, or else $t' \in I'(p)$ and we would have set $x'_{p,\tau_t^p} = 1$ by condition (i). In either case, $\sum_{t' \in D_t^p} x_{p,t'} \geq 1$. This shows that (x', y') is a feasible solution to (IP-D3p).

We now show the converse. Let (x', y') be a feasible solution to (IP-D3p). We construct a feasible solution (x, y) to (IP-D1p). As above, we first set $x = x', y = y'$. Furthermore, for every (p, t) such that $x'_{p,t} = 1$, let $I = [t_1, t_2]$ be the interval in $\mathcal{K}_p$ containing t, and set $x_{p,t_2} = 1$. The cost guarantee for x follows easily. To show feasibility, fix a time t for which $y_{I_t} = 0$, and let P be the set of pages for which the left-hand side (LHS) equals 1 in constraint (D3p). For a page p, recall that $I'(p)$ is the interval in $\mathcal{K}_p$ which ended before I_t started, but $\mathtt{rt}(I'(p))$ intersects I_t. Now consider the constraint (D1p) for time t and set of pages $\mathcal{C}$. We claim that the LHS term for every page $p \in P$ equals 1. If $\mathsf{Dext}(I, t) \supseteq D_t^p$, then this claim follows from the fact that $p \in P$. So assume that $\mathsf{Dext}(I, t) \not\supseteq D_t^p$, and since both intervals share the same right end-point, $\mathsf{Dext}(I, t) \subseteq D_t^p$. But then, by the greedy construction process, $\mathtt{rt}(I'(p))$ is contained in D_t^p. Since $p \in P$ we know that $x'_{p,t'} = 1$ for some $t' \in D_t^p$ and hence t' is in $\mathtt{rt}(I'(p)) \cup I_t$. If $t' \in I_t$, then $t' \in \mathsf{Dext}(I, t)$ where $I \in \mathcal{C}$ is the request interval for p. Since $x_{p,t'} = 1$ as well, we see that the LHS term for p in (D1p) equals 1. On the other hand, if $t' \in \mathtt{rt}(I'(p))$, we will set $x_{p,t''}$ to 1, where t'' is the right end-point of $\mathtt{rt}(I'(p))$. Since $t'' \in I_t$, we have that $t'' \in \mathsf{Dext}(I, t)$, and so the same conclusion holds. ∎

The compact LP can clearly be solved offline. The following theorem, whose proof is deferred to the appendix, shows that the fractional relaxation of (IP-D3p) can also be solved online losing only a logarithmic factor.

THEOREM 5.6 (solving the LP online). *There is an $O(\log k)$-competitive online algorithm which maintains a solution to the fractional relaxation of* (IP-D3p). *At time t, the fractional solution only changes (and in fact increases) the variables $x_{p,t}$ for all pages p, and y_{I_t}.*

COROLLARY 5.7 (integral penalty variables). *Given an online fractional solution $(\tilde{x}, \tilde{y})$ to* (IP-D3p)*, we can maintain another online fractional solution $(\bar{x}, \bar{y})$ whose cost is at most twice of that of $(\tilde{x}, \tilde{y})$, such that $\bar{y}_I$ is integral for every I. This solution also has the same property that at time t, it only increases the variables $x_{p,t}$ and y_{I_t} corresponding to time t.*

Proof. Fix a time t and the corresponding solution $(\tilde{x}, \tilde{y})$. We set $\bar{y}_I = 1$ if $\tilde{y}_I > 1/2$ and to 0 otherwise; we also set $\bar{x}_{p,t} = \min(1, 2\tilde{x}_{p,t})$. This at most doubles the cost of the solution and maintains feasibility. Furthermore, at time t, $(\bar{x}, \bar{y})$ only increases the variables $\bar{x}_{p,t}$ for all pages p and $\bar{y}_{I_t}$. ∎

Let the problem defined by (IP-D3p) be called $\mathsf{SolveDextP}$, and fix an instance $\mathcal{I}$ of this problem. For the rest of the discussion, we maintain an online fractional solution $(\bar{x}, \bar{y})$ to $\mathcal{I}$ with the properties mentioned in Corollary 5.7. In order to maintain

an online integral solution to $\mathcal{I}$, we first solve the problem for nonnested instances, and then prove an "extension" theorem to translate from nonnested instances to all instances.

5.2.1. Solving globally nonnested cases of SolveDextP. We say that interval $I = [t_1, t_2]$ is *strictly nested* within $I' = [t_1', t_2']$ if $t_1' \le t_1 \le t_2 \le t_2'$, and either the first or the last inequality is strict. (We drop the use of strict, and simply say "nested" henceforth.) Two intervals are nested if one of them is nested within another. Let $\mathcal{T}'$ be some subset of the timeline $[T]$ such that for every $t_1 \neq t_2 \in \mathcal{T}'$, their critical intervals I_{t_1}, I_{t_2} are not nested. We define the NonNestDextP problem, which solves the problem (IP-D3p) where the constraints correspond to times $t \in \mathcal{T}'$ and we are also given that $\bar{y}_{I_t} = 0$ for all $t \in \mathcal{T}'$ (i.e., we are not paying the penalty at these times). First, we show that the intervals $D_t^p = [\tau_t^p, t]$ for any fixed page are also nonnested.

CLAIM 5.8. *For times $t, t' \in \mathcal{T}'$ and a page p such that $p \notin \{p_t, p_{t'}\}$, the intervals $D_t^p, D_{t'}^p$ are nonnested.*

Proof. Assume wlog that $t < t'$. Let $I_t = [t_1, t], I_{t'} = [t_1', t']$. By the nonnested property, $t_1 < t_1'$. Therefore, by construction, $\tau_t^p \le \tau_{t'}^p$. ∎

We reduce the instance NonNestDext to a tiled interval cover problem with the exclusions (TiledICEx) instance. TiledICEx is like TiledIC, where additionally for each time t we are specified a page p_t, and we cannot use the intervals in $\mathcal{I}_{p_t}$ for the coverage requirement at time t. In Appendix D we give a constant factor approximation algorithm and $O(\log k)$-competitive algorithm for TiledICEx.

The reduction of the instance $(\mathcal{I}, \mathcal{T}')$ of NonNestDextP to an instance $\mathcal{I}'$ of TiledICEx proceeds as follows. For each page p, we build a disjoint collection of intervals $\mathcal{D}_p$ as shown in Figure 11, by greedily picking a set of nonoverlapping intervals in $\{D_t^p : t \ge 0\}$ and extending them to partition the timeline. In the instance $\mathcal{I}'$, the set $\mathcal{I}'(p)$ of intervals for page p is given by the intervals in $\mathcal{D}_p$, and the excluded page p_t is the page requested at time t. Furthermore, the cost of each interval in $\mathcal{I}'(p)$ is given by $w(p)$, and the covering requirement R at each time t is $n - k$. The natural LP relaxation for $\mathcal{I}'$ has a variable z_I for each interval I such that for every time t,

$$\sum_{I : I \in \mathcal{I}'(p), p \neq p_t, t \in I} z_I \ge n - k. \tag{5.1}$$

LEMMA 5.9. *Given the instance $(\mathcal{I}, \mathcal{T}')$ as above, let $\mathcal{I}'$ be the corresponding TiledICEx instance.*

(i) *Let $(\bar{x}, \bar{y})$ be a fractional solution to* (IP-D3p) *with $\bar{y}_{I_t} = 0$ for all $t \in \mathcal{T}'$. Then there is a fractional solution to the above LP relaxation for $\mathcal{I}'$ of cost at most twice that of $(\bar{x}, \bar{y})$.*

---

1 Initialize $\mathcal{D}_p \leftarrow \varnothing, t^\star \leftarrow 0$.
2 **for** $t = 1, 2, \ldots$ **do**
3 **if** D_t^p *does not contain* $t^\star$ *in the interior* **then**
4 Add $[t^\star, t]$ to $\mathcal{D}_p$
5 Update $t^\star \leftarrow t$.

FIG. 11. *The online procedure to construct the partition $\mathcal{D}_p$ for page p.*

(ii) *Let $\mathcal{S}$ be an integral solution to $\mathcal{I}'$. Then there is an integral solution to the* NonNestDextP *instance $(\mathcal{I}, \mathcal{T}')$ of cost at most twice that of $\mathcal{S}$.*

Moreover, both the above constructions can be done efficiently.

Proof. Let $(\bar{x}, \bar{y})$ be a fractional solution to (IP-D3p). We construct a fractional solution z for the instance $\mathcal{I}'$ as follows. For every variable $\bar{x}_{p,t}$, let I be the interval in $\mathcal{D}_p$ containing t, and let $\mathtt{lt}(I)$ be the interval in $\mathcal{D}_p$ immediately to the left of I. We raise both z_I and $z_{\mathtt{lt}(I)}$ by $\bar{x}_{p,t}$. Finally, if any z_I variable exceeds one, we cap it at one. The cost of z is at most twice that of $\bar{x}$. To show feasibility, consider a time t. Let I_1 be the interval in $\mathcal{D}_p$ containing t and $I_2 = \mathtt{rt}(I_1)$ be the interval immediately to the right of I_1 in $\mathcal{D}_p$. Both I_1 and I_2 contain intervals $D^p_{t_1}$ and $D^p_{t_2}$ for some times t_1 and t_2, respectively. Therefore, Claim 5.8 implies that D^p_t is contained in $I_1 \cup I_2$. It follows that we will raise z_{I_1} by at least $\min(1, \sum_{t' \in D^p_t} \bar{x}_{p,t'})$. From (D3p) it follows that z is a feasible solution.

We now prove the second part of the lemma. Let $\mathcal{S}$ be a feasible solution to $\mathcal{I}'$. We will build an integral solution (x, y), where $y_{I_t} = 0$ for all $t \in \mathcal{T}'$, for $(\mathcal{I}, \mathcal{T}')$. For every $I = [t_1, t_2] \in \mathcal{S} \subseteq \mathcal{D}_p$, we set $x_{p,t_1} = x_{p,t_2} = 1$. The cost guarantee follows easily. To verify feasibility, consider a time t. For each page $p \neq p_t$, let $I(p)$ be the interval in $\mathcal{I}'(p)$ which contains t. Let $P(t)$ be the set of pages for which $\mathcal{S}$ contains $I(p)$. By feasibility of the z-solution, $|P(t)| \geq (n - k)$. The interval D^p_t and $I(p)$ overlap (both of them contain time t). Also, D^p_t cannot be contained in $I(p)$, by the way the set $\mathcal{D}_p$ is constructed. Therefore, D^p_t must contain one of the two end-points of $I(p) := [t_1, t_2]$. Since we set both x_{p,t_1}, x_{p,t_2} to 1, it follows that the LHS term corresponding to page p in (D3p) (for time t) is also 1. This shows that (x, y) is feasible. ∎

Combining Lemma 5.9 with the 2-approximation for TiledICEx from Lemma D.1 gives an 8-approximation algorithm for NonNestDextP. As in subsection 5.1.1, the reduction from the proof of Lemma 5.9(ii) can be carried out in an online manner. If the online algorithm for TiledICEx selects an interval $I := [t_1, t_2]$ at a time t (note that $t \leq t_2$), we need to add the stars (p, t_1) and (p, t_2) in our solution for $\mathcal{I}$. It turns out that the proof of Lemma 5.9 holds if we add the stars (p, t) and (p, t_2) instead. Further, the star (p, t_2) can be added at time t_2. Since the set of constraints (D3p) corresponding to time t involve variables at t and earlier only, the online algorithm need not *remember* at time t the stars which will appear in future—it can keep track of all the stars which have been added at time t, and any such star which corresponds to time $t' > t$ will only appear at time t' in the algorithm. Thus, the algorithm satisfies the property that at any time t, it will only add stars corresponding to time t—we call such algorithms *present restricted*; this is a stronger property than both past-preservation and sparsity (which were defined in section 3).

LEMMA 5.10. *There is an online $O(\log k)$-competitive present restricted algorithm to* NonNestDextP*. Moreover, there is an offline 8-approximation algorithm for* NonNestDext*.*

5.2.2. Algorithm for the general case of SolveDext. We now consider the general setting where the critical intervals I_t may be nested. Corollary 5.7 shows that at every time t, we know whether $\bar{y}_{I_t} = 1$ or not, so we need only worry about times for which $\bar{y}_{I_t} = 0$—call these times $\mathcal{T}$. Let $\mathcal{I}$ be a general instance of SolveDextP, where we obtained a cover for times in $\mathcal{T}$. We show how to extend a solution for a NonNestDextP subinstance into one for the original instance $\mathcal{I}$, while losing a constant factor in the cost. Let us give some useful notation. Given a set of times $\mathcal{T}$, a subset $\mathcal{N}$ is a *nonnested net* of $\mathcal{T}$ if

(i) for times $t_1 \neq t_2 \in \mathcal{N}$, their critical intervals I_{t_1}, I_{t_2} are nonnested, and
(ii) for every time $t \in \mathcal{T} \setminus \mathcal{N}$, there is a time $t' \in \mathcal{N}$ such that I_t contains $I_{t'}$.

A greedy algorithm to construct a nonnested net $\mathcal{N}$ of $\mathcal{T}$ simply scans times in $\mathcal{T}$ from left to right and adds time t to $\mathcal{N}$ whenever I_t does not contain $I_{t'}$ for any $t' \in \mathcal{N}$. This procedure is implementable online: whenever we see a time t, we know whether it gets added to $\mathcal{N}$ or not. Given a set $\mathcal{T}$ of times and a nonnested net $\mathcal{N}$ of $\mathcal{T}$, we define a map $\varphi : \mathcal{T} \setminus \mathcal{N} \to \mathcal{N}$ as follows—for a time $t \in \mathcal{T} \setminus \mathcal{N}$, let $\varphi(t)$ be the rightmost time $t' \in \mathcal{N}$ such that I_t contains $I_{t'}$.

CLAIM 5.11 (monotone map). *Let $\mathcal{T}$ be a set of times, $\mathcal{N}$ be a nonnested net of $\mathcal{T}$, and φ be the associated map as above. Then for any $t'_1, t'_2 \in \mathcal{T} \setminus \mathcal{N}$, $t'_1 < t'_2 \implies \varphi(t'_1) \leq \varphi(t'_2)$.*

Proof. Suppose there are $t'_1 < t'_2 \in \mathcal{T} \setminus \mathcal{N}$ such that $\varphi(t'_1) = t_1 > t_2 = \varphi(t'_2)$. Let $I_{t_1} = [s_1, t_1]$ and $I_{t_2} = [s_2, t_2]$. Since these two intervals are nonnested, it must be the case that $s_1 \geq s_2$. But then $I_{t'_2}$ contains I_{t_1} as well and we would have set $\varphi(t'_2) = t_1$. □

Given an integer solution $(\bar{x}, \bar{y})$ for SolveDextP, we identify it with a set $A^\star$ of stars, where $A^\star := \{(p,t) \mid x_{p,t} = 1\}$. For a time t and a set of elements $A^\star$, let $P(A^\star, t)$ denote the set of pages for which the corresponding intervals D^p_t are hit by $A^\star$. That is, we can rephrase constraint (D3p) as wanting to find a set $A^\star$ such that $P(A^\star, t) \setminus \{p_t\}$ has at least $n - k$ pages. The main technical ingredient is the following extension result.

THEOREM 5.12 (extension theorem). *There is an algorithm that takes a set $\mathcal{T}'$ of times, a nonnested net $\mathcal{N}' \subseteq \mathcal{T}'$, the associated monotone map φ, and a set $A^\star$, and outputs another set $B^\star \supseteq A^\star$ such that*

(i) *$P(B^\star, t) \supseteq P(A^\star, \varphi(t))$ for all $t \in \mathcal{T}' \setminus \mathcal{N}'$, and*
(ii) *$w(B^\star) \leq 3\, w(A^\star)$.*

This algorithm can be implemented in an online manner as well. More formally, assume there is a present preserving online algorithm which generates the set $A^\star_t$ at time $t \in \mathcal{T}$. Then there is a present preserving online algorithm which generates $B^\star_t$ at time $t \in \mathcal{T}$ and satisfies conditions (i) *and* (ii) *above (with $A^\star$ and $B^\star$ replaced by $A^\star_t$ and $B^\star_t$, respectively).*

We defer the proof to Appendix C, and instead explain how to use the result in the offline setting first. We invoke the extension theorem twice. For the first invocation, we use Theorem 5.12 with the entire set of times $\mathcal{T}$, a net $\mathcal{N}$ and the associated monotone map φ, and with $A^\star$ being a solution of weight at most $8\,\mathsf{opt}(\mathcal{I})$ given by Lemma 5.10 on the subinstance $\mathcal{N}$. This outputs a set $B^\star$ with $w(B^\star) \leq 24\,\mathsf{opt}(\mathcal{I})$. Moreover, since $A^\star$ is feasible for $\mathcal{N}'$, it follows from the first property of Theorem 5.12 that $|P(B^\star, t) \setminus \{p_t\}| \geq |P(A^\star, \varphi(t))| - 1 \geq n - k - 1$ for every time $t \in \mathcal{T} \setminus \mathcal{T}'$.

For the second invocation, let $\mathcal{T}_1 \subseteq \mathcal{T} \setminus \mathcal{N}$ be the subset of times t such that $|P(B^\star, t) \setminus \{p_t\}| = n - k - 1$, i.e., those with unsatisfied demand. We use Theorem 5.12 again, this time with $\mathcal{T}_1$, a net $\mathcal{N}_1$ and the associated monotone map φ_1, and a solution $A^\star_1$ obtained by using Lemma 5.10 on the subinstance $\mathcal{N}_1$. This gives us $B^\star_1$ with weight at most $24\,\mathsf{opt}(\mathcal{I})$. We output $B^\star \cup B^\star_1$ as our solution. Somewhat surprisingly, this set $B^\star_1$ gives us the extra coverage we want, as we show next.

LEMMA 5.13 (feasibility). *For any time t, $|P(B^\star \cup B^\star_1, t) \setminus \{p_t\}| \geq n - k$.*

Proof. We only need to worry about times $\mathcal{T}_1 \setminus \mathcal{N}_1$. Consider such a time t_1. Let $t_2 := \varphi_1(t_1)$ and $t_3 := \varphi(t_2)$. For sake of brevity, let p_i denote p_{t_i}, and let I_i denote

I_{t_i}. Note that $I_3 \subset I_2 \subset I_1$, and since there are no nested intervals of the same page, also $p_1 \neq p_2 \neq p_3$. Recall that we want to show $|P(B^\star \cup B_1^\star, t_1)| \setminus \{p_1\}| \geq n-k$.

Note that $P(B^\star, t_2)$ contains $P(A^\star, t_3)$, and the latter has size at least $n-k$ by construction. Hence, for t_2 to appear in $\mathcal{T}_1$, we must have $|P(B^\star, t_2)| = n-k$ and $p_2 \in P(A^\star, t_3)$. Since I_1 contains I_3, $D_{t_1}^{p_2}$ contains $D_{t_3}^{p_2}$, and so, $B^\star$ hits $D_{t_1}^{p_2}$, i.e., $p_2 \in P(B^\star, t_1)$.

Since $P(A_1^\star, t_2) \setminus \{p_2\}$ has size $n-k$ (by construction of $A_1^\star$), and $P(B_1^\star, t_1)$ contains $P(A_1^\star, t_2)$, it follows that $P(B_1^\star, t_1) \setminus \{p_2\}$ also has size at least $n-k$. Therefore, $P(B^\star \cup B_1^\star, t_1)$ has size at least $n-k+1$ because $P(B^\star, t_1)$ contains p_2. This implies the lemma. ∎

Since $w(B^\star \cup B_1^\star) \leq 48\mathsf{opt}(\mathcal{I})$, we get a 48-approximation algorithm for SolveDextP. It is easy to check that these arguments carry over to the online case as well; we briefly describe the main steps. The set $\mathcal{N}$ can be generated in an online manner using a greedy algorithm (as mentioned in the beginning of this section). We invoke the online algorithm in Lemma 5.10 to get a present restricted solution $A_t^\star$ for all $t \in \mathcal{N}$. Theorem 5.12 implies that the present restricted solution $B_t^\star$ can be constructed at time t. Given $B_t^\star$, we can tell whether a particular time t qualifies for being in $\mathcal{T}_1$. The same argument can now be repeated to show that we can maintain $(B_1^\star)_t$ for all $t \in \mathcal{T}_1$. Combining this with Lemma 5.10, we get the next lemma.

LEMMA 5.14. *There is an online $O(\log k)$-competitive present restricted algorithm to SolveDextP. Moreover, there is an offline 48-approximation algorithm for SolveDextP.*

COROLLARY 5.15. *There is a constant factor (offline) approximation algorithm for* (IPp). *Further, there is an $O(\log k)$-competitive online solution that satisfies the past-preserving and sparsity property as in section* 3.

Proof. Let $\mathcal{I}$ be an instance of PageTWPenalties. Let $A_1^\star$ and $A_2^\star$ be (offline) solutions for (IP-Rp) and (IP-D3p) as guaranteed by Lemmas 5.4 and 5.14, respectively. Lemma 5.5 shows that $A_2^\star$ can be mapped to a solution $A_3^\star$ that satisfies (IP-D1p), and $\mathsf{cost}(A_3^\star) \leq 2\ \mathsf{cost}(A_2^\star)$. Further $A^\star := A_1^\star \cup A_3^\star$ is a feasible solution to (IPp). Since (IP-Rp) and (IP-D1p) are special cases of (IPp), Lemmas 5.4, 5.5, and 5.14 imply that

$$\mathsf{cost}(A_1^\star)+\mathsf{cost}(A_3^\star) \leq 6\ \mathsf{opt}(\mathcal{I})+2\ \mathsf{cost}(A_2^\star) \leq 6\ \mathsf{opt}(\mathcal{I})+96\ \mathsf{opt}(\text{IP-D1p}) \leq O(1)\cdot\mathsf{opt}(\mathcal{I}).$$

The online version follows analogously. Note that the conversion from $A_2^\star$ to $A_3^\star$ in Lemma 5.5 can be carried out in an online manner, and if $A_2^\star$ is present restricted, then so is $A_3^\star$. Since $A_1^\star$ and $A_3^\star$ are past-preserving, so is $A^\star$. Also the sparsity of $A_1^\star$ and the fact that $A_3^\star$ does not add any star in the future implies that $A^\star$ also satisfies the sparsity property. ∎

Corollary 5.15, along with Theorems 3.1 and 4.1, implies Theorems 1.1 and 1.2, respectively. The integrality gap of (IPp) is constant for the following reason—the integrality gap of the LP relaxations for SolveDext and NonNestDext are $O(1)$, and the reductions in Lemmas 5.2, 5.3, and 5.9 also hold between the fractional solutions to the corresponding problems.

6. Extension to paging with delay. In this section, we show a simple reduction from the *(weighted) paging with delays* (PageD) problem to the PageTWPenalties problem which allows us to translate the results of the previous sections giving an $O(1)$-approximate offline algorithm and an $O(\log k \log n)$-competitive online algorithm for the PageTWPenalties problem to get the same asymptotic performance for the PageD problem.

We transform an instance $\mathcal{I}$ of PageD to an instance $\mathcal{I}'$ of PageTWPenalties as follows. Recall that each request in $\mathcal{I}$ is specified by a triple (p, t, F), where p is the requested page, t is the time at which this request is made, and $F : \{t, t+1, \dots, \} \to \mathbb{R}_{\geq 0}$ denotes the nondecreasing loss function associated with it. We may assume wlog that $F(t) = 0$, since otherwise we can work with the function $F'(t') := F(t') - F(t)$, and the competitive ratio is no worse. To model this request, we create an ensemble of intervals $[t, t']$ for each $t' \geq t$ in the PageTWPenalties instance $\mathcal{I}'$, where the penalty for the interval $I := [t, t']$ is $F(t'+1) - F(t')$, for each $t' \geq t$.

To see the equivalence, suppose this request (p, t, F) is served at time t'—i.e., the page p enters the cache after time t only at time t'. Then all intervals in its ensemble ending at later times are also satisfied. Moreover, intervals ending at earlier times $t, t+1, \dots, t'-1$ are not satisfied, and their penalty adds up to $F(t') - F(t) = F(t')$, as desired. Given this equivalence and the algorithmic results for the PageTWPenalties problem, we get the following.

THEOREM 6.1. *There is an $O(\log k \log n)$-competitive online algorithm and an $O(1)$-approximate offline algorithm for the PageD problem.*

This completes the proof of Theorems 1.1 and 1.2.

Appendix A. NP-hardness of PageTW. We now show that the PageTW is APX-hard, even when the cache size $k = 1$ and we have unit weights. The reduction is the same as that of Nonner and Souza [NS09] for the joint-replenishment problem, and we give it here for completeness. The reduction is from the (unweighted) Vertex Cover problem on bounded-degree graphs.

Consider an instance $\mathcal{I}$, consisting of a graph $G = (V, E)$, of the Vertex Cover problem. We reduce it to an instance $\mathcal{I}'$ of the PageTW problem. In the instance $\mathcal{I}'$, we have one page p_e for every edge $e \in E$. We also have a special page $p^\star$. All pages have unit weight and the cache size k is 1. We now specify the request intervals for each page. The timeline T is the line $[0, |V| + 1]$. For the page $p^\star$ we have request intervals $[t, t]$ for every integer $t \in T$, i.e., this page must be in the cache (or brought into the cache) at each integer time t. Now consider the page p_e for the edge $(u, v) \in E$. Assume wlog that $u < v$. We have three request intervals for this page e: $I_e^1 = [0, u], I_e^2 = [u, v], I_e^3 = [v, n+1]$, where n denotes $|V|$. Note that these are closed intervals. This completes the description of the reduction. We first prove the easier direction (see Figure 12 for an example).

CLAIM A.1. *Suppose there is a vertex cover of G of size at most r (in the instance $\mathcal{I}$). Then there is a solution to $\mathcal{I}'$ of cost at most $r + 2|E| + 1$.*

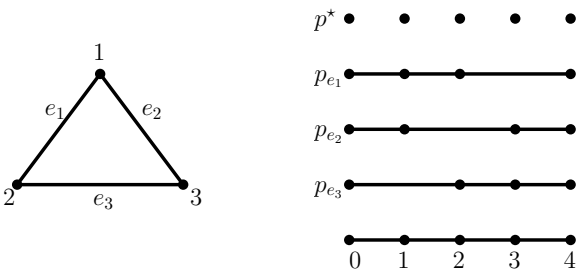

FIG. 12. *Illustration of the reduction from Vertex Cover to PageTW. The page $p^\star$ is requested at each time. All the other pages have three request intervals, shown by solid lines with end-point delimiters.*

Proof. Let U be a vertex cover of size r. The caching schedule is as follows: we will ensure that at the end of each time t, the page $p^\star$ is in the cache. This will ensure that all requests for $p^\star$ are satisfied. For every $u \in U$, we do the following: let $N(u)$ be the edges incident to u in G. We bring each of the pages $p_e \in N(u)$ in the cache and then evict it. At the end of this process (at time u) we bring $p^\star$ back in the cache.

For every edge $e \in E$, we have ensured that we bring e in the cache either at time u or v (or both). If we bring in e at both times, we have satisfied all the requests for p_e. Otherwise, we would have satisfied two out of the three request for p_e, and the unsatisfied request would be either I_e^1 or I_e^3. Let E_1 be the set of edges e for which the request I_e^1 is unsatisfied, and let E_3 be the set of edges e for which I_e^3 is unsatisfied. At time 0, we bring in and then evict all pages in I_e^1. Then we bring in page $p^\star$. At time $n+1$, we evict $p^\star$ and bring in and then evict all the pages in E_3. This yields a feasible caching schedule. The total number of times $p^\star$ is evicted is at most $r+1$ (at each of the times in U, and maybe at time $n+1$). Every page p_e is evicted exactly twice. This proves the claim. ∎

CLAIM A.2. *Suppose there is a solution to $\mathcal{I}'$ of cost at most $r + 2|E| + 1$. Then there is a vertex cover of G of size at most $r+1$.*

Proof. Let $\mathcal{S}$ be a solution to the caching problem. For an edge e, let T_e be the time-steps when e is brought into the cache. Since $p^\star$ must be present at the end of each integer time, T_e must have nonempty intersection with each of the three request intervals for e. We now modify $\mathcal{S}$ to a solution $\mathcal{S}'$ which has the following property: (i) the total cost of $\mathcal{S}'$ is at most that of $\mathcal{S}$, and (ii) for every edge $e = (u, v)$, the corresponding set T'_e in $\mathcal{S}'$ has nonempty intersection with $\{u, v\}$.

Initialize $\mathcal{S}'$ and T'_e to $\mathcal{S}$ and T_e, respectively. While there is an edge $e = (u, v)$ such that T'_e does not contain u or v, we do the following: T'_e must contain a distinct time in each of the intervals I_e^1, I_e^2, I_e^3—let t_1, t_2, t_3 denote these three times. Note that t_2 must be in the interior of I_e^2. Assume wlog that $u < v$. Instead of bringing in p_e at times t_1 and t_2 (and evicting them at these times), we will bring in p_e at time u. This will save us a cost of 1 in the total eviction cost of p_e. However, it may happen that earlier $p^\star$ was not getting evicted at time u, and now we will need to evict it (and then bring it back into cache) at time u. Still, this will not increase the cost of the solution.

Thus, we see that $\cup_e T'_e$ must contain a vertex cover U of G. Since $p^\star$ must be getting evicted at each of these times, the total cost of $\mathcal{S}'$ (and hence that of $\mathcal{S}$) is at least $2|E| + |U|$. This implies the claim. ∎

Using the above two claims, we show that the PageTW problem is APX-hard.

LEMMA A.3. *Let G be a graph of maximum degree 4. Suppose there is a $(1+\varepsilon)$-approximation for PageTW problem. Then there is a $(1+9\varepsilon)$-approximation for Vertex Cover on G.*

Proof. Let $\mathcal{A}$ be the α-approximation algorithm for PageTW. The algorithm for Vertex Cover on G is as follows: use the reduction described above to get an instance $\mathcal{I}'$ of PageTW. Run $\mathcal{A}$, and then use the proof of Claim A.2 to get a vertex cover for G.

Suppose G has a vertex cover of size r. Since the maximum degree of G is 4, we know that $|E| \le 4r$. Now Claim A.1 implies that $\mathcal{I}'$ has a solution of cost at most $2|E| + r + 1$, and so $\mathcal{A}$ outputs a solution of cost at most $(1+\varepsilon)(2|E| + r + 1) = 2|E| + 2\varepsilon|E| + (1+\varepsilon)(r+1) \le 2|E| + (1+9\varepsilon)r + O(1)$. Claim A.2 now shows that

there is a vertex cover of size at most $(1+9\varepsilon)r + O(1)$ in G. This proves the lemma; the additive 1 can be ignored because we can take multiple copies of G and make r as large as we want. ∎

Finally, the fact that vertex cover is hard to approximate to within ≈ 1.02 on 4-regular graphs [CC06] implies that PageTW is ≈ 1.002-hard and completes the proof.

Appendix B. Some illustrative examples.

B.1. Evictions at end-points are insufficient. It is easy to check that we cannot hope to service every interval I at either $s(I)$ or $t(I)$, which we can do for the unweighted case. Indeed, consider the following input (see Figure 13): suppose $k = 1$ and there is a very heavy page which is requested very frequently, i.e., there are many disjoint short request intervals for it (as shown below) of the form $[n-\varepsilon, n+\varepsilon]$ for all positive integers n and small enough parameter $\varepsilon > 0$ (although the start and end times here are fractional, it is easy to make these integral by suitably scaling the instance). So we need to have this heavy page in the cache at every time. Now there are n unit weight pages, but their request intervals are $[0, n], [1, n+1]$, $[2, n+2], \ldots$.

The optimal solution is to service all these requests at time n, because then we will evict the heavy page only once. Thus, our algorithm needs to use these windows of opportunity to service as many cheap requests as possible.

B.2. An integrality gap for the interval hitting LP. We now consider a natural LP relaxation for PageTW which extends that for weighted caching and show that it has a large integrality gap. We have variables $x_{p,J}$ for pages p and intervals $J \subseteq [T]$, indicating that J is the maximal interval during which the page p is in the cache for the entire interval J. Recall that we are allowed to service many requests at each time-step, so each time-step may have up to n loads and n evictions. To handle this situation, we "expand" the timeline so that all such "instantaneous" services can be thought of as loading each page in the cache for a tiny amount of time, and then evicting it. This will ensure that we can write a packing constraint in the LP relaxation which says that no more than k pages are in the cache at any particular time.

Let N be a large enough integer ($N \geq n$, where n is the number of distinct pages will suffice). We assume that all $s(I), t(I)$ values for any request interval I are multiples of N (this can be easily achieved by rescaling). Let E denote the set of end-points of the request intervals (so each element in E is a multiple of N). As above, we have variables $x_{p,J}$, where the end-points of J are integers (which *need not* be multiples of N). The idea is that between two consecutive intervals of E, we can

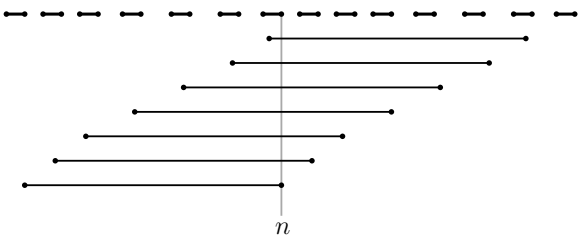

FIG. 13. *An example to show that all requests cannot be served at the beginning or the end of the interval.*

pack N distinct unit size intervals, each of which may correspond to loading and then evicting a distinct page. We can now write the LP relaxation:

$$\min \sum_{p,J} w(p) \cdot x_{p,J},$$

$$\text{(B.1)} \qquad \sum_{J: J \cap I \neq \varnothing} x_{p,J} \geq 1 \qquad \forall \text{ request intervals } I \text{ with } p = \mathsf{page}(I),$$

$$\text{(B.2)} \qquad \sum_{p} \sum_{J: t \in J} x_{p,J} \leq k \qquad \forall \text{ integer times } t,$$

$$x_{p,J} \geq 0.$$

THEOREM B.1. *The above LP has an integrality gap of* $\Omega(k)$.

Proof. Suppose we have k "heavy" pages with weight k each and one "light" page with weight 1. The request intervals for each of the pages are $E_0, E_1, \ldots, E_T$, where $E_i = [ikN, (i+1)kN]$, and N and T are suitable large parameters ($T, N > k^3$ will suffice).

We first argue that any integral solution must have $\Omega(T)$ cost. To see this, consider the request intervals $E_0, E_4, E_8, \ldots$ for the light page p. The page p must be brought at least once during each of these intervals—say at timeslots $t_0, t_4, t_8, \ldots$, where $t_{4i} \in E_{4i}$ for all i. Notice that E_{4i+2} lies strictly between t_{4i} and t_{4i+4}, and hence each of the heavy pages must be present at least once during E_{4i+2}. Since there can be at most $k-1$ heavy pages in the cache at time t_{4i}, it follows that at least one heavy page must be brought into the cache during $[t_{4i}, t_{4i+4}]$. This argument shows that the cost of any integral solution must be $\Omega(Tk)$.

Now we argue that there is a fractional solution to the LP of total cost $O(T)$. For each heavy page q, we define $x_{q,J} = 1 - 1/k^2$, where $J = [0, (T+1)kN]$, i.e., J is the entire timeline. Notice that each interval E_i is of length Nk. We can therefore find $(k+1)k^2$ disjoint intervals of length 1 each in it, and "assign" k^2 of these intervals to each of the $k+1$ pages. Let $\mathcal{S}_{i,p}$ be the set of k^2 unit length intervals assigned to page p (which could be the light page or one of the heavy pages). For each heavy page q and each unit length interval H assigned to it, we set $x_{q,H}$ to $1/k^4$. For the light page p and each unit length interval H assigned to it, we set $x_{p,H}$ to $1/k^2$.

Now we check feasibility of this solution. Consider a heavy page q and the request interval E_i for it. The LHS of constraint (B.1) for this request is $(1-1/k^2)+k^2/k^4 = 1$, where the first term corresponds to $x_{p,J}$ and the second term comes from the k^2 unit length intervals in $\mathcal{S}_{i,q}$. For the light page p and the request interval E_i for it, the LHS of this constraint is 1, because each of the k^2 unit length intervals H in $\mathcal{S}_{i,p}$ has $x_{p,H}$ equal to $1/k^2$. The constraint (B.2) is easy to check—for any time t, the LHS is at most $k(1-1/k^2)+1/k^2 \leq k$, because the first term comes from $x_{q,J}$ for each heavy page q, and the second term comes from the fact that all the unit length intervals are disjoint.

Let us now compute the cost of this solution. For a heavy page q, the total cost is $k(1-1/k^2)+T/k$, where the first term comes because of the long interval J and the second term is because of the unit length intervals. This is $O(T/k)$. Summing over all heavy pages, this cost is $O(T)$. For the light page, we have Tk^2 unit length intervals, each to a fractional extent of $1/k^2$. Therefore the total cost here is T as well. This proves the integrality gap of $\Omega(k)$. ∎

The essential problem with this LP is that the heavy pages are being almost completely fractionally assigned, leaving a tiny ε amount of space. Since all the

requests are long, they can be slowly satisfied over $1/\varepsilon$ time periods, which is much less cost than the cost of actually evicting a heavy page.

Appendix C. Proof of the extension theorem.

THEOREM 5.12 (extension theorem). *There is an algorithm that takes a set $\mathcal{T}'$ of times, a nonnested net $\mathcal{N}' \subseteq \mathcal{T}'$, the associated monotone map φ, and a set $A^\star$, and outputs another set $B^\star \supseteq A^\star$ such that*

(i) $P(B^\star, t) \supseteq P(A^\star, \varphi(t))$ *for all* $t \in \mathcal{T}' \setminus \mathcal{N}'$, *and*
(ii) $w(B^\star) \leq 3\, w(A^\star)$.

This algorithm can be implemented in an online manner as well. More formally, assume there is a present preserving online algorithm which generates the set $A^\star_t$ at time $t \in \mathcal{T}$. Then there is a present preserving online algorithm which generates $B^\star_t$ at time $t \in \mathcal{T}$ and satisfies conditions (i) *and* (ii) *above (with $A^\star$ and $B^\star$ replaced by $A^\star_t$ and $B^\star_t$, respectively).*

Proof. The procedure to obtain $B^\star$ from $A^\star$ is a simple greedy procedure and appears in Figure 14: it goes over the times in $\mathcal{T}' \setminus \mathcal{N}'$ and fixes any violations to the containment condition of the theorem by adding a new element to $B^\star$. For brevity, define $\mathcal{T}'' := \mathcal{T}' \setminus \mathcal{N}'$. It immediately follows that for any $t' \in \mathcal{T}''$, the set $P(A^\star, \varphi(t))$ is a subset of $P(B^\star, t)$. We just need to bound the cost of $B^\star$.

Let the times in $\mathcal{T}''$ be $t_1 < t_2 < \ldots$, and Claim 5.11 shows that $\varphi(t_1) \leq \varphi(t_2) \leq \ldots$. The desired result now easily follows from the following claim.

CLAIM C.1. *Suppose we add $(p, t_i), (p, t_j), (p, t_k)$ to the set $B^\star$ for some page p and times $t_i < t_j < t_k$. Then there is a time $t \in [t_i, t_k]$ such that $(p, t) \in A^\star$.*

Proof. Fix a page p with t_i, t_j, t_k as in the statement above, and let $t \leq t_k$ be the largest time such that $(p, t) \in A^\star$. We now make a sequence of observations:

(i) We claim that $\varphi(t_k) \geq t_j$ and $I_{\varphi(t_k)} \subseteq [t_j, \varphi(t_k)]$. If either is false, then I_{t_k} contains t_j, in which case there is no need to add (p, t_k) to $B^\star$, because $B^\star$ already contains (p, t_j).
(ii) Moreover, the interval $D^p_{t_j} \subseteq [t_i, t_j]$. Clearly $D^p_{t_j}$ ends at t_j (by definition). If it starts before t_i, then there is no need to add (p, t_j) to $B^\star$.
(iii) Next, $D^p_{\varphi(t_k)} \subseteq [t_i, \varphi(t_k)]$: Since $\varphi(t_k) \geq t_j$, $D^p_{\varphi(t_k)}$ starts after (or at the same time as) $D^p_{t_j}$ starts, and so this follows by (ii) above.

Now since p lies in $P(A^\star, \varphi(t_k))$, statement (iii) implies that we have $(p, t) \in A^\star$ for some $t \in [t_i, \varphi(t_k)]$. ∎

Claim C.1 means that we can charge the three elements added to $B^\star$ to this element $(p, t) \in A^\star$ that lies in between $[t_1, t_3]$. This proves the cost bound, and hence Theorem 5.12.

In the online setting, $B^\star_t$ can be easily constructed from $A^\star_t$ using the procedure in Figure 14, and it is easy to check it is also present restricted. ∎

```
1 Initialize B* ← A*.
2 for t ∈ T' \ N' in increasing order do
3     for every page p ∈ P(A*, φ(t)) \ P(B*, t) do
4         Add (p, t) to B*.
5 return B*.
```

FIG. 14. *The extension procedure to prove Theorem* 5.12.

Appendix D. The tiled interval cover problem. In the *tiled interval cover* problem (TiledIC), the input is the following. For each page $p \in [n]$, we have a collection $\mathcal{I}_p$ of disjoint intervals that cover the entire timeline, with each such interval having weight $w(p)$. We also have a requirement $n-k$. The goal is to pick some set of intervals from $\mathcal{I} = \cup_p \mathcal{I}_p$ that minimize their total weight, such that every time t is covered by $n-k$ different intervals. In the version *with exclusions* (TiledICEx), the interval E_t ending at time t does not count toward the requirement of $n-k$ at time t. (As always we assume that a unique interval ends at each time.)

D.1. The offline case.

LEMMA D.1. *The linear relaxation for the TiledIC problem is integral, whereas that for the TiledICEx problem has an integrality gap of at most* 2.

Proof. We can even show this for the case where the weights and requirements are nonuniform, i.e., each time t has a potentially different requirement R_t, and each interval has a different weight w_I. Indeed, for the TiledIC problem, the constraint matrix (where $\mathcal{S}$ is the set of all intervals in the instance)

$$\min_{z \in [0,1]^{|\mathcal{S}|}} \quad \sum_I w_I \, z_I, \qquad \sum_{I \in \mathcal{S}: t \in I} z_I \geq R_t \qquad \forall t \tag{D.1}$$

has the consecutive-ones property and forms a totally unimodular system, so the linear relaxation has integer extreme points and an optimal integer solution can be found in polynomial time.

Now let z be a solution to the LP relaxation for (note the exclusion of E_t from the sum)

$$\min_{z \in [0,1]^{|\mathcal{S}|}} \quad \sum_I w_I \, z_I, \qquad \sum_{I \in \mathcal{S}: t \in I, I \neq E_t} z_I \geq R_t \qquad \forall t. \tag{D.2}$$

Recall that E_t is the interval ending at t (though we will not need this for our solution). To construct an integer solution $\mathcal{S}'$, first add to $\mathcal{S}'$ all the intervals I with $z_I \geq 1/2$. Now for each time t, let R'_t be the residual coverage needed; i.e., define $R'_t := R_t - \#\{I \in \mathcal{S}' \mid t \in I, I \neq E_t\}$. Moreover, define $\tilde{z}_I := 2z_I$ for $I \notin \mathcal{S}'$, and $z_I = 0$ for $I \in \mathcal{S}'$. Clearly, $\sum_{I \ni t: I \neq E_t, I \notin \mathcal{S}'} \tilde{z}_I \geq 2R'_t$. Treat this as a solution to a TiledIC instance on the subcollection $\mathcal{S} \setminus \mathcal{S}'$ (crucially, ignoring the exclusions) with these adjusted requirements $2R'_t$, and let $\mathcal{S}''$ be an optimal integer solution. For each time t, there are now $(R_t - R'_t)$ nonexcluded sets in $\mathcal{S}'$ and at least $(2R'_t - 1)^+ \geq \max(R'_t, 0)$ nonexcluded sets from $\mathcal{S}''$ covering it, which gives the desired coverage level of R_t. Due to the rounding up by a factor of 2, the cost of the solution is at most $2\, w^\intercal z$. ∎

D.2. The online case. The online model for TiledIC and TiledICEx is that intervals are revealed online: specifically, the end-point of an interval is revealed only when it ends (and since we are dealing with tiled instances, the next interval for that page starts immediately thereafter).

In the online case, the TiledICEx happens to be essentially identical to the formulation used in online primal-dual algorithms for weighted paging, e.g., by [BBN12].

There are n pages and a cache of size k, so these constraints say that at time t, there must have been $n-k$ pages apart from p that are evicted since they were last requested. Hence, the intervals for a page start just after each request for the page and end at the time of the next request. This means we can simulate the end of intervals in $\mathcal{I}_p$ by requesting page p. The integer program is the following, where the page I_t corresponds to the page p_t requested at time t:

$$\min \left\{ \sum_p \sum_{I \in \mathcal{I}_p} w(p) x_I \mid \sum_{I \in \mathcal{I} \setminus \{I_t\} : t \in I} x_I \geq n-k \ \ \forall t, \quad x_I \in \{0,1\} \ \ \forall I \right\}.$$

Using this connection and the result of [BBN12] immediately gives us an $O(\log k)$ randomized online algorithm for TiledICEx. To make the online model closer to the rest of the paper, let us reformulate the above IP as follows:

$$\min \left\{ \sum_{p,t} w(p) x_{p,t} \mid \sum_{I \in \mathcal{I} \setminus \{I_t\} : t \in I} \min \left(\sum_{t' \in I, t' \leq t} x_{p,t'}, 1 \right) \geq n-k \ \forall t, \quad x_{p,t} \in \{0,1\} \ \forall p, t \right\}.$$

It is easy to switch between these two formulations, using the correspondence that at some time t, the variable x_I has value equal to $\min(1, \sum_{t' \in I : t' \leq t} x_{pt'})$. The algorithm from [BBN12] gives us an algorithm that only changes the variables at the current time t; hence this is clearly a *past-preserving* algorithm.

To get an algorithm for TiledIC, we change the instance slightly: we add in a new page p_0 (so there are $n+1$ pages) and make the cache of size $k+1$. This new page has weight zero, so it can be brought in and evicted at will. Now we request page p_0 immediately after a request for any other page. (Denote the original request times by integers, and the requests for p_0 by half-integers.) Observe that at times $t - 1/2$ when p_0 is requested, the constraints force $(n+1) - (k+1) = n-k$ "real" intervals covering time t to have been chosen, which is precisely what we wanted. Now, suppose time t corresponds to the interval in $\mathcal{I}_{p_t}$ ending, and causing us to request the page p_t. The paging constraint then asks for $n-k$ pages except page p_t to be chosen. But since page p_0 has zero weight, we can choose it, so we need to only choose $n-k-1$ intervals from the rest of the pages except $\{p_0, p_t\}$. Since p_0 will always be chosen, this constraint is implied by the constraint at time $t - 1/2$. So this reduction to paging exactly models the TiledIC problem, and we get an $O(\log k)$-competitive algorithm from [BBN12] again. We summarize the discussion of this section in the following lemma.

LEMMA D.2. *There are randomized online algorithms for the TiledIC and TiledICEx problems that are $O(\log k)$-competitive against oblivious adversaries.*

Appendix E. Proof of Theorem 5.6. Recall the integer program (IP-D3p):

$$\text{(IP-Dp)} \qquad \min \sum_{p,t} w(p)\, x_{p,t} + \sum_I \ell(I) y_I,$$

$$\text{(D2p)} \qquad \sum_{p : p \neq p_t} \min(1, \sum_{t'=\mathcal{T}(p,t)}^{t} x_{p,t'}) \geq (n-k)(1 - y_{I_t}) \qquad \forall t.$$

In this section, we prove the following result.

THEOREM 5.6 (solving the LP online). *There is an $O(\log k)$-competitive online algorithm which maintains a solution to the fractional relaxation of* (IP-D3p). *At time*

t, the fractional solution only changes (and in fact increases) the variables $x_{p,t}$ for all pages p, and y_{I_t}.

For the sake of brevity, we rename y_{I_t} as y_t, ℓ_{I_t} as ℓ_t, $(n-k)$ as R and the interval $[\mathcal{T}(p,t),t]$ as $I(p,t)$: the only fact we use about this interval is that $I(p,t)$ always moves to the right (Claim 5.8). We can now rewrite the linear relaxation of the above IP as

$$\text{(LP-p)} \qquad \min \sum_{p,t} w(p)\, x_{p,t} + \sum_t \ell_t y_t,$$

$$\text{(E.1)} \qquad \sum_{p:p\neq p_t} \min(1, \sum_{t'\in I(p,t)} x_{p,t'}) + R y_t \geq R \qquad \forall t,$$

$$\text{(E.2)} \qquad x_{p,t}, y_t \geq 0 \qquad \forall p,t.$$

Observe that the cost of all $x_{p,t}$ variables corresponding to the same page p is the same. If the penalty costs $\ell_t = \infty$ we get the hard covering problem. The following algorithm is a simple extension of a result of Bansal, Buchbinder, and Naor [BBN10]; we give it here for the sake of completeness.

All the variables are initialized to 0. For an interval I and page p, let $x_{p,I}$ denote $\sum_{t\in I} x_{p,t}$. Let $\delta = \frac{1}{k+1}$. The algorithm is simple: at each time t, if the corresponding constraint for time t is violated, then we raise some variables. Imagine this happening via a continuous process, with a clock starting at $\tau = 0$ and continuously increasing until the constraint is satisfied. Let $P_\tau = \{p \mid p \neq p_t, x_{p,I(p,t)} < 1\}$ be the pages in this constraint that are "active" at clock value τ, i.e., the variables $x_{p,I(p,t)}$ are not already at their maximum value. We must have $|P_\tau| \geq k+1$, otherwise the LHS of the constraint would have $R = n-k$ of the $x_{p,I(p,t)}$ values already at 1, and the constraint would be satisfied. Now raise the variables $x_{p,t}$ for every page p in P_τ at the following rate:

$$\frac{dx_{p,t}}{d\tau} = \frac{x_{p,I(p,t)} + \delta}{w(p)}.$$

Also,

$$\frac{dy_t}{d\tau} = \frac{y_t R + \delta(|P_\tau| - k)}{\ell_t}.$$

Note that we raise only the last variable $x_{p,t}$ for each interval $I(p,t)$, but it is raised proportional to the value of the entire interval. As these values rise, more pages fall out of the set P_τ until the constraint is satisfied.

To show the competitiveness, let x^* denote the optimal integer solution to (IP-D3p) after satisfying the constraint for time t. Also, the interval $I_{p,t}$ is not defined for $p = p_t$, we define it for the sake of analysis to be same as $I(p,t')$ where $t' < t$ is the most recent time such that $p \neq p_{t'}$. Now let the potential be

$$\Phi_t := 3 \sum_{p:x^*_{p,I(p,t)}\geq 1} w(p) \log\left(\frac{1+\delta}{\min(1, x_{p,I(p,t)}) + \delta}\right) + 3 \sum_{s\leq t: y^*_s = 1} \ell_s \log\left(\frac{1+\delta}{y_s + \delta}\right).$$

Note that each term in the potential is nonnegative. We show that the amortized cost of the algorithm with respect to this potential can be paid for by the optimal cost times $O(\log(1 + 1/\delta))$.

First, suppose the constraint at time t is revealed. This causes the current intervals $I(p,t)$ to possibly change, and hence some terms from the potential may disappear

(since current intervals only move to the right). But dropping terms can only decrease the potential function. Next, let OPT augment its solution. Suppose OPT decides to set $y_t^* = 1$; then OPT's cost is ℓ_t, whereas the potential goes up by $3\ell_t \log(\frac{1+\delta}{\delta})$. Moreover, for each variable $x_{p,t}^*$ that is set to 1 (there is no reason to raise any other variable), the cost to OPT is $w(p)$, whereas the potential increase is at most $3w(p)\log(\frac{1+\delta}{\delta})$. Hence we have

$$\Delta\Phi \le \Delta OPT \cdot 3\log(1 + 1/\delta).$$

Observe that since the current intervals $I(p,t)$ only move rightward, we charge each optimal variable only once.

Finally, the algorithm moves via the continuous process above. The instantaneous cost incurred by the algorithm is

$$\frac{dALG}{d\tau} = \sum_{p \in P_\tau} w_i \frac{dx_{p,t}}{d\tau} + \ell_t \frac{dy_t}{d\tau} \tag{E.3}$$

$$= \sum_{i \in P_\tau} (x_{p,I(p,t)} + \delta) + R\, y_t + (|P_\tau| - k)\delta \tag{E.4}$$

$$= \sum_{p \in P_\tau} x_{p,I(p,t)} + Ry_t + \delta\ (|P_\tau| + |P_\tau| - k) \tag{E.5}$$

$$< R - (R - |P_\tau|) + \delta\ (|P_\tau| + |P_\tau| - k) \tag{E.6}$$

$$< (|P_\tau| - k) + 2|P_\tau|\,\delta. \tag{E.7}$$

But since $|P_\tau| \ge k+1$ and $\delta = \frac{1}{k+1}$, we get $|P_\tau|\,\delta \le |P_\tau| - k$; this is the first time we use the value of δ. Hence

$$\frac{dALG}{d\tau} \le 3(|P_\tau| - k).$$

Finally, using the chain rule and the definition of the continuous process, the decrease in potential is

$$-\frac{d\Phi}{d\tau} = 3 \sum_{p \in P_\tau : x^*_{p,I(p,t)} \ge 1} \frac{w(p)}{x_{p,I(p,t)} + \delta} \cdot \frac{x_{p,I(p,t)} + \delta}{w(p)} + 3\frac{\ell_t}{y_t + \delta} \cdot \frac{Ry_t + (|P_\tau| - k)\delta}{\ell_t} \cdot \mathbf{1}_{y_t^* = 1}$$

$$\ge 3 \cdot \#\{p \in P_\tau \mid x^*_{p,I(p,t)} \ge 1\} + 3(|P_\tau| - k) \cdot \mathbf{1}_{y_t^* = 1}.$$

The first equality above uses the fact that for pages in P_τ, $x_{p,I(p,t)}$ is strictly less than 1, and so the truncation by 1 does not have any effect. Now either $y_t^* = 1$, in which case the second term gives us $3(|P_\tau| - k)$, or $y_t^* = 0$ and the second term is not present, but then x^* is a feasible solution to the covering constraint (E.2) at time t. Therefore, at most $n - R = k$ intervals $I(p,t)$ are not hit by x^*. So the contribution of the first term in this case is at least $3(|P_\tau| - k)$. Putting these together, we get that $-\frac{d\Phi}{d\tau} \ge 3(|P_\tau| - k)$, and hence

$$\frac{d}{d\tau}(ALG + \Phi) \le 0.$$

This shows $\log(1 + 1/\delta)$-competitiveness. Using the setting of $\delta = \frac{1}{k+1}$ completes the proof of Theorem 5.6.

Acknowledgment. We thank Ravishankar Krishnaswamy for valuable discussions about this problem; many of the ideas here arose in discussions with him.

REFERENCES

[AAC+17] I. Ashlagi, Y. Azar, M. Charikar, A. Chiplunkar, O. Geri, H. Kaplan, R. Makhijani, Y. Wang, and R. Wattenhofer, *Min-cost bipartite perfect matching with delays*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, K. J. José, D. P. Rolim, D. Williamson, and S. S. Vempala, eds., LIPIcs Leibniz Int. Proc. Inform. 81, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 1:1–1:20.

[ACK17] Y. Azar, A. Chiplunkar, and H. Kaplan, *Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays*, in Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, Barcelona, 2017, pp. 1051–1061.

[AF20] Y. Azar and A. J. Fanani, *Deterministic min-cost matching with delays*, Theory Comput. Syst., 64 (2020), pp. 572–592.

[AGGP17] Y. Azar, A. Ganesh, R. Ge, and D. Panigrahi, *Online service with delay*, in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, Montreal, 2017, pp. 551–563.

[AT19] Y. Azar and N. Touitou, *General framework for metric optimization problems with delay or with deadlines*, in Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science, D. Zuckerman, ed., Baltimore, MD, IEEE, 2019, pp. 60–71.

[AT20] Y. Azar and N. Touitou, *Beyond tree embeddings—a deterministic framework for network design with deadlines or delay*, in Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science, Durham, NC, Sandy Irani, ed., IEEE, 2020, pp. 1368–1379.

[BBB+16] M. Bienkowski, M. Böhm, J. Byrka, M. Chrobak, C. Dürr, L. Folwarczný, L. Jez, J Sgall, N. K. Thang, and P. Veselý, *Online algorithms for multi-level aggregation*, in Proceedings of the 24th Annual European Symposium on Algorithms, Aarhus, Denmark, 2016, pp. 12:1–12:17.

[BBF+01] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, *A unified approach to approximating resource allocation and scheduling*, J. ACM, 48 (2001), pp. 1069–1090.

[BBN10] N. Bansal, N. Buchbinder, and J. Naor, *A Simple Analysis for Randomized Online Weighted Paging*, manuscript, 2010.

[BBN12] N. Bansal, N. Buchbinder, and J. Naor, *A primal-dual randomized algorithm for weighted paging*, J. ACM, 59 (2012), pp. 19:1–19:24.

[Bel66] L. A. Belady, *A study of replacement algorithms for virtual-storage computer*, IBM Systems J., 5 (1966), pp. 78–101.

[BFNT17] N. Buchbinder, M. Feldman, J. Naor, and O. Talmon, *O(depth)-competitive algorithm for online multi-level aggregation*, in Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, Barcelona, 2017, pp. 1235–1244.

[BKL+13] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko, *Online make-to-order joint replenishment model: Primal-dual competitive algorithms*, Oper. Res., 61 (2013), pp. 1014–1029.

[BKS18] M. Bienkowski, A. Kraska, and P. Schmidt, *Online service with delay on a line*, in Structural Information and Communication Complexity—25th International Colloquium, SIROCCO 2018, Z. Lotker and B. Patt-Shamir, eds., Lecture Notes in Comput. Sci. 11085, Springer, New York, 2018, pp. 237–248.

[CBD+15] M. Claeys, N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, S. Latré, and F. De Turck, *An announcement-based caching approach for video-on-demand streaming*, in Proceedings of the 11th International Conference on Network and Service Management, 2015, pp. 310–317.

[CC06] M. Chlebík and J. Chlebíková, *Complexity of approximating bounded variants of optimization problems*, Theoret. Comput. Sci., 354 (2006), pp. 320–338.

[CK99] E. Cohen and H. Kaplan, *Lp-based analysis of greedy-dual-size*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, 1999, pp. 879–880.

[CKPV91] M. Chrobak, H. J. Karloff, T. H. Payne, and S. Vishwanathan, *New results on server problems*, SIAM J. Discrete Math., 4 (1991), pp. 172–181.

[DDH+12] K. De Schepper, B. De Vleeschauwer, C. Hawinkel, W. Van Leekwijck, J. Famaey, W. Van de Meerssche, and F. De Turck, *Shared content addressing protocol (SCAP): Optimizing multimedia content distribution at the transport layer*, in Proceedings of the IEEE Network Operations and Management Symposium, 2012, pp. 302–310.

[EKW16] Y. Emek, S. Kutten, and R. Wattenhofer, *Online matching: Haste makes waste!*, in Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, Cambridge, 2016, pp. 333–344.

[FKL+91] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, *Competitive paging algorithms*, J. Algorithms, 12 (1991), pp. 685–699.

[KKR03] A. R. Karlin, C. Kenyon, and D. Randall, *Dynamic TCP acknowledgment and other stories about e/(e-1)*, Algorithmica, 36 (2003), pp. 209–224.

[Lin] *Deadline Task Scheduling*, The Linux Kernel https://www.kernel.org/doc/html/latest/scheduler/sched-deadline.html.

[NS09] T. Nonner and A. Souza, *Approximating the joint replenishment problem with deadlines*, Discrete Math. Algorithms Appl., 1 (2009), pp. 153–174.

[ST85] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Commun. ACM, 28 (1985), pp. 202–208.

[You91] N. E. Young, *On-line caching as cache size varies*, in Proceedings of the 2nd Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, San Francisco, 1991, pp. 241–250.