Caching with Time Windows

Anupam Gupta
anupamg@cs.cmu.edu
Computer Science Department,
Carnegie Mellon University
Pittsburgh, PA, USA

Amit Kumar
amitk@cse.iitd.ac.in
Department of Computer Science and Engineering,
IIT Delhi
New Delhi, India

Debmalya Panigrahi
debmalya@cs.duke.edu
Department of Computer Science,
Duke University
Durham, NC, USA

ABSTRACT

We consider the (weighted) Paging with Time Windows (PageTW) problem, which is identical to the classical weighted paging problem but where each page request only needs to be served by 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. From a theoretical perspective, this generalizes the caching problem to allow delayed service, a line of work that has recently gained traction in online algorithms. (e.g., Emek et al. STOC '16, Azar et al. STOC '17, Azar and Touitou FOCS '19, etc.).

Our main result is an $O(\log k \log n)$-competitive algorithm for the PageTW problem on n pages with a cache of size k. This significantly improves on the previous best bound of $O(k)$ (Azar et al. (STOC '17).

We also consider the offline PageTW problem, for which we give an $O(1)$ approximation algorithm and prove APX-hardness. These are the first results for the offline problem; 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 algorithm with delays/deadlines.

CCS CONCEPTS

• **Theory of computation → Approximation algorithms analysis**; **Caching and paging algorithms**.

KEYWORDS

Online caching, approximation algorithms

ACM Reference Format:
Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. 2020. Caching with Time Windows. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20), June 22–26, 2020, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3357713.3384277

STOC '20, June 22–26, 2020, Chicago, IL, USA

ACM ISBN 978-1-4503-6979-4/20/06…$15.00
https://doi.org/10.1145/3357713.3384277

1 INTRODUCTION

Caching/paging is one of the most widely studied problems in online algorithms. In this problem, page requests from a universe of n pages arrive over time, and 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, and the goal is to minimize the sum of weights of evicted pages. But, in some situations, page requests do not need to be served immediately and can be delayed up to a given deadline. For instance, in mixed workload environments such as in cloud computing or operating systems, requests from time-sensitive applications (such as interactive ones) have short deadlines, but batch processes can 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 [1].) 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., [13, 15]). 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 call this the *(weighted) Paging with Time Windows* (PageTW) problem.

Our main result is an $O(\log k \log n)$-competitive algorithm for the PageTW problem. This significantly improves upon the previous best deterministic bound of $O(k)$ due to Azar *et al.* [4]. Furthermore, since PageTW generalizes the classical paging problem, a natural lower bound on the competitive ratio is $\Omega(\log k)$, which we match up to the factor of $O(\log n)$. At the heart of our algorithm is a novel "hitting-set" LP relaxation of the PageTW problem that overcomes the $\Omega(k)$ integrality gap (see Section A.2) of the natural LP relaxation for this problem. 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., [3–5, 9, 10, 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 now state our main theorem.

Theorem 1.1 (Online Algorithm). *There is an $O(\log k \log n)$-competitive randomized algorithm for the PageTW problem in the online setting, where n is the number of pages and k is the size of the cache.*

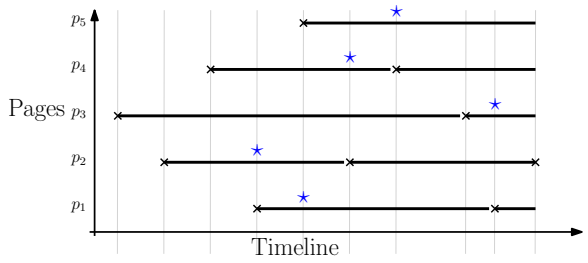

Figure 1: A two-dimensional view of page requests and evictions. The crosses represent page requests and stars represent page evictions. This illustration is for a cache of size 3.

We also study the *offline* version of the PageTW problem, where the request sequence is given up-front. The first question is tractability: since weighted paging is solvable in polynomial time offline, it is conceivable that so is PageTW. We show that the PageTW problem is APX-hard. We complement this lower bound with an $O(1)$-approximation for the offline PageTW problem.

Theorem 1.2 (Offline Algorithm). *The PageTW problem is NP-hard (and APX-hard) even when the cache size $k = 1$, and the pages have unit weight. Moreover, there is an $O(1)$-approximation algorithm, based on rounding a linear program to show a constant integrality gap.*

1.1 Our Techniques

To begin with, recall the "interval covering" IP formulation for the weighted paging problem [6, 20]:

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

For every page p, define an interval starting at each request for it, and ending just before the next request. Because of the 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 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 2-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 Fig. 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 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 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. E.g., if a heavy page is being evicted, we should serve some outstanding light requests while there is an empty slot in the cache (see Section A.1).

The Hitting Set IP Relaxation. To overcome these challenges, we first re-interpret the interval covering IP for classical paging. We again use $x_{p,t}$ variables (saying 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: 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 (non-canonical) *last* request before time t'. Extending the previous intuition, we define the following intervals for time t': corresponding to a request 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 rightwards 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 Fig. 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

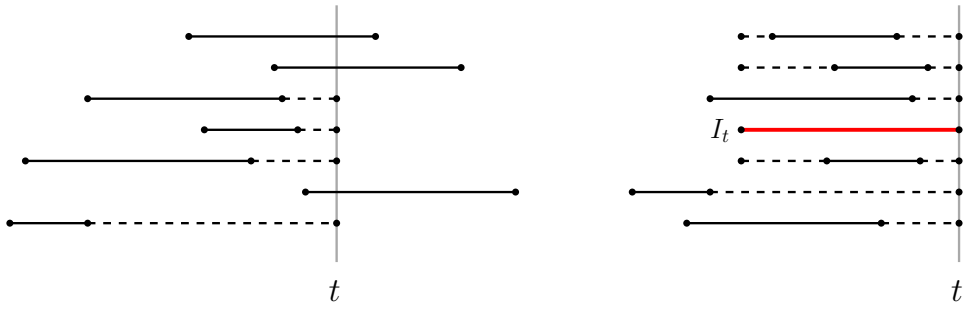

Figure 2: The left figure illustrates right extensions $\mathrm{Rext}(I,t)$ for request intervals I shown by solid lines. The dotted lines show how these intervals are extended. The figure on the right shows double extensions $\mathrm{Dext}(I,t)$, with the critical interval I_t in red.

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 $(\min(s, s(I)), t_p)$. 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 sub-interval (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 Fig. 2). Our "hitting set" IP comprises these two sets of requests, for right extensions and double extensions. We give details of this formulation in §2.

Solving the Hitting Set IP Online. Loosely, we extend ideas from Bansal *et al.* [6] for solving the weighted paging IP online to our hitting set IP. There are some challenges, however. Firstly, the hitting set IP is of exponential size, since we wrote covering constraints for every choice of request interval for every page. Secondly, 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. Thirdly, 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 "non-nested", the techniques of [6] 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 non-nested; 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 §4.

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 either represent insertion or eviction of the page from the cache. Moreover, the IP solution does not necessarily give *all* the insertions and evictions. E.g., 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 Section A.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 §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. [4] 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 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 [5].) A related problem is *online multilevel aggregation* [9] 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 [10], improving on [9]; the model itself combines elements of TCP acknowledgment [18] and online joint replenishment [11]. Online problems with delays were first proposed by Emek et al. [16] for online matching; see [2, 3] 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 [8]; in contrast, the offline PageTW problem is APX-hard. We know deterministic k-competitive and randomized $O(\ln k)$-competitive algorithms; both are optimal [17, 19]. *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 [12]. Bansal et al. [6] 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 [7, 14], 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. Then we turn to solving the IP online in Section 4. We defer the proof of the APX-hardness of PageTW to the full version, and give some illustrative examples in Section A. Finally, we give the offline algorithm for PageTW in Section B.

2 PROBLEM DEFINITION AND IP RELAXATION

Formally, there is a universe of n pages. The cache can hold k pages at any time. Each page p incurs a cost when we evict it from the cache, which is denoted by its *weight* $w(p) \geq 0$. Each request specifies a page p and an interval $I = [s(I), t(I)]$: the page p has to 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 $I := [s(I), \ldots, e(I)]$. Note that in the traditional paging problem, each interval I contains a single timestep, 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 more restricted non-clairvoyant setting where the deadline is revealed when it is reached [4].

We now 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, (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 the page p is "hit" at time t: 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, since we can remove timesteps with no deadlines, and split times with multiple intervals ending at it. Hence each request interval I corresponds to a unique page $\text{page}(I) \in [n]$. 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 these constraints (See Figure 2.):

if $s(I) \leq t \implies$
right extension of I $\text{Rext}(I, t) := [s(I), \ldots, \max(t, e(I))]$.
if $e(I) \leq t \implies$
double extension of I $\text{Dext}(I, t) := [\min(s(I_t), s(I)), \ldots, t]$.

The "Hitting Set IP" is the following for variables $x_{pt} \in \{0, 1\}$:

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

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

$$\sum_{I \in C} \sum_{t' \in \text{Dext}(I,t)} x_{\text{page}(I),t'} \geq 1 \;\forall t \,\forall C \text{ with } k-1 \text{ requests for distinct pages (excluding } p_t) \text{ ending before } t \tag{D1}$$

In the appendix, we show both these sets of constraints are valid:

Claim 1. *The constraints (R1) and (D1) are valid for any solution to PageTW.*

3 SOLVING PAGETW ONLINE USING ONLINE SOLUTION TO (IP)

Now that we have the IP, we need to solve it (it has an exponential number of constraints), and show how to convert a solution to this IP into one for the PageTW problem. Indeed, (IP) does not have any explicit capacity constraints, so we need to extract a "schedule" from this solution in an online manner. In this section we show the latter step; we will show how to solve the IP in Section 4.

Theorem 3.1. *There is an efficient online algorithm that converts an α-competitive integral solution to (IP) into a valid solution for the PageTW instance, with competitive ratio of $O(\alpha \log n)$.*

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 2-dimensional plane. We list some properties of the online solution $A^\star_t$ to (IP) (which should satisfy all the constraints corresponding to times t and earlier) maintained by the algorithm in Section 4.

(A1) *Monotonicity*: $A^\star_t \subseteq A^\star_{t+1}$ for all t.
(A2) The *past preserving* property: 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) The *sparsity* property: for every page p, $A^\star_t$ contains at most one star (p, t') with $t' > t$. Furthermore, if $A^\star_t$ has such a

star, then it 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 of these 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 $A_t^\star$.

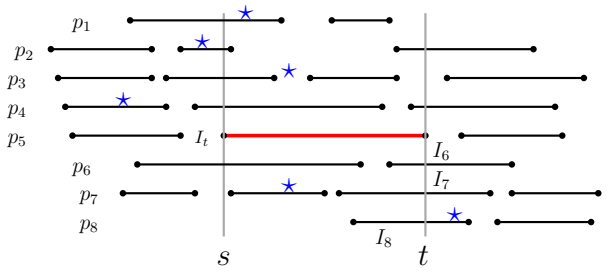

Figure 3: Illustration of the definitions used in Algorithm 1. The request intervals for each page are shown in one horizontal line (these do not overlap in our example). Focus on time t: the cache has pages $C(t) = \{p_1, \ldots, p_4\}$. The page $p_5 = \text{page}(I_5)$ is critical at time t, with 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 $U^\circ = \{I_6, I_7\}$.

3.1 The Online Algorithm for Non-Overlapping Requests

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

1 **foreach** $t = 0, 1, \ldots$ **do**
2 **let** I_t be the interval with deadline t, and let $p_t \leftarrow \text{page}(I_t)$
3 **if** *cache* $C(t)$ *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 $I_t^p \leftarrow$ the request interval I with $\text{page}(I) = p$ and largest ending time $e(I) < t$.
9 **if** $\text{Dext}(I_t^p, t)$ is hit by $A_t^\star$ then add p to $Z^\star$.
10 $U \leftarrow$ unsatisfied request intervals active at time t (one per page, page requests are disjoint).
11 $U^\circ \leftarrow \{I \in U \mid \nexists t' \in I \text{ with } (\text{page}(I), t') \in A_t^\star\}$ be intervals in U **not** hit by $A_t^\star$
12 **serve** and **evict** all requests in $U \setminus U^\circ$.
13 **let** $U^\circ_{\le w}$ and $Z^*_{\le w}$ denote pages in U° and $Z^\star$ respectively with weight at most w.
14 **let** $p^\star$ be a page in $Z^\star$ such that $w(U^\circ_{\le 2w(p^\star)}) \le 2 \cdot w(Z^\star_{\le w(p^\star)})$.
15 **evict** all pages in $Z^\star_{\le w(p^\star)}$.
16 **serve** and **evict** all requests in $U^\circ_{\le 2w(p^\star)}$.
17 **if** I_t not satisfied **then** bring page p_t into cache.

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 §3.2 for the online algorithm for the general case, and Section B for the offline algorithm).

Algorithm 1 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 $\text{Dext}(I_t^p, t)$ is hit by $A_t^\star$. (Since $\text{Dext}(I_t^p, t)$ 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 $\text{Dext}(I_{t'}^p, t')$ for a future time t' may also be hit by the same star in $A_t^\star$. So we evict a subset of $Z^\star$—ones for which we are sure that the corresponding stars of $A_t^\star$ 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, else we can charge the eviction to the 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 $A_t^\star$ (lines 10–11)—by the sparsity property, these are the pages p for which $A_t^\star$ 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 $A_t^\star$ and can be directly paid for (because of the non-overlapping intervals). It is trickier to decide which requests in U° to service. In Lemma 3.2 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 stars in X of weight at

most a. We service all the requests in $U^{\circ}_{\leq 2w(p^\star)}$ and evict the pages $Z^\star_{\leq w(p^\star)}$. This ensures that all the remaining unsatisfied requests are much heavier than the current pages remaining in the cache. By the observation at the start of this paragraph, the stars in $A^\star_t$ which are being charged for the eviction of $Z^\star_{\leq w(p^\star)}$ are not going to be charged again.

Finally, we serve I_t by bringing $p_t = \text{page}(I_t)$ 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.

3.1.1 The Analysis. We first need some supporting claims to show that the algorithm is well-defined, and then bound the cost. The proof of the following claim is deferred to the appendix.

Claim 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 $t_1 < s(I) \leq e(I) \leq t_2$.*

We now show that the algorithm is well-defined.

Claim 3. *The set $Z^\star$ defined in lines 6–9 is non-empty.*

Proof. For each page $p \in C(t)$, let I^p_t be the request interval defined in line 8—such a request interval exists because of Claim 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 $A^\star_t$, and hence the corresponding page belongs to $Z^\star$. □

Lemma 3.2. *There exists a page $p^\star \in Z^\star$ such that*

$$w(U^{\circ}_{\leq 2w(p^\star)}) \leq 2\, w(Z^\star_{\leq 2w(p^\star)}).$$

Proof. We first claim that $|U^\circ| \leq |Z^\star|$. 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^p_t for page p as defined in line 8. Since $p \notin Z^\star$, we must have $(p, t') \notin A^\star_t$ for all times $t' \in \text{Dext}(I^p_t, t)$. But since the interval ends before t, we get $\text{Rext}(I^p_t, t) \subseteq \text{Dext}(I^p_t, t)$ and so $A^\star_t$ does not hit the right-extended interval for I^p_t 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 $A^\star_t$, so by constraint (R1) this collection has size at most k. This proves that $|U^\circ| \leq |Z^\star|$.

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 $w(p') \leq 2\, w(p)$. If this graph has a perfect matching, then $w(B) \leq 2\, w(A)$. We choose $p^\star$ to be the highest-weight page in $Z^\star$.

Else 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, $w(B') \leq 2\, w(A' \setminus \{a\}) \leq 2w(A')$. Now choose $p^\star$ to be the highest weight page in A', to get $w(U^{\circ}_{\leq 2w(p^\star)}) \leq 2\, w(Z^\star_{\leq w(p^\star)})$. □

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, whose proof is deferred to the appendix, shows that the request interval I_t gets served.

Claim 4. *Suppose I_t is unsatisfied at time t. If $w(p_t) \leq 2w(p_{\min})$, then the page p_t belongs to either $U \setminus U^\circ$ in line 12 or to $U^{\circ}_{\leq 2w(p^\star)}$ in line 16, and is served and evicted. Else p_t is served by line 17, and remains in the cache.*

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. Else, if the unsatisfied intervals crossing time t which are hit by $A^\star_t$ 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 $A^\star_t$: 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 (since bringing in pages is for free). We maintain the invariant that each page p in the cache has at most $w(p)$ "load" on it; pages outside the cache have zero load. The load measures the evictions which have not been paid for till now. If we bring in p_t and if $w(p_t) \geq 2w(p_{\min})$, its load becomes the load of $p_{\min}$ plus the cost of evicting $p_{\min}$; thus the total load on p_t is at most its weight, and p_t remains in the cache, maintaining the invariant. 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. This adds one to the competitive ratio. Else if $w(p_t) < 2w(p_{\min})$, we evict at least one page in $Z^\star$ (by Claim 3) and can charge evicting $p_{\min}$ to the eviction of that page—which we show below how to charge to $A^\star_T$.

Next: we charge evicting $U \setminus U^\circ$ to $w(A^\star_T)$ as follows. Each interval in $U \setminus U^\circ$, say for page p, contains some star at (p, t') in $A^\star_t$ (and hence in $A^\star_T$, and we can charge to star). Since the request intervals for a page are disjoint (by our simplifying assumption), (p, t) cannot lie in any other request interval for p, and will not be charged by line 12 again.

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 I in $Z^\star_{\leq w(p^\star)}$, the doubly-extended interval $\text{Dext}(I^p_t, t)$ is hit by an star at $(\text{page}(I), t') \in A^\star_t$ for some $t' \leq t$, so we want to charge to this star of $A^\star_t$. Moreover, each page in $Z^\star_{\leq w(p^\star)}$ is at least as heavy as $p_{\min}$, so any of these stars of $A^\star_t$ can pay to evict $p_{\min}$ (and its load). We finally show that no star of $A^\star_T$ can be charged twice in this manner.

Lemma 3.3. *No star in $A^\star_T$ can be charged twice because of evictions in lines 15-16.*

Proof. For a contradiction, suppose an 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 pages 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 2 shows that $I^q_{t_2}$ starts after t_1. But we know that $t_q \le t_1$, 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 $\mathrm{Dext}(I^q_{t_2}, t_2)$, 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 $\mathrm{page}(I_{t_2})$. Then $w(p_{t_2}) \le 2w(q)$, else 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. □

This proves Theorem 3.1 (without losing the extra $\log n$ factor) in the case of non-overlapping requests for any page p. The general case gets trickier. Indeed, consider the example with a page p having request intervals $[t_1, t], [t_2, t], \ldots, [t_k, t]$, where $t_1 < t_2 < \ldots < t_k < t$. 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 off-line case (which appears in Section B), one can add a *reverse delete* step, where we look at all these times when we service some of these requests, and realize that a subset of them would suffice. However, we discuss the more involved online case in the next section.

3.2 Online Algorithm for the General Setting

The algorithm from Section 3.1 assumes the requests for a page are non-overlapping. We now extend it to handle overlapping requests in an online fashion. Algorithm 2 gives the online algorithm. We call a request interval I *non-dominating* if it does not contain another request interval for $page(I)$—we know whether I is non-dominating only at time $e(I)$. Notice that the definition of I^p_t in line 8 looks only at non-dominating 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 $A^\star_t$ before time t, and some small set of requests that are hit by $A^\star_t$ after time t. As shown in the example at the end of Section 3.1 serving all such requests may lead to unbounded number of chargings 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).

It is also worth noting 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 3.2 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. Indeed, for an unsatisfied request I_t at its deadline

Algorithm 2: ConvertOnline((IP) solution $A^\star_t$ appearing online)

1 **foreach** $t = 0, 1, \ldots$ **do**
2 **let** I_t be the interval with deadline t, and let $p_t \leftarrow \mathrm{page}(I_t)$
3 **if** *cache* $C(t)$ *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 $Z^\star \leftarrow \emptyset$.
7 **for** *every page* p *in* $C(t)$ **do**
8 $I^p_t \leftarrow$ non-dominating request interval I with $\mathrm{page}(I) = p$ and largest ending time $e(I) < t$.
9 **if** $\mathrm{Dext}(I^p_t, t)$ is hit by $A^\star_t$ then add I^p_t 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 $(\mathrm{page}(I), t') \in A^\star_t\}$ be intervals in U **not** hit by $A^\star_t$.
12 **if** $I_t \notin U^\circ$ **then**
13 $U^\star_t \leftarrow$ request intervals I in $(U \setminus U^\circ)$ which are hit by $A^\star_t$ at some time $\le t$.
14 **serve** and **evict** all requests in $U^\star_t$.
15 **evict** the cheapest page $p^\dagger$ in $Z^\star$ (this may be the same as $p_{\min}$)
16 **sort** intervals in $U \setminus (U^\circ \cup U^\star_t)$ 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.

t, either p_t has twice the weight of $p_{\min}$ and is handled in line 22. Else, either the request interval I_t is hit by $A^\star_t$ and so it belongs to $U^\star_t$ and is served/evicted in line 14, or it is not hit by $A^\star_t$ and so belongs to $U^\circ_{\le 2w(p^\star)}$ (by Claim 4) and is served/evicted in line 21. 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 $w(A^\star_T)$. 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 Section 3.1. Indeed, paying for $p_{\min}$ and its load is done by either putting a load on p_t if $w(p_t) \ge 2w(p_{\min})$ 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 $2w(A^\star_T)$, since the proof for Lemma 3.3 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 $Z^\star_{\le w(p^\star)}$, and these stars are never charged again due to Lemma 3.3.

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 $p^\dagger_t, p^\star_t$ denote the pages chosen in lines 15 and 19. The evictions in line 14 are easy to pay for, and we defer the proof of the following statement to the appendix.

Claim 5. $\cup_{t\in\mathfrak{T}} w(U^\star_t) \le w(A^\star_T)$.

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 15,17. 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 $w(P^\star_t[a, b]) \le w(A^\star_t[a, b])$ for any time t and interval $[a, b]$.

Claim 6. *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 the effective cost at time t_1 is at most at most $\frac{5}{2}\, w(P^\star_{t_2}[t_1, t_2])$.*

Proof. By design, $w(U^\dagger_{t_1}) \le 4w(p^\dagger_{t_1})$, so the effective cost at time t_1 is at most $5w(p^\dagger_{t_1})$. Thus it suffices to bound $w(p^\dagger_{t_1})$. Since $t_2 \in \mathfrak{T}$, the interval I_{t_2} was not served at time t_1. The possible reasons are:

(1) The interval $I_{t_2} \in U^\circ_{t_1}$ and $w(p_{t_2}) > 2\, w(p^\star_{t_1})$. Since $w(p^\dagger_{t_1}) \le w(p^\star_{t_1})$ 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 star at (p_{t_2}, t') in $A^\star_{t_2}$ for some $t' \in [t_1, t_2]$. Therefore, $w(p_{t_2}) \le w(P^\star_{t_2}[t_1, t_2])$.

(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 earliest-deadline-first 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 7. *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 an star for $(p^\dagger, t')$ in $A^\star_{t_2}$ for some $t' \in (t_1, t_2]$.*

Proof. Page $p^\dagger$ is evicted at time t_1, and so it must have been brought in by an unsatisfied request I; this request must start after t_1 (else it would be satisfied at t_1) and end 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 is either itself non-dominating, or contains a non-dominating request for $p^\dagger$. In either case, there is a non-dominating 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 non-dominating.

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

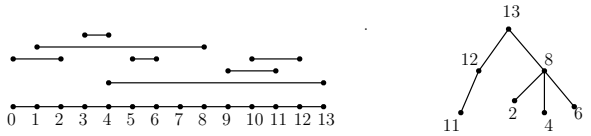

Figure 4: Illustration of F: the intervals on the left are I_t for $t \in \mathfrak{T}$ (note that these intervals are identified using their right end-points). The corresponding forest F is shown on the right.

The Charging Forest. Motivated by Claims 6 and 7, 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 out-degree). The following lemma gives some natural properties of the forest F; the proof is deferred to the appendix.

Lemma 3.4. *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 then any node t'' in the subtree rooted at t' satisfies $t'' > t$.

We now divide pages and times into classes. For each class c, we will consider a sub-forest 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 $p^\dagger_t$ is of class c. Let V^c be the vertices of class c in F. Let F^c be the minimal sub-graph 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 $A^\star_T(c)$ be the total weight of the stars in $A^\star_T$ 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 8. *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 $O(A^\star_T(c))$.*

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 will be an ancestor of t_{i-1} (Lemma 3.4). Claim 7 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. Since all the leaf nodes in F^c belong to class c, he result follows. □

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

Claim 9. *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 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 6 and 8 show 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 6. 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 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. Let $p_{t_1}^\dagger$ be of class c. Then the effective cost at time t_1 is at most at most*

$$10\left(\sum_{c'=c-\log n-3}^{c} w(A_T^\star(c', [t_1, t_2])) + \sum_{c'>c} \frac{w(A_T^\star(c', [t_1, t_2]))}{2^{c'-c}}\right).$$

Proof. Claim 6 shows that $w(P_T^\star[t_1, t_2])$ is at least $\frac{w(p_{t_1}^\dagger)}{2}$. Let P' be the pages of weight at most $w(p^\dagger)/4n$ in $P_T^\star[t_1, t_2]$—since the total weight of these pages is at most $w(p^\dagger)/4$. Thus the pages of class $c - \log n - 2$ and higher contribute at least half of $w(P_T^\star[t_1, t_2])$. Further, if $P_T^\star[t_1, t_2]$ contains a page p of weight higher than $w(p)$, then we can just charge it $w(p_{t_1}^\dagger)$ (and may not even charge to other pages in $P_T^\star[t_1, t_2]$). The desired result now follows from Claim 6. □

Claims 8 and 9 along with Claim 10 imply that the total cost incurred during times of class c is a constant times

$$\sum_{c'=c-\log n-3}^{c} w(A_T^\star(c')) + \sum_{c'>c} \frac{w(A_T^\star(c'))}{2^{c'-c}}.$$

Summing over all classes yields Theorem 3.1.

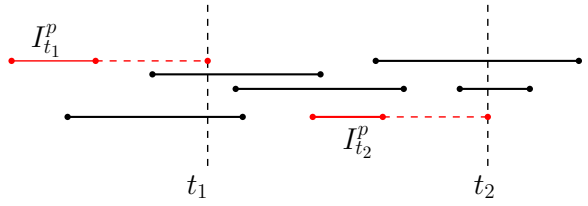

Figure 5: Illustration of I_t^p and R_t^p: the figure shows several request intervals for a page p. The intervals $I_{t_1}^p$ and $I_{t_2}^p$ for two times t_1 and t_2 are shown in solid red lines. These are extended using dotted red lines to $R_{t_1}^p$ and $R_{t_2}^p$ respectively.

4 SOLVING THE INTEGER PROGRAM

We show how to solve the integer program (IP). Not only does it have an exponential number of constraints, it happens to have an unbounded integrality gap. For the toy problem of picking $n - k$ out of n items, all of unit weight, any feasible solution has cost at least $n - k$. Using variables $x_i \in [0, 1]$ indicating if we pick item i, we can write linear constraints for every choice C of $k + 1$ items: $\sum_{i \in C} x_i \ge 1$. Now setting $x_i = \frac{1}{k+1}$ for all i gives a fractional solution of cost $\frac{n}{k+1} \ll n - k$. But the fix is simple: we can replace these exponentially-many constraints by a single constraint $\sum_i x_i \ge n - k$ to get an integral LP. We will emulate this approach and write compact LPs with bounded integrality gaps.

It will be convenient to handle the constraints related to the right and double extensions separately, in §4.1 and §4.2 respectively. This at most doubles the cost of the solution. We will rely on existing algorithms for special instances of interval cover, where we pick intervals of the timeline so that each time is covered by $n - k$ intervals. These special instances are called "tiled" since the intervals come in n groups (one for each page), and the intervals for each group partition the entire timeline. More formally, an input of the *tiled interval cover* problem (TiledIC) is specified as follows: 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. The goal is to pick a minimum weight subset of intervals from $\mathcal{I} = \cup_p \mathcal{I}_p$ such that every time t is covered by at least $n - k$ of these selected intervals. The offline algorithms for this *tiled interval cover* problem rely on total-unimodularity, and the online ones reduce the interval-cover problem to the classical paging; details are deferred to the full version.

4.1 Solving for the Right Extensions Constraints

The idea for solving (IP) is conceptually simple: first, we sparsify the constraints (and use the integrality-gap removal idea above) to get a more compact and better integer program. Then for each page we choose a set of (interval, time) pairs such that the right extensions for these are disjoint and maximal. We extend these extended intervals to cover all of time, view this as a tiled interval cover instance with demand $n - k$, and solve it. Then we show that picking the start and end of each interval chosen in the interval cover solution is also a solution to the constraints (R1).

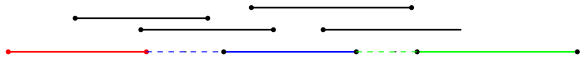

Figure 6: Illustration of $\mathcal{K}_p$: the figure shows several intervals R_t^p for a page p. The colored intervals are the ones selected by the greedy algorithm, and the extended ones (using dashed lines and the matching color) form $\mathcal{K}_p$.

Now for the details: for any page $p \in [n]$ and time t, the interval I_t^p is defined as the request interval $I = [s', t']$ for page p with ending time $t' \le t$, whose start time s' is the largest (see Figure 5 for an example). Now suppose we write the constraints:

$$\sum_{p \in \mathcal{S}} \min\left(1, \sum_{t' \in \mathsf{Rext}(I_t^p, t)} x_{p,t'}\right) \ge 1, \ \forall\, t, \ \forall\, (k+1)\text{-subsets } \mathcal{S} \text{ of pages}$$

This is a subset of the constraints from (R1), and hence the solution to this can only have lower cost. To avoid the integrality gap, we write a single constraint for each time, to get the integer program.

$$\min_{x \in \{0,1\}^{nT}} \sum_{p,t} w_p\, x_{p,t} \qquad \text{(IP-R)}$$

$$\sum_{p} \min\left(1, \sum_{t' \in \mathsf{Rext}(I_t^p, t)} x_{p,t'}\right) \ge n - k \qquad \forall t \qquad \text{(R2)}$$

For brevity, define $R_t^p := \mathsf{Rext}(I_t^p, t)$. Note that each $R_t^p = [s_t, t]$ ends at time t. The following observation is easy to check (recalling that $[a, b]$ strictly contains $[c, d]$ if $a < c \le b < d$).

Claim 11. *For any page p, and times t, t' the interval R_t^p is not strictly contained in $R_{t'}^p$.*

For each page p we build a *maximal* subcollection $\mathcal{K}_p$ of disjoint intervals from $\{R_t^p\}_{t \in [T]}$ as follows:

> Sort these intervals in increasing order of their ending times, and pick an interval if it does not intersect any previously-picked intervals. Then we extend each interval "leftwards" so that this extended interval starts at time $t' + 1$ if the previous interval ends at time t'. Call this collection $\mathcal{K}_p$.

Note that each interval $[t_1, t_2] \in \mathcal{K}_p$ completely contains an interval $R_{t_2}^p$ that ends at time t_2. Let $\mathcal{K} := \cup_p \mathcal{K}_p$ be the set of all these new intervals. Define the weight of each interval in $\mathcal{K}_p$ as $w(p)$, and set the coverage requirement $R := n - k$ for all times t. Observe that all this can be done online, where we start each interval in $\mathcal{K}_p$ when the previous one ends, and end it at some later time t as soon as some interval R_t^p ends and is completely contained within it. This gives an instance $\mathcal{I}$ of tiled interval cover TiledIC, and we now show that we can solve $\mathcal{I}$ to get a solution for SolveRext. As always, an integral solution x to (R2) corresponds to a set $A^\star$ of "stars" (p, t), where $x_{p,t} = 1 \iff (p, t) \in A^\star$.

Lemma 4.1 (Forward Direction). *If $A^\star$ satisfies (R2), there is a solution for TiledIC of weight at most $2w(A^\star)$.*

Proof. We construct a solution $\mathcal{S}$ to $\mathcal{I}$ as follows: for every $(p, t) \in A^\star$, let $K \in \mathcal{K}_p$ be the interval containing time t, and $K' \in \mathcal{K}_p$ be the interval to the right of K. We add K and K' to $\mathcal{S}$. The weight guarantee for $\mathcal{S}$ follows easily.

To show that $\mathcal{S}$ is a feasible solution to $\mathcal{I}$, fix a time $t \in \mathcal{T}$. Since $A^\star$ satisfies (R2), there is set P of $n - k$ pages such that for each page $p \in P$, there is a a star $(p, t_p) \in A^\star$ for some time $t_p \in R_t^p$. Fix a page $p \in P$. Let a be the left endpoint of R_t^p (i.e., $R_t^p = [a, t]$). Let K_a and K_t be the intervals in $\mathcal{K}_p$ containing a and t respectively. We claim that there cannot be an interval in $\mathcal{K}_p$ between K_a and K_t, else it would be completely within R_t^p, and the $R_{t'}^p$ lying within K would violate the not-strictly-nested property from Claim 11. So $K_a \cup K_t$ covers all of the times in R_t^p, and t_p is in K_a or K_t. In either of the two cases, we would have added K_t into $\mathcal{S}$. Thus, $\mathcal{S}$ contains the interval in $\mathcal{K}_p$ containing t for all $p \in P$. This shows that $\mathcal{S}$ is a feasible solution to $\mathcal{I}$. □

Lemma 4.2 (Reverse Direction). *Let $\mathcal{S}$ be a solution to the instance $\mathcal{I}$. Then there is a solution $A^\star$ that satisfies the constraints (R1) and has cost at most $2w(\mathcal{S})$.*

Proof. We construct the solution $A^\star$ as follows: for each interval $K = [t_1, t_2] \in \mathcal{K}_p \cap \mathcal{S}$, we add stars at $(p, t_1), (p, t_2)$ to the solution $A^\star$. The weight guarantee on $A^\star$ follows trivially. To show feasibility, fix a time t and let P be the set of $n - k$ pages p for which the solution $\mathcal{S}$ contains the interval $K_p \in \mathcal{K}_p$ that contains the time t. Hence, for any collection C of $k + 1$ intervals for distinct pages, there must be at least one page p in $C \cap \mathcal{S}$. We claim that the right extension $\mathsf{Rext}(I, t)$ of the corresponding interval is hit by some star in $A^\star$.

Indeed, suppose the interval $K_p \in \mathcal{K}_p \cap \mathcal{S}$ containing t is $K_p = [t_1, t_2]$. For a contradiction assume that $R_t^p := \mathsf{Rext}(I, t)$ starts to the right of t_1 and ends before t_2. In our construction of $\mathcal{K}_p$, the interval K_p was formed by some $R_{t_2}^p$ ending at t_2. But since our procedure picks a maximal set in increasing order of end-times, we would have picked R_t^p instead, which gives the desired contradiction. □

Combining these with an total-unimodularity-based algorithm for solving the interval-cover problem (whose details appear in the full version of the paper), we infer that we can solve (IP-R) in the offline setting losing a factor of 4.

For the *online setting*, the solution from Lemma 4.2 is not satisfactory. Picking the start and end of the intervals chosen by the interval cover procedure (which it itself being run online) causes two problems: (i) At time t, we add an element (p, a) to our solution, where a lies in the past (with respect to t). This violates the past-preserving property required in §3. Thankfully, Lemma 4.2 continues to hold if we add (p, t) to $A^\star$ instead of adding (p, a); this change also makes it past-preserving. (ii) The online algorithm does not know the right end-point b of K at time t, and hence cannot add the star in the future[1]. However, we just carry along this "to-be-added" star, and put it down when we see the end of the interval $K \in \mathcal{K}_p$. We have at most one such star for each page p; moreover, all intervals containing the current time t are already hit by elements added at times $\le t$, or by this to-be-added element. This satisfies the *sparsity* property of §3. To summarize:

[1] The constraints in (IP-R) at time t involve variables in the past only, but those in (R1) involve future as well. So we need to keep track of stars which are going to put in the future at time t.

Lemma 4.3. *There is an online $O(\log k)$-competitive algorithm to solve (IP-R) that satisfies the monotonicity, past-preserving, and sparsity properties required in §3. Moreover, there is an offline algorithm to solve (IP-R) that is a 4-approximation.*

4.2 Solving for the Double Extension Constraints

As in the previous section §4.1, define I_t^p to be the request interval for page p that ends no later than time t, and which has the largest starting time. Similarly, define $D_t^p := \text{Dext}(I_t^p, t)$ to be the double-extension, and write the smaller set of constraints:

$$\min_{x \in \{0,1\}^{nT}} \sum_{p,t} w_p \, x_{p,t} \tag{IP-D}$$

$$\sum_{p: p \neq p_t} \min\left(1, \sum_{t' \in D_t^p} x_{p,t'}\right) \geq n - k \qquad \forall t \in \mathcal{T}. \tag{D2}$$

For ease of notation, we refer to the above IP as SolveDext. Again, we have chosen one double-extended interval for each page and time, and only written the constraints for them. Moreover, we have combined the exponentially-many constraints into one, to reduce the integrality gap. Since there are fewer constraints, any x satisfying constraints (D1) also satisfies (D2). Finally, observe that we allow specifying some set $\mathcal{T}$ of times for which we want a solution: for now, we think of $\mathcal{T} = [T]$, but this generality will be useful.

Unlike Claim 11, it is possible that an interval D_t^p strictly contains another interval $D_{t'}^p$ for some page p. The non-nesting property of R_t^p intervals (for a given page p) was crucial in the proof of Lemma 4.1. Therefore, our solution for (IP-D) is presented in two parts. Two intervals are *non-nested* if neither of them (strictly) contains the other. In a "globally non-nested" case of SolveDext, denoted by NonNestDext, the subset $\mathcal{T}$ has the property that for every two times $t_1, t_2 \in \mathcal{T}$, the corresponding critical intervals I_{t_1} and I_{t_2} are non-nested.[2] We first solve globally non-nested instances in Section 4.2.1 and then extend this solution to the general case in Section 4.2.2.

4.2.1 Solving Globally Non-Nested Cases of SolveDext. We now consider an instance of NonNestDext, where the set of times is given by $\mathcal{T}$ – recall that the intervals $I_t, t \in \mathcal{T}$, are non-nested. As alluded above, we begin by showing that the intervals D_t^p for any fixed page are also non-nested.

Claim 12. *Consider times $t, t' \in \mathcal{T}$ and a page p such that $p \notin \{p_t, p_{t'}\}$. The doubly-extended intervals $\text{Dext}(I_t^p, t)$ and $\text{Dext}(I_{t'}^p, t')$ are non-nested.*

Proof. Suppose $t < t'$, and $D_{t'}^p$ strictly contains D_t^p. Hence $D_{t'}^p$ starts strictly before D_t^p. Since I_t^p ends before t', the starting time of $I_{t'}^p$ is at least that of I_t^p. Moreover, I_t and $I_{t'}$ are themselves non-nested by assumption, so $e(I_{t'}) > e(I_t) \implies s(I_{t'}) \geq s(I_t)$ (recall that $s(I)$ and $e(I)$ denote the starting and the ending time the interval I respectively). This would mean that $D_{t'}^p$ starts no earlier than D_t^p, a contradiction. □

[2] We already assumed that any two intervals for the same page are non-nested, but here we require this non-nestedness property over *all* request intervals regardless of their associated pages.

We now reduce the instance NonNestDext to a tiled interval cover problem with exclusions (TiledICEx) instance: this is like the TiledIC, but for each time t we cannot use the interval ending at t in the cover (or more generally, at each time t, we are given a page p_t, and we cannot use the intervals in $\mathcal{I}_{p_t}$ for the coverage requirement at time t). In the full version, we show that there is a constant factor approximation algorithm and $O(\log k)$-competitive algorithm for NonNestDext.

The reduction to NonNestDext instance proceeds in analogous manner as the reduction to SolveDext in the previous section §4.1. For each page p, we construct a maximal subcollection $\mathcal{K}$ of disjoint intervals for $\{D_t^p\}_{t \in \mathcal{T}}$ in exactly the same manner as the corresponding construction from R_t^p in Section §4.1. Again define $\mathcal{K} := \cup_p \mathcal{K}_p$ and the weights of all the intervals in $\mathcal{K}_p$ is equal to $w(p)$. Define the R at each time to be $n - k$. The proof of the following lemma follows along the same lines as those of Lemmas 4.1 and 4.2.

Lemma 4.4. *Given a NonNestDext instance $\mathcal{I}$, let $\mathcal{I}'$ be the corresponding TiledICEx instance $\mathcal{I}'$ constructed above:*

(i) If a solution $A^\star$ satisfies (D2), there is a solution for $\mathcal{I}'$ of weight at most $2w(A^\star)$.

(ii) Let $\mathcal{S}$ be a solution to $\mathcal{I}'$. Then there is a solution $A^\star$ to $\mathcal{I}$ satisfying the constraints in (D1) of cost at most $2w(\mathcal{S})$.

Moreover, both the above constructions can be done efficiently.

Lemma 4.4 along with the fact that we have a 2-approximation for TiledICEx (which appears in the full version of the paper) implies that there is a 12-approximation algorithm for NonNestDext.

As in Lemma 4.2, the reduction from the proof of Lemma 4.4(ii) can be carried out in an online manner. At each time t, the online algorithm (for NonNestDext) may add some stars at time t and at some points of time $t' > t$. Since the set of constraints (D1) 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*; note that this is a stronger property that past preserving or sparsity (as mentioned in §3. We get

Lemma 4.5. *There is an online $O(\log k)$-competitive present restricted algorithm to NonNestDext. Moreover, there is an offline algorithm 12-approximation algorithm for NonNestDext.*

4.2.2 Algorithm for the General Case of SolveDext. We now consider the general setting where the intervals I_t can be nested. Let $\mathcal{I}$ be a general instance of SolveDext, where we want to handle all times $\mathcal{T}$. We show how to extend a solution for a non-nested sub-instance 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 *non-nested net* if (i) for times $t_1 \neq t_2 \in \mathcal{N}$, their critical intervals I_{t_1}, I_{t_2} are non-nested, 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.

Observe that a non-nested net of set of times $\mathcal{T}$ can be easily constructed by a greedy algorithm which scans the times in $\mathcal{T}$ from left to right, and adds a time t to $\mathcal{N}$ whenever I_t does not contain

$I_{t'}$ for any $t' \in \mathcal{N}$. This procedure is also 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 non-nested net $\mathcal{N}$ of $\mathcal{T}$, we define a map $\varphi : \mathcal{T} \setminus \mathcal{N} \to \mathcal{N}$ as follows – for a time $t \in \mathcal{T} \setminus \mathcal{N}$, $\varphi(t)$ is the right-most time $t' \in \mathcal{N}$ such that I_t contains $I_{t'}$.

Claim 13 (Monotone Map). *Let $\mathcal{T}$ be a set of times, $\mathcal{N}$ be a non-nested net of $\mathcal{T}$ and φ be the associated map as above. Then for any $t'_1, t'_2 \in \mathcal{T} \setminus \mathcal{N}$, $t'_1 < t'_2 \implies \varphi(t'_1) \le \varphi(t'_2)$.*

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 non-nested, it must be the case that $s_1 \ge s_2$. 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 x for SolveDext, 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 whose doubly-extended intervals D^p_t are hit by $A^\star$. I.e., we can rephrase constraint (D2) 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 4.6 (Extension Theorem). *There is an algorithm that takes a set $\mathcal{T}'$ of times, a non-nested 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) $w(B^\star) \le 3\, w(A^\star)$.

This algorithm can be implemented in 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 the full version, and instead explain how to use the result in the off-line setting first. We invoke the extension theorem twice. For the first invocation, we use Thereom 4.6 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 $12\,\mathrm{opt}(\mathcal{I})$ given by Lemma 4.5 on the sub-instance $\mathcal{N}$. This outputs a set $B^\star$ with $w(B^\star) \le 36\,\mathrm{opt}(\mathcal{I})$. Moreover, since $A^\star$ is feasible for $\mathcal{N}'$, it follows from the first property of Theorem 4.6 that $|P(B^\star, t) \setminus \{p_t\}| \ge |P(A^\star, \varphi(t))| - 1 \ge n - k - 1$ 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 4.6 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 4.5 on the sub-instance $\mathcal{N}_1$. This gives us $B^\star_1$ with weight at most $36\,\mathrm{opt}(\mathcal{I})$. 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 4.7 (Feasibility). *For any time t, $|P(B^\star \cup B^\star_1, t) \setminus \{p_t\}| \ge n - k$.*

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 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 $p_1 \ne p_2 \ne p_3$. Recall: we want to show $|P(B^\star \cup B^\star_1, t_1)| \setminus \{p_1\}| \ge n - k$.

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 $p_2 \in P(B^\star, t_2)$. Since $\mathrm{Dext}(I^{p_2}_{t_2}, t_2) = I_2$, the set $B^\star$ hits I_2, and hence I_1 as well: i.e., $P(B^\star, t_1)$ contains p_2.

Since $P(A^\star_1, t_2) \setminus \{p_2\}$ has size $n-k$ (by construction of $A^\star_1$), and $P(B^\star_1, t_1)$ contains $P(A^\star_1, t_2)$, it follows that $P(B^\star_1, t_1) \setminus \{p_2\}$ also has size at least $n-k$. Therefore, $P(B^\star \cup B^\star_1, t_1)$ has size at least $n-k+1$ because $P(B^\star, t_1)$ contains p_2. This implies the lemma. □

Since $w(B^\star \cup B^\star_1) \le 72\mathrm{opt}(\mathcal{I})$, we get a 72-approximation algorithm for SolveDext. 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 4.5 to get a present restricted solution $A^\star_t$ for all $t \in \mathcal{N}$. Theorem 4.6 implies that the present restricted solution $B^\star_t$ can be constructed at time t. Given $B^\star_t$, 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 $(B^\star_1)_t$ for all $t \in \mathcal{T}_1$. Combining this with Lemma 4.5, we get

Lemma 4.8. *There is an online $O(\log k)$-competitive present restricted algorithm to SolveDext. Moreover, there is an offline algorithm 72-approximation algorithm for SolveDext.*

Corollary 4.9. *There is a constant factor approximation algorithm for (IP). Further, there is an $O(\log k)$-competitive online solution which satisfies the past-preserving and sparsity property as in §3.*

Proof. Let $A^\star_1$ and $A^\star_2$ be (offline) solutions for (R2) and (D2) as guaranteed by Lemmas 4.3 and 4.8 respectively. Since $A^\star_1$ satisfies (R1) (Lemma 4.2), so does $A^\star := A^\star_1 \cup A^\star_2$. By definition, $A^\star_2$ satisfies the constraints (D2). Now consider a time t, page $p \ne p_t$ and a request interval for p which ends before t. Then it is easy to check that $\mathrm{Dext}(I, t)$ contains D^p_t. It follows that $A^\star_2$ also satisfies all the constraints in (D1). Thus, $A^\star$ is a feasible solution to (IP).

The online version follows analogously. Since $A^\star_1$ and $A^\star_2$ are past preserving, so is $A^\star$. Also the sparsity of $A^\star_1$ and the fact that $A^\star_2$ does not add any star in the future implies that $A^\star$ also satisfies the sparsity property. □

Corollary 4.9, along with Theorems 3.1 and B.1, implies Theorems 1.1 and 1.2 respectively. The integrality gap of (IP) 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 4.1, 4.2, and 4.4 also hold between the fractional solutions to the corresponding problems.

ACKNOWLEDGMENTS

We thank Ravishankar Krishnaswamy for valuable discussions about this problem; many of the ideas here arose in discussions with him. This research was done under the auspices of the Indo-US Virtual Networked Joint Center IUSSTF/JC-017/2017. AG was supported in part by NSF award CCF-1907820. DP was supported in part by NSF award CCF-1535972, and an NSF CAREER award CCF-1750140.

APPENDIX

A SOME ILLUSTRATIVE EXAMPLES

A.1 Evictions at Endpoints 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: suppose $k = 1$ and there is a very heavy page which is requested at each time, and so we need to have it in the cache at every time. Now there are n unit weight pages, but there request intervals are $[0, n]$, $[1, n+1]$, $[2, n+2]$, ...

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.

A.2 An Integrality Gap for the Interval Hitting LP

We now consider a natural LP relaxation for PageTW which extends that for weighed caching, and show that it has large integrality gap. We have variables $x_{p,J}$ for pages p and intervals $J \subseteq [T]$, indicating that J is 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 timestep, so each timestep 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 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:

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

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

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

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

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

Proof. We show the integrality gap example, the proof that it has the desired properties is deferred to the full version. Suppose we have k "heavy" pages with weight k each, and 1 "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 $\Omega(Tk)$. □

B OFFLINE ALGORITHM FOR PAGETW

In Section 3, we gave an online algorithm for PageTW using an online solution to (IP). We shall now prove the following offline version of Theorem 3.1.

Theorem B.1. *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 $O(\alpha)$.*

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 always remove the outer interval. 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. The first part of the algorithm until line 17 is same as in Algorithm 1. 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 12. Let Sat_p be the requests in Sat that correspond to p (and T_p be the time at which they are served), and Sat'_p be a maximal collection of disjoint intervals in Sat_p. Since each of the intervals 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, and hence $|T'_p| \leq 4|\text{Sat}'_p|$. It is not difficult to show that each interval in Sat_p has non-empty 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 Claim 2–3, and Lemma 3.2 remain unchanged. We restate these here for sake of completeness.

Claim 14. *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 $t_1 < s(I) \leq e(I) \leq t_2$.*

Claim 15. *The set $Z^\star$ defined in lines 6–9 is non-empty.*

Lemma B.2. *There exists a page $p^\star \in Z^\star$ such that*

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

Lemma B.2 shows the existence of page $p^\star$ in line 14. The proof of the following claim is same as that of Claim 4.

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

1 **foreach** $t = 0, 1, \ldots$ **do**
2 **let** I_t be the interval with deadline t, and let $p_t \leftarrow \text{page}(I_t)$
3 **if** *cache* $C(t)$ *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 $I_t^p \leftarrow$ the request interval I with $\text{page}(I) = p$ and largest ending time $e(I) < t$.
9 **if** $\text{Dext}(I_t^p, t)$ is hit by $A^\star$ then add p to $Z^\star$.
10 $U \leftarrow$ unsatisfied request intervals active at time t (one per page, page requests are disjoint).
11 $U^\circ \leftarrow \{I \in U \mid \exists t' \in I$ with $(\text{page}(I), t') \in A^\star\}$ be intervals in U **not** hit by $A^\star$
12 **serve** and **evict** all requests in $U \setminus U^\circ$.
13 **let** $U^\circ_{\le w}$ and $Z^*_{\le w}$ denote pages in U° and $Z^\star$ respectively with weight at most w.
14 **let** $p^\star$ be a page in $Z^\star$ such that $w(U^\circ_{\le 2w(p^\star)}) \le 2 \cdot w(Z^\star_{\le w(p^\star)})$.
15 **evict** all pages in $Z^\star_{\le w(p^\star)}$.
16 **serve** and **evict** all requests in $U^\circ_{\le 2w(p^\star)}$.
17 **if** I_t not satisfied **then** bring page p_t into cache.
18 Sat $\leftarrow$ set of requests serviced in line 12
19 **for** *every page* p **do**
20 $T_p \leftarrow$ set of times when request intervals in Sat for page p are serviced (in line 12)
21 $\text{Sat}'_p \leftarrow$ maximal disjoint collection of request intervals for page p in Sat.
22 $T'_p \leftarrow$ set of times in T_p closest (on either side) to the two end-points of intervals in Sat'_p
23 **cancel** all movements of p into cache at times in $T_p \setminus T'_p$ (during line 12).

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

The following result, whose proof is deferred to the full version, shows 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.

Claim 17. *Every request interval in Sat_p has non-empty intersection with T'_p.*

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

It remains to analyze the cost incurred by the algorithm. This analysis is again very similar to that in Section 3.1.2. We defer the details to the full version.

REFERENCES

[1] The Linux Kernel - Deadline Task Scheduling. https://www.kernel.org/doc/html/latest/scheduler/sched-deadline.html.
[2] I. Ashlagi, Y. Azar, M. Charikar, A. Chiplunkar, O. Geri, H. Kaplan, R. M. 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, August 16-18, 2017, Berkeley, CA, USA*, pages 1:1–1:20, 2017.
[3] Y. Azar, A. Chiplunkar, and H. Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1051–1061, 2017.
[4] 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, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 551–563, 2017.
[5] Y. Azar and N. Touitou. General framework for metric optimization problems with delay or with deadlines. *CoRR*, abs/1904.07131, 2019.
[6] N. Bansal, N. Buchbinder, and J. Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.
[7] 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(5):1069–1090, 2001.
[8] L. A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.
[9] 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 *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 12:1–12:17, 2016.
[10] N. Buchbinder, M. Feldman, J. S. Naor, and O. Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244, 2017.
[11] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Operations Research*, 61(4):1014–1029, 2013.
[12] M. Chrobak, H. J. Karloff, T. H. Payne, and S. Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.
[13] 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 *2015 11th International Conference on Network and Service Management (CNSM)*, pages 310–317, Nov 2015.
[14] E. Cohen and H. Kaplan. Lp-based analysis of greedy-dual-size. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA.*, pages 879–880, 1999.
[15] 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 *2012 IEEE Network Operations and Management Symposium*, pages 302–310, April 2012.
[16] 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, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344, 2016.
[17] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.
[18] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic TCP acknowledgment and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003.
[19] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.
[20] N. E. Young. On-line caching as cache size varies. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 28-30 January 1991, San Francisco, California, USA.*, pages 241–250, 1991.