Non-Linear Paging

Ilan Doron-Arad
Computer Science Department, Technion, Haifa, Israel

Joseph (Seffi) Naor
Computer Science Department, Technion, Haifa, Israel

Abstract

We formulate and study *non-linear paging* - a broad model of online paging where the size of subsets of pages is determined by a monotone non-linear set function of the pages. This model captures the well-studied classic weighted paging and generalized paging problems, and also *submodular* and *supermodular* paging, studied here for the first time, that have a range of applications from virtual memory to machine learning.

Unlike classic paging, the cache threshold parameter k does not yield good competitive ratios for non-linear paging. Instead, we introduce a novel parameter ℓ that generalizes the notion of cache size to the non-linear setting. We obtain a tight deterministic ℓ-competitive algorithm for general non-linear paging and a $o\left(\log^2(\ell)\right)$-competitive lower bound for randomized algorithms. Our algorithm is based on a new generic LP for the problem that captures both submodular and supermodular paging, in contrast to LPs used for submodular cover settings. We finally focus on the supermodular paging problem, which is a variant of online set cover and online submodular cover, where sets are repeatedly requested to be removed from the cover. We obtain polylogarithmic lower and upper bounds and an offline approximation algorithm.

2012 ACM Subject Classification Theory of computation

Keywords and phrases paging, competitive analysis, non-linear paging, submodular and supermodular functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.57

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.13334` [14]

Funding *Joseph (Seffi) Naor*: Supported in part by Israel Science Foundation grant 2233/19 and United States – Israel Binational Science Foundation (BSF) grant 2033185.

1 Introduction

In the well studied *paging* problem, we are given a collection of n pages and a cache that can contain up to k pages simultaneously, where $k < n$. At each time step, one of the pages is requested. If the requested page is already in the cache, the request is immediately served. Otherwise, there is a *cache miss* and the requested page is fetched to the cache; to ensure that the cache contains at most k pages, some other page is potentially evicted. In the most fundamental model, the goal is to minimize the number of cache misses (or equivalently, number of evictions).

In more general models, pages may have different sizes and costs (see, e.g., [1, 5]) and then the sum of the sizes of the pages in cache cannot exceed its capacity. However, a linear function over page sizes that defines cache feasibility fails to capture scenarios with more involved relations between subsets of pages that can reside together in cache. Consider a system in which pages share parts of their memory and then only the missing memory parts of a requested page can contribute to the increase in cache size. This setting can be modeled using a *submodular* function that defines cache feasibility. Another example is a setting in


which items stored in cache have dependencies yielding additional overhead in their mutual storage demand. The *rule caching* problem is one such setting and it has been well studied in networking [12, 46, 37, 21, 25, 39, 24, 17, 9, 35, 26, 16, 45, 34, 33, 47, 13].

Our focus in this paper will be on settings where such non-linear behavior exists. We will further motivate our model in detail in Section 1.2.

1.1 Our Model

Before presenting our model, we give some required preliminary definitions. Let $\mathcal{P}$ be a set and let $f : 2^{\mathcal{P}} \to \mathbb{N}$ be a set function of $\mathcal{P}$. The function f is called *monotone* if for every $S \subseteq \mathcal{P}$ and $S' \subseteq S$ it holds that $f(S') \leq S$. In addition, f is called *submodular* if for every $S, S' \subseteq \mathcal{P}$ it holds that $f(S) + f(S') \geq f(S \cup S') + f(S \cap S')$. Conversely, f is called *supermodular* if for every $S, S' \subseteq \mathcal{P}$ it holds that $f(S) + f(S') \leq f(S \cup S') + f(S \cap S')$.

In this work, we introduce a very general model of paging with an arbitrary function defining cache feasibility. In the *non-linear paging* problem, we are given a collection $\mathcal{P}$ of n pages where each page $p \in \mathcal{P}$ has a fixed eviction cost $c(p)$. We are also given a monotone *feasibility* function $f : 2^{\mathcal{P}} \to \mathbb{N}$ that assigns a value to every subset of pages, indicating their size. Finally, we are given a cache threshold k. As in standard paging, in each time step t there is a request p_t for one of the pages. If p_t is already in cache, the request is immediately served. Otherwise, p_t is fetched to the cache and possibly some subset of pages is evicted to ensure that the set of pages in the cache, denoted by S_t, is feasible, i.e., $f(S_t) \leq k$. The goal is to minimize the total cost incurred from page evictions. The classic paging problem is obtained by setting $f(S) = |S|$ for all $S \subseteq \mathcal{P}$. Other interesting applications of our model are described below.

- *Generalized Paging* [5, 1]. Here the feasibility function is linear; that is, for every $S \subseteq \mathcal{P}$ it holds that $f(S) = \sum_{p \in S} f(p)$, where $f(p)$ is the size of page $p \in \mathcal{P}$.
- *Submodular Paging.* The feasibility function is *submodular*. A natural application of this variant is to settings where pages share memory items (see Section 1.2).
- *Supermodular Paging.* The feasibility function is *supermodular*, implying a submodular cover function for pages remaining out of the cache. This setting effectively captures online submodular covering problems [2, 23, 20]. Supermodular paging will be the main focus of our paper.

Supermodular paging is a variant of *online set cover* [2] and *online submodular cover* [20]. In online set cover, we are given a ground set X and a family $\mathcal{S}$ of subsets of X. Requests for elements of X arrive online; if a requested element is not already covered by a previously chosen set, a set $S \in \mathcal{S}$ containing it is chosen, paying a cost $c(S)$. The goal is to minimize the cost of the selected sets. Online submodular cover generalizes online set cover - the goal is to cover a general (monotone) submodular function with an increasing cover demand over time. In supermodular paging (submodular cover), the cover demand does not change over time, as the same page (a set $S \in \mathcal{S}$ in online set cover) may be requested (removed from the cover) multiple times. We show in the full version of the paper (see [14]) that supermodular paging is even more challenging than (online) submodular cover.

1.2 Motivation

Non-linear paging generalizes several fundamental caching problems, capturing many real world applications. Besides known applications of the classic paging models with linear feasibility functions [38, 15, 4, 5, 1], there are many scenarios in which the interaction between

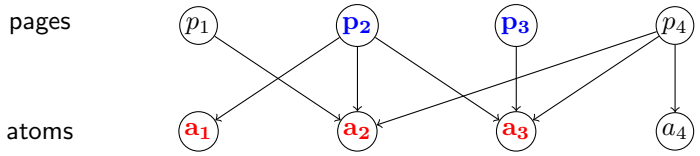

Figure 1 An illustration of a shared memory systems with four pages and four shared memory units ("atoms"). The cache is of size $k = 3$ and contains pages p_2, p_3 (in blue) whose shared memory is of size 3 - the three red atoms. Observe that either fetching p_1 or evicting it will not change the number of atoms stored in cache, however, fetching p_4 requires evicting p_2.

pages is non-linear, requiring the more general non-linear paging model. As already indicated, supermodular paging roughly generalizes online set cover and therefore has both theoretical and practical importance [20, 2]. We describe below several interesting applications of non-linear paging.

A major motivation for the study of non-linear paging is caching in shared memory systems (e.g., [28, 7, 43, 40]). Each process in a multi-process memory system is associated with a *virtual memory* [11], providing the illusion that it has a much larger memory. In some shared memory systems, caching policies are defined over entire processes, that is, the entire virtual address space of a process can be taken to the cache. Since the virtual memory of two processes typically overlaps in physical memory, the increase in (physical) memory in cache is larger if the cache is empty, as the entire memory of a process is loaded to the cache. In contrast, if the cache is nearly full, and a process is loaded to the cache, the increase in total size is smaller, as most of the virtual memory addresses are already in cache. In a similar vein, when caching hot data at the network edge, to avoid serving requests from a remote cloud, space efficiency of similar files is achieved through *deduplication* (e.g. [27]), which is very similar to the way overlaps are handled in virtual memory.

Thus, caching in shared memory systems is a special case of submodular paging. More formally, consider a cache that can store up to k *atoms* from a larger set $A = \{a_1, \ldots, a_n\}$ spanning the physical memory. Each page (process) p contains a subset of atoms $a(p) \subseteq A$ corresponding to the physical memory to which process p is mapped. The feasibility function f ensures that a collection S of pages contains jointly at most k atoms. Thus, $f(S) = \left|\bigcup_{p \in S} a(p)\right|$, and it is a submodular function. See Figure 1 for an illustration.

Paging with a supermodular feasibility function arises in common settings where the storage demand grows rapidly as a function of the number of "pages" stored in cache. In these scenarios, there is a large set of n *entities* (corresponding to vertices), and among subsets of entities certain interactions exist (represented by hyperedges). This data structure, known as a hypergraph, is ubiquitous in applications such as recommendation systems [19, 41] (where vertices represent individuals and hyperedges represent communities), image retrieval [29](vertices represent images and hyperedges represent correlations), and bioinformatics [32] (vertices represent substances and hyperedges stand for biochemical interactions). Other applications arise in machine learning [42, 18, 49] and databases [6].

In various practical settings, the hypergraph is very large (e.g., [36, 31, 22]). Therefore, a natural approach is to store frequently accessed vertices in a cache. However, caching is effective only if all interactions (i.e., hyperedges) among subsets of vertices in cache are also stored therein. As a set of x vertices may have up to 2^x induced hyperedges, the storage demand for hyperedges tends to be significantly larger compared to the number of vertices. Caching hypergraphs in the non-linear paging framework can be formally defined as follows:

consider a set $\mathcal{P}$ of vertices and define a feasibility function f over $\mathcal{P}$, such that for a subset of vertices $S \subseteq \mathcal{P}$, the storage demand $f(S)$ is the number of induced hyperedges in S plus the cardinality of S. Function f is a supermodular function.

For example, a real-world problem related to supermodular paging is caching in device-to-device (D2D) communication networks, with social ties among users and common interests that are used as key factors in determining the caching policy and are modeled via a hypergraph [3]. Here, the number of hyperedges (describing roughly interferences among users of the network, content, transmission rate, etc.) grows in a supermodular manner with respect to the number of users placed in cache. In addition, our reduction from online set cover (similarly, online submodular cover) to supermodular paging implies that applications of online set cover are also applications of supermodular paging.

1.3 Our Results and Techniques

We now present our results and elaborate on the techniques used. We start with general non-linear paging and then proceed to the special case of supermodular paging.

1.3.1 General Non-Linear Paging

In classic paging models, competitive ratios are typically given as a function of k, the cache size. However, non-linear paging is more difficult. Even for non-linear paging instances with $k = 0$ the competitive ratio can be very high. For example, consider a (classic) paging instance I' with cache threshold k'; define a non-linear paging instance I with a (non-linear) feasibility function f for which $f(S) = 0$ if S is feasible for I' and $f(S) = 1$ otherwise. In addition, we set $k = 0$ as the cache threshold of I. Clearly, a solution for I implies a solution for I'. Hence, by the hardness of paging [38, 15] the best competitive ratio of non-linear paging is arbitrarily large as a function of k (we give the remaining details in [14]). Thus, instead of k, we look for a parameter that better captures the competitiveness of general non-linear paging. This parameter turns out to be the maximum cardinality of a *minimally infeasible* set, i.e., an infeasible set where every proper subset of it is feasible[1]. Formally,

▶ **Definition 1.** *A set $S \subseteq \mathcal{P}$ is called* feasible *if $f(S) \leq k$ and is* infeasible *otherwise. Additionally, S is* minimally infeasible *if S is infeasible and every $S' \subset S$ is feasible, and let $\mathcal{M} = \{S \subseteq \mathcal{P} \mid f(S) > k \text{ and } f(S') \leq k \ \forall S' \subset S\}$ be all minimally infeasible sets. Finally, let the* width *of f be*

$$\ell(f) = \max_{S \in \mathcal{M}} (|S| - 1) .$$

We simply let $\ell = \ell(f)$ when it is clear from context. Clearly, for paging (or weighted paging) the width ℓ equals k. Hence, the width accurately captures the optimal performance of paging algorithms: there is a tight ℓ-competitive deterministic algorithm [38] and a tight $\Theta(\log(\ell))$-competitive randomized algorithm [15]. However, the width behaves quite differently in other scenarios. For example, in the setting of submodular paging described in Section 1.2, the instance can have a fixed width $\ell = O(1)$, but the number of pages in cache can be unbounded (e.g., all pages use the same atom). Interestingly, we show that the width gives a tight competitive ratio also for general non-linear paging via a new LP for the problem (see Sections 2 and 2.1).

[1] Technically, we subtract one so that the definition coincides with the parameter k in paging.

▶ **Theorem 2.** *There is a deterministic ℓ-competitive algorithm for* non-linear paging. *Moreover, every deterministic algorithm for* non-linear paging *is at least ℓ-competitive.*

The algorithm achieving Theorem 2 is based on a new LP relaxation, designed for the general setting of non-linear paging. We now explain why previous techniques for classic paging are not sufficient for obtaining a competitive online algorithm for this setting.

Previous work on generalized paging [5, 1] and on other online covering problems [20, 10] use *knapsack cover* constraints. This powerful technique, originated by Wolsey [44], is very useful for relaxing submodular cover constraints using linear inequalities. Indeed, paging problems are often studied from the viewpoint of a *covering* problem (e.g., [4]), where the complement of the cache (i.e., pages outside the cache) needs to be covered. In classic paging, at any point of time at least $n - k$ pages are not in the cache.

Consider the *covering* function $g : 2^{\mathcal{P}} \to \mathbb{N}$ of a feasibility function f defined to be the (non-linear) size requirement *outside* of the cache: $g(S) = f(\mathcal{P}) - f(\mathcal{P} \setminus S)$ for every $S \subseteq \mathcal{P}$. Observe that for classic paging it holds that $g(S) = n - (n - |S|) = |S|$ and the feasibility constraint translates to $g(S) \geq n - k$ at all times. If f is submodular (i.e., submodular paging) then g is supermodular, and vice versa. Knapsack cover constraints yield a relaxation of the covering problem when the cover function g is submodular [20], as is the case in classic paging problems.

However, for submodular paging, the covering function g is supermodular and knapsack cover constraints do not even provide a relaxation of the problem. For example, let $g(S) = 1$ for $S = \mathcal{P}$ and $g(S) = 0$ otherwise. Then, the knapsack constraints are not satisfied by the unique solution that covers a demand of $k = 1$ (the entire set $\mathcal{P}$). Specifically, $\mathcal{P}$ does not satisfy the knapsack constraint for $S = \emptyset$, i.e., $\sum_{p \in \mathcal{P}} x_p \cdot g_\emptyset(\{p\}) = 0$, but $g_\emptyset(\mathcal{P}) = 1$.

To circumvent the limits of knapsack cover constraints for submodular paging, we formulate a new set of covering constraints that are valid for any feasibility functions f and g. Specifically, the constraints require removing at least one page from every infeasible set. Then, using the online primal-dual approach applied to this set of constraints, we obtain a tight ℓ-competitive deterministic algorithm for non-linear paging. Specifically, upon arrival of a page that induces an infeasible set of pages in cache, our algorithm identifies a minimally infeasible set of pages and continuously increases their corresponding dual variable in the LP, evicting *tight* pages.

Interestingly, as a special case, Theorem 2 gives a simple k-competitive deterministic algorithm for generalized paging; to the best of our knowledge, there are only $(k + 1)$-competitive deterministic algorithms [8, 48] for generalized paging. Thus, our bound is tight for this problem.

▶ **Corollary 3.** *There is a deterministic k-competitive algorithm for* generalized paging.

We emphasize that the lower bound of Theorem 2 can be obtained for *any* function f with a minimally infeasible set of cardinality ℓ (regardless of whether f is linear, submodular, supermodular, or any other function). This shows the robustness of the parameter ℓ as an indicator for the competitiveness of non-linear paging. Thus, it is natural to ask whether the parameter ℓ for non-linear paging is analogous to the parameter k for classic paging when allowing randomization. We answer this question in the negative by showing that in contrast to paging, that admits an $O(\log(\ell))$-competitive randomized algorithm [15], non-linear paging is substantially harder w.r.t. the parameter ℓ.

▶ **Theorem 4.** *Unless* NP $\subseteq$ BPP*, there is no polynomial-time randomized $o(\log^2(\ell))$-competitive algorithm for* non-linear paging.

A very intriguing open question is whether there exists a randomized polylog(ℓ)-competitive algorithm for general non-linear paging. Unfortunately, we show that the LP used for obtaining Theorem 2 has an integrality gap of ℓ, and thus would need to be strengthened to achieve this end (see Section 2.3). Hence, obtaining a polylog(ℓ) competitive factor would require a new set of techniques. As we have already discussed earlier, existing techniques for obtaining randomized online algorithms for paging problems and related variants focus on solving covering linear programs (LP) online, and they break down in the presence of general non-linear paging constraints.

To overcome the integrality gap of our LP, we formulate a stronger LP for non-linear paging (see Section 2.3). Obtaining even a fractional polylog(ℓ)-competitive algorithm for this LP seems a hard task. Thus, we consider another parameter which allows us to get a better competitive ratio. The parameter is the maximum number of pages that fit together in the cache. Formally, define

$$\mu = \max_{S \subseteq \mathcal{P} \text{ s.t. } f(S) \leq k} |S| \tag{1}$$

as the maximum cardinality of a feasible set in cache. Observe that $\mu = k$ for e.g., generalized paging, hence it is a natural parameter to also consider in our setting. We also remark that in many practical settings, such as classic paging, it holds that $n \gg \mu$ and obtaining a competitive ratio that depends on μ (rather than n) is much more desirable.

Clearly, $\mu \geq \ell$. The following example illustrates a scenario demonstrating a scenario where $\mu \gg \ell$. Consider a non-linear paging instance on a set P of pages which is partitioned into disjoint sets X, Y, where $|X| = n$, $|Y| = k+1$, and $n \gg k$. Define a feasibility function f such that for all subsets S of P, $f(S) = |Y \cap S|$. Define the cache threshold as k. Therefore, the only minimally infeasible set is Y; thus, $\ell = k$. On the other hand, the maximum cardinality of a set that fits into cache is the cardinality of all pages in X and any k pages from Y; thus, $\mu = n + k$. Since $n \gg k$ (i.e., n can be chosen to be arbitrarily large with respect to k) it follows that $\mu \gg \ell$.

We give the following fractional algorithm for solving the strengthened LP online.

▶ **Theorem 5.** *There is an $O(\log \mu)$-competitive algorithm for obtaining a fractional solution for the strengthened* LP*.*

It as an interesting open question if a randomized polylog(μ)-competitive algorithm for non-linear paging can be designed by rounding the fractional solution obtained in Theorem 5.

1.3.2 Supermodular Paging

We now discuss our results and techniques for supermodular paging. Our main result is a polylogarithmic randomized competitive algorithm for supermodular paging, i.e., submodular cover paging. As we show later on, our upper bound turns out to be quite close to the lower bound we prove.

▶ **Theorem 6.** *There is an $O\left(\log^2 \mu \cdot \log\left(\frac{c_{\max}}{c_{\min}} \cdot f(\mathcal{P})\right)\right)$-competitive randomized algorithm for* supermodular paging (submodular cover paging)*, where $c_{\max}, c_{\min}$ are the maximum and minimum costs of pages, respectively.*

In particular, for unweighted supermodular paging (where $c_p = 1 \; \forall p \in \mathcal{P}$), the above theorem implies an $O\left(\log^2(\mu) \cdot \log\left(f(\mathcal{P})\right)\right)$-competitive algorithm.

Our algorithm relies on a different LP relaxation than the one described in Section 1.3.1. As we aim to solve supermodular paging, which implies a submodular cover function g, we design an LP relaxation inspired by the submodular cover relaxation of Wolsey [44] (see

also [20]). We solve this LP in the case that the constraints arrive online to obtain a fractional $O(\log(\mu))$-competitive (deterministic) algorithm, while maintaining the property that the entries are either integral or multiples of $\frac{1}{k}$.

To obtain an integral solution, one is tempted to maintain online a set of feasible *cache states* respecting (even approximately) the marginal probabilities induced by the fractional solution, similarly to previous works on weighted paging and generalized paging [4, 5, 1]. However, techniques for online cache state maintenance of [4, 5, 1] seem to break down in the presence of submodular cover constraints. Instead, we augment the fractional solution by an additional polylogarithmic *boosting factor* and perform randomized rounding, with possible corrections to ensure feasibility. The probability of a page to be evicted at some point in time is shown to be proportional to the page's fractional increase normalized by the probability that the page is in cache.

We also give lower bounds for online supermodular paging. Surprisingly, even though online set cover seems starkly different from supermodular paging, in particular since the cover constraints change over time for online set cover, we can show that supermodular paging is in a sense harder to solve online.

▶ **Theorem 7.** *For any $\rho \geq 1$, if there is a ρ-competitive algorithm for* supermodular paging*, then there is a ρ-competitive algorithm for* online set cover *having the same running time up to polynomial factors.*

By Theorem 7 and the results of [2, 23], we give a lower bound on the competitiveness of supermodular paging, indicating the necessity of the factor $\log \mu \cdot \log(\mathcal{P})$.

▶ **Corollary 8.** *Unless* NP $\subseteq$ BPP*, there is no polynomial $o(\log \mu \cdot \log(f(\mathcal{P})))$-competitive algorithm for* supermodular paging*. Moreover, there is no deterministic $o\left(\frac{\log \mu \cdot \log(f(\mathcal{P}))}{\log\log \mu + \log\log(f(\mathcal{P}))}\right)$-competitive algorithm for* supermodular paging *of any running time.*

We remark that some of our results from Section 1.3.1 can be stated using the parameter μ as well. However, as we showed earlier, there are non-linear paging instances in which the parameter $\mu \gg \ell$ and it grows artificially apart from the true hardness of the instance – in terms of competitive analysis. In contrast, every minimally infeasible set of cardinality $\ell + 1$ can be used to obtain the lower bounds for classic paging [38, 15] (i.e., ℓ- deterministic and $O(\log \ell)$-randomized lower bounds). Thus, in an informal sense, an increase in ℓ always incurs an increase in the difficulty of the problem, unlike an increase in μ. For this reason, we believe that searching for competitive algorithms and lower bounds in terms of ℓ, rather than μ, may be of greater interest in the non-linear paging setting.

Finally, we also aim at obtaining an offline approximation algorithms for supermodular paging (i.e., where all requests are known in advance). A first attempt would be to reduce the problem to offline submodular cover, as follows. Define a covering function G on the domain containing all pairs (p, j), for every page p and the j-th time it is requested. Define the value of G on a subset of pairs S as $G(S) = \sum_{t \in T} g(S_t)$, where g is the covering function of the instance (see Section 1.3.1), and S_t is the set of all pages p, where (p, j) belongs to S and the time interval between requests j and $(j + 1)$ for p intersects t. Clearly, the total cover demand, even with a unit demand per-time slot, is a function of T; thus, applying an offline set cover algorithm in a black box manner gives only an $\Omega(\log(T))$-approximation [30]. As T may be very large, a different technique is in place.

Combined with the strong *round-or-separate* algorithm of [20], our techniques yield an approximation algorithm for supermodular paging independently of T (see Section 3). We defer the details to the full version of the paper.

▶ **Theorem 9.** *There is an offline* $O\left(\log\left(\frac{c_{\max}}{c_{\min}} \cdot f(\mathcal{P})\right)\right)$*-approximation algorithm for* supermodular paging.

Due to space constraints, some of the proofs (including our lower bounds) are given only in the full version of the paper [14].

2 Deterministic Algorithm for Non-Linear Paging

In this section, we present a deterministic algorithm for non-linear paging and show that it gives a tight competitive ratio for the problem, thus providing proofs for Theorem 2 and Corollary 3. Our algorithmic results are based on a new LP relaxation for the problem which we introduce here.

2.1 LP Relaxation for Non-Linear Paging

Consider a non-linear paging instance with a set of pages $\mathcal{P}$, a feasibility function f, cache threshold k, a set of time points T, and a page request $p_t \in \mathcal{P}$ for every time $t \in T$. To simplify notation, for a set S and an element p we use $S + p = S \cup \{p\}$ and $S - p = S \setminus \{p\}$. The LP is as follows.

The variables of the LP are $x_p(j)$ for every page $p \in \mathcal{P}$ and the j-th time that p is requested. Intuitively, the value of the variable $x_p(j)$ indicates to what extent page p is evicted between its j-th and $(j+1)$-th requests. The constraints state that for every infeasible set S containing the requested page p_t, for time point t, we must "break" this set - i.e., evicting at least one page from $S - p_t$. If this constraint is satisfied for every such infeasible set by an integral solution, then it induces a feasible set of pages in the cache at all times. See Figure 2 for a visualization of these constraints.

We use n_p to denote the number of requests for page p during the request sequence. Also, we use $r(p,t)$ to denote the number of requests for page p until time t. We assume that $x_p(0)$ is a variable always set to 0, for all $p \in \mathcal{P}$. To simplify notation, for every $t \in T$, let $\mathcal{S}(t) = \{S \subseteq \mathcal{P} \mid p_t \in S \text{ and } f(S) > k\}$ denote the infeasible sets containing p_t. For any $w \in \mathbb{N}$, let $[w] = \{1, 2, \ldots, w\}$. Our LP relaxation is as follows.

$$\begin{aligned} \min \quad & \sum_{p \in \mathcal{P}} \sum_{j \in [n_p]} x_p(j) \cdot c(p) \\ & \text{s.t.} \\ & \sum_{p \in S - p_t} x_p(r(p,t)) \geq 1, && \forall t \in T \ \forall S \in \mathcal{S}(t) \\ & x_p(j) \geq 0 && \forall p \in \mathcal{P} \ \forall j \in [n_p] \end{aligned} \tag{2}$$

We now define the dual LP. For page $p \in \mathcal{P}$ and $j \in [n_p]$ let $I(p,j) = \{t \in T \mid j = r(p,t)\}$ be the *interval* of all time points between the j-th request to p till the last time point before the $(j+1)$-st request for page p. The *dual* of (2) is the following.

$$\begin{aligned} \max \quad & \sum_{t \in T} \sum_{S \in \mathcal{S}(t)} y_t(S) \\ & \text{s.t.} \\ & \sum_{t \in I(p,j)} \sum_{S \in \mathcal{S}(t) \mid p \in S - p_t} y_t(S) \leq c(p), && \forall p \in \mathcal{P} \ \forall j \in [n_p] \end{aligned} \tag{3}$$

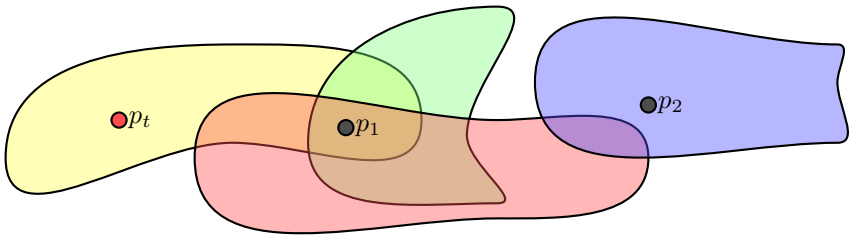

Figure 2 An illustration of the LP constraints. Each colored shape represents a minimally infeasible set of pages upon the arrival of a page p_t. Removing pages p_1 and p_2 ensures cache feasibility.

We use Primal-LP and Dual-LP to denote the values of the optimal solutions for the primal and dual programs, respectively, and by OPT the offline (integral) optimum. The next result follows from weak duality and since our LP is a relaxation of non-linear paging.

▶ **Lemma 10.** Dual-LP $\leq$ Primal-LP $\leq$ OPT.

Proof. The first inequality follows from weak duality, since (3) is the dual program of (2). For the second inequality, we show that (2) is a relaxation of non-linear paging. Consider an integral feasible solution M to our instance; define a solution x for (2) such that $x_p(j) = 1$ if and only if page p is removed from the cache during the interval $I(p, j)$ in M, for every $p \in \mathcal{P}$ and $j \in [n_p]$. Then, for every time $t \in T$ and $S \in \mathcal{S}(t)$, S cannot be fully contained in the cache at time t, since M is a feasible solution, and there is at least one $p \in S - p_t$ which is not in the cache at this time. Therefore, the primal constraint corresponding to t and S is satisfied. We conclude that x is feasible for (2); consequently, (2) is indeed a relaxation of non-linear paging, implying the second inequality. ◀

2.2 A Deterministic Algorithm for Non-Linear Paging

In this section, we give a primal-dual algorithm, based on the LP relaxation of non-linear paging presented in Section 2.1. For brevity, we denote the left handside of the dual constraint corresponding to page p and $j \in [n_p]$ as

$$Y_p(j) = \sum_{t \in I(p,j)} \sum_{S \in \mathcal{S}(t) \mid p \in S - p_t} y_t(S). \tag{4}$$

We call a page p *tight* at time t if $Y_p(j) = c(p)$.

The algorithm initializes an infeasible primal solution and a feasible dual solution as vectors of zeros $\bar{0}$. In every time step t, if the set of pages currently in cache, denoted by Cache_t, is infeasible, then our algorithm finds a subset of pages Q in the cache such that: first, Q is infeasible; second, Q contains the requested page p_t, i.e., $Q \in \mathcal{S}(t)$; third, Q has minimum cardinality amongst all such sets. we increase the variable $y_t(Q)$ continuously. Once one of the pages becomes tight, we remove it from cache. If the cache is feasible, the algorithm proceeds to time $t + 1$; otherwise, we repeat this process with a new set Q' from the current cache. The pseudocode is given in Algorithm 1.

We start by showing the feasibility of the algorithm.

Algorithm 1 Deterministic.

Initialize (infeasible) primal and (feasible) dual solutions $x \leftarrow \bar{0}$ and $y \leftarrow \bar{0}$.
for *time* $t \in T$ **do**
  Let $\mathsf{Cache}_t = \{p \in \mathcal{P} \mid r(p,t) \geq 1 \text{ and } x_p(r(p,t)) = 0\}$ be the pages currently in cache.
  while $f(\mathsf{Cache}_t) > k$ **do**
    Find $Q \subseteq \mathsf{Cache}_t$ such that $Q \in \mathcal{S}(t)$ of minimum cardinality.
    while $Y_p(r(p,t)) < c(p)\ \forall p \in Q - p_t$ **do**
      Increase $y_t(Q)$ continously.
    end
    for $p \in Q - p_t$ *such that* $Y_p(r(p,t)) = c(p)$ **do**
      Remove p from cache: $x_p(r(p,t)) \leftarrow 1$, $\mathsf{Cache}_t \leftarrow \mathsf{Cache}_t - p$.
    end
  end
end
Return the (primal) solution x and the (dual) solution y.

▶ **Lemma 11.** *Algorithm 1 returns primal and dual feasible solutions of* (2).

Proof. By Algorithm 1, the algorithm keeps evicting pages until reaching a feasible set of pages in the cache. By the monotonicity of the feasibility function f, we eventually reach a feasible set in cache: every page p is considered to be feasible alone in the cache, i.e., $f(\{p\}) \leq k$; thus, in the worst case, the content of the cache at the end of time step t is p_t. The above shows that the primal solution x is feasible, being a relaxation of the integer problem. In addition, the dual y is feasible since for all $p \in \mathcal{P}$ and $j \in [n_p]$ it holds that $Y_p(j) \leq c(p)$ by Algorithm 1. ◀

In the following we bound the competitive ratio of the algorithm. Let $c(x)$ be the cost of our (integral) solution x and let $v(y)$ be the value of the dual solution y obtained by Algorithm 1. Using the selection of minimal sets for Q in Algorithm 1 we have the following result. Recall the width parameter ℓ defined in Definition 1.

▶ **Lemma 12.** $c(x) \leq \ell \cdot v(y)$

Proof. By LP (2),

$$\begin{aligned} c(x) &= \sum_{p \in \mathcal{P}} \sum_{j \in [n_p]} x_p(j) \cdot c(p) \leq \sum_{p \in \mathcal{P}} \sum_{j \in [n_p]} x_p(j) \cdot Y_p(j) \\ &= \sum_{p \in \mathcal{P}} \sum_{j \in [n_p]} x_p(j) \cdot \left(\sum_{t \in I(p,j)} \sum_{S \in \mathcal{S}(t) \mid p \in S - p_t} y_t(S) \right). \end{aligned} \tag{5}$$

The inequality holds since we only evict tight pages. By changing summation order in (5),

$$c(x) \leq \sum_{t \in T} \sum_{S \in \mathcal{S}(t)} y_t(S) \cdot \sum_{p \in S - p_t} x_p(r(p,t)) \leq \sum_{t \in T} \sum_{S \in \mathcal{S}(t)} y_t(S) \cdot \ell = \ell \cdot v(y).$$

The second inequality holds since in Algorithm 1 we always choose a minimally infeasible set of cardinality at most $\ell + 1$; hence, $y_t(S) \neq 0$ only if $|S| \leq \ell + 1$. ◀

We are now ready to give the proof of our main theorem.

Proof of Theorem 2. By Lemma 11, the returned solution x is a feasible solution. Combined with Lemma 10 and Lemma 12, $c(x) \leq \ell \cdot v(y) \leq \ell \cdot \text{OPT}$. Along with the tight lower bound from the special case of paging [38], the statement of the theorem follows.

We remark that for any ℓ and function f such that $\ell(f) = \ell$, there is a minimally infeasible set S of cardinality ℓ; as a result, the lower bound for deterministic algorithms of paging [38] can be obtained from the set S. Namely, given a deterministic algorithm $\mathcal{A}$ for unweighted non-linear paging, construct a sequence of requests for pages in S that always requests the page missing in cache by algorithm $\mathcal{A}$; clearly, $\mathcal{A}$ pays a cost of one at each time step, since S is minimally infeasible. Conversely, the offline optimum would evict the page that is requested latest in the future, missing only once in ℓ requests, giving the lower bound of ℓ. ◀

2.3 Integrality Gap

In this section, we show the limitations of the LP. Specifically, we show that the integrality gap of the LP defined in (2) is at least ℓ; together with our algorithmic upper bound the integrality gap is exactly ℓ. We then discuss a stronger LP formulation based on (2).

▶ **Lemma 13.** *The integrality gap of* LP (2) *is* ℓ.

Proof. For some $n \geq 1$, consider an instance with n pages $\mathcal{P} = \{p_1, \ldots, p_n\}$ with uniform costs $c(p) = 1 \; \forall p \in \mathcal{P}$ and a uniform feasibility function f as in classic paging, i.e., $f(S) = |S|$ for all $S \subseteq \mathcal{P}$, and some cache capacity k. Thus, all minimally infeasible sets are of cardinality $k+1$ and it follows that $\ell(f) = k$. Define the sequence of requests $p_1, \ldots, p_n$. That is, each page is requested exactly once. Observe that each request results in a page fault. Hence, any integral solution, in particular the optimal integral solution, evicts at least $n - k$ pages. On the other hand, define a fractional solution such that $x_p(1) = \frac{1}{k}$ for all $p \in \mathcal{P}$, i.e., each page is evicted after its first request with fraction $\frac{1}{k}$ (note that each page is requested exactly once). Consider some infeasible set $S \subseteq \mathcal{P}$ such that $|S| > k$. It holds that

$$\sum_{p \in S} x_p(1) = |S| \cdot \frac{1}{k} \geq \frac{k+1}{k} \geq 1.$$

Thus, x satisfies the constraints of (2). The cost paid by the fractional solution x is $\frac{n}{k}$. Therefore, as n and k can be chosen arbitrarily, the integrality gap of the LP is at least

$$\lim_{n \to \infty} \frac{n-k}{\frac{n}{k}} = \lim_{n \to \infty} \left(k - \frac{k^2}{n} \right) = k = \ell(f).$$

Therefore, in general, the integrality gap cannot be smaller than ℓ. ◀

2.3.1 Discussion: a Stronger LP

The integrality gap example shows that LP (2) is not sufficient for obtaining a randomized polylog(ℓ)-competitive algorithm for general non-linear paging. Instead, we describe a stronger version of our LP (2), in which we require removing from each infeasible set S a set of pages S', so that the complement of S' in S, $S \setminus S'$, will be feasible in the cache (i.e., $f(S \setminus S') \leq k$). The strengthened LP and our fractional algorithm for solving the LP online are presented in [14], giving the proof of Theorem 5.

3 A Randomized Algorithm for Supermodular Paging

We provide here an $O(\log \mu)$-competitive algorithm for a fractional version of supermodular paging. Then, we design randomized online and offline algorithms for supermodular paging.

3.1 LP for Supermodular Paging

The starting point of our fractional algorithm is earlier work on submodular cover LP [44, 20, 10]. Consider a supermodular paging instance with a set of pages $\mathcal{P}$, a feasibility function f inducing the submodular cover function g (recall that $g(S) = f(\mathcal{P}) - f(\mathcal{P} \setminus S)$ for every $S \subseteq \mathcal{P}$), cache threshold k, a set of time points T, and a page request $p_t \in \mathcal{P}$ for every time $t \in T$. We use the following LP relaxation of supermodular paging. As in the LP introduced in Section 2.1, the variables of the LP are $x_p(j)$ for every page $p \in \mathcal{P}$ and the j-th time that p is requested. We will use the same notation as in the LP of (2). Recall that for some $S \subseteq P$ and $p \in \mathcal{P}$ we use $g_S(p) = g_S(\{p\}) = g(S+p) - g(S)$. Let $N = f(\mathcal{P}) - k$ be our *cover demand*; at any point of time, the (non-linear) total size outside of the cache must be at least N.

Our LP relaxation goes as follows; the constraints of the LP require that for every time t and subset of pages S (assumed to already be outside of the cache) we must evict from the cache (fractionally) a total size of at least $N - g(S) = f(\mathcal{P} \setminus S) - k$, since we cannot have more than a total size of k in cache. This constraint is very natural in the linear case (i.e., classic paging), but more involved in the submodular cover setting.

$$\begin{aligned} \min \sum_{p \in \mathcal{P}} \sum_{j \in [n_p]} & x_p(j) \cdot c(p) \\ \text{s.t.} & \\ \sum_{p \in \mathcal{P} - p_t} & x_p(r(p,t)) \cdot g_S(p) \geq N - g(S), && \forall t \in T \ \forall S \subseteq \mathcal{P} \\ & x_p(j) \geq 0 && \forall p \in \mathcal{P} \ \forall j \in [n_p] \end{aligned} \tag{6}$$

In the following we define the *dual* LP of (6).

$$\begin{aligned} \max \sum_{t \in T} \sum_{S \subseteq \mathcal{P}} & y_t(S) \cdot (N - g(S)) \\ \text{s.t.} & \\ \sum_{t \in I(p,j)} \sum_{S \subseteq \mathcal{P} \,\middle|\, p \in S - p_t} & y_t(S) \leq c(p), && \forall p \in \mathcal{P} \ \forall j \in [n_p] \\ & y_t(S) \geq 0 && \forall t \in T \ \forall S \subseteq \mathcal{P}. \end{aligned} \tag{7}$$

We use Primal-LP and Dual-LP to denote the values of the optimal solutions for the primal and dual programs, respectively, and by OPT the offline (integral) optimum.

Before we describe our fractional algorithm, we show that it is sufficient to satisfy only *minimal constraints* rather than all constraints of the LP. Formally, a primal constraint of (6) corresponding to $t \in T$ and $S \subseteq \mathcal{P}$ is called *minimal* for some solution x' if for all $p \in \mathcal{P}$ where $x'_p(r(p,t)) = 1$ it holds that $p \in S$.

▶ **Lemma 14.** *If x' satisfies all minimal constraints of (6), then x' is feasible for (6).*

Our fractional algorithm initializes the (infeasible) primal solution x and the (feasible) dual solution y both as vectors of zeros $\bar{0}$. When the requested page p_t at time t arrives, we do the following til x satisfies all LP constraints (6) up to time t.

At time t, the algorithm considers a *minimally violating set* of pages $Q \subseteq \mathcal{P}$. This set is a minimal set for our solution x violating the primal constraint in (6) corresponding to t and Q. We increase variable $y_t(\mathcal{P} \setminus Q)$ continuously and at the same time increase variables $x_p(r(p,t))$ for all pages in $(\mathcal{P} - p_t) \setminus Q$ except p_t. The increasing rate is a function of $Y_p(r(p,t))$, where

$$Y_p(j) = \sum_{t \in I(p,j)} \sum_{S \subseteq \mathcal{P} \mid p \in S - p_t} y_t(S) \tag{8}$$

is the left hand side of the corresponding dual constraint of p and $j = r(p,t)$ in (6) (analogously to (4)). This growth function has an exponential dependence on $Y_p(r(p,t))$ scaled by the cost of the page $c(p)$ and the number of pages possible in cache - the parameter μ. The growth of the variable $x_p(j)$ stops once it reaches $\frac{1}{2}$.

Algorithm 2 Fractional.

Initialize (infeasible) primal solutions $x, z \leftarrow \bar{0}$
Initialize (feasible) dual solution $y \leftarrow \bar{0}$.
for $t \in T$ **do**
  while x *is not feasible for* t **do**
    Find a minimal set $Q \subseteq \mathcal{P}$ for x such that $\sum_{p \in \mathcal{P} - p_t} x_p(r(p,t)) \cdot g_Q(p) < N - g(Q)$.
    while $\sum_{p \in \mathcal{P} - p_t} x_p(r(p,t)) \cdot g_Q(p) < N - g(Q)$ and Q is minimal for x **do**
      Increase $y_t(\mathcal{P} \setminus Q)$ continously.
      forall $p \in (\mathcal{P} - p_t) \setminus Q$ **do**
        increase $x_p(r(p,t))$ according to

$$x_p(r(p,t)) \leftarrow \frac{1}{\mu} \cdot \left(\exp\left(\frac{\ln(\mu+1)}{c(p)} \cdot Y_p(r(p,t)) \right) - 1 \right).$$

        if $x_p(r(p,t)) - \frac{z_p(r(p,t))}{2} \geq \frac{1}{4 \cdot N \cdot \mu}$ **then**
          $z_p(r(p,t)) \leftarrow 2 \cdot x_p(r(p,t))$.
        end
        if $x_p(r(p,t)) \geq \frac{1}{2}$ **then**
          $z_p(r(p,t)), x_p(r(p,t)) \leftarrow 1$.
        end
      end
    end
  end
end
Return the (primal) solution x and the (dual) solution y.

Once the primal constraint corresponding to t, Q is satisfied, or Q is no longer minimal for x (a page $p \in \mathcal{P} \setminus Q$ reaches 1), there are two cases. If x is feasible, the algorithm proceeds to the next time step. Otherwise, the algorithm repeats the above process with a new minimal violating set Q'. An additional property that will be useful in the analysis is that all non-zero entries of the obtained solution will be larger than $\Omega\left(\frac{1}{N \cdot \mu}\right)$ and there will be no fractional

entries larger than $\frac{1}{2}$. Thus, we update through the algorithm another primal solution z; we update an entry $z_p(j)$ to be the value of $2 \cdot x_p(j)$ whenever the difference $x_p(j) - z_p(j)$ becomes larger than $\Omega\left(\frac{1}{N \cdot \mu}\right)$. Moreover, we increase $z_p(j)$ immediately to 1 once $x_p(j)$ reaches $\frac{1}{2}$. The pseudocode of the algorithm is given in Algorithm 2.

3.2 Analysis of Algorithm 2

We now analyze the competitive ratio of the fractional algorithm. We start by claiming the feasibility of the solutions obtained throughout the execution of the algorithm.

▶ **Lemma 15.** *The primal solution x defined by Algorithm 2 is feasible for the* LP (6).

▶ **Lemma 16.** *The solution z returned by Algorithm 2 is feasible for* (6).

▶ **Lemma 17.** *Algorithm 2 returns a feasible dual solution to the* LP (6).

It remains to prove that the algorithm is $O(\log(\mu))$-competitive. To do so, we bound the increase in the primal x by a $O(\log \mu)$ factor of the dual increase, at any time.

▶ **Lemma 18.** *The cost of x is bounded by $O(\log(\mu))$ times the value of the dual y.*

Proof. Consider an infinitesimal increase in the value of the dual solution y. Specifically, assume that the algorithm chooses a minimal set Q in Algorithm 2 for time step t and that the dual variable $y_t(\mathcal{P} \setminus Q)$ increases infinitesimally by $dy_t(\mathcal{P} \setminus Q)$. Let dx and dy denote the infinitesimal change in the objective value of x and y, respectively. We bound the increase dx in x as a function of the increase dy.

$$\begin{aligned} dx &= \sum_{p \in \mathcal{P} - p_t} dx_p(r(p,t)) \cdot c(p) \\ &= \sum_{p \in (\mathcal{P} - p_t) \setminus Q} \frac{dx_p(r(p,t)) \cdot c(p) \cdot dy_t(\mathcal{P} \setminus Q)}{dy_t(\mathcal{P} \setminus Q)} \\ &= \sum_{p \in (\mathcal{P} - p_t) \setminus Q} \ln(\mu + 1) \cdot \left(x_p(r(p,t)) + \frac{1}{\mu} \right) \cdot dy_t(\mathcal{P} \setminus Q). \end{aligned} \tag{9}$$

The first equality holds since the increase in $y_t(\mathcal{P} \setminus Q)$ induces an increase only on the primal variables corresponding to pages in $(\mathcal{P} - p_t) \setminus Q$ by (6). The last equality follows from the growth rate of a variable $x_p(r(p,t))$, for some $p \in (\mathcal{P} - p_t) \setminus Q$, as a result of the growth in $y_t(\mathcal{P} \setminus Q)$. We separately analyze two of the expressions in (9). First, since y changes as a result of the increase in the variable $y_t(\mathcal{P} \setminus Q)$, by Algorithm 2 it implies that

$$\sum_{p \in (\mathcal{P} - p_t) \setminus Q} x_p(r(p,t)) \cdot g_Q(p) \leq \sum_{p \in \mathcal{P} - p_t} x_p(r(p,t)) \cdot g_Q(p) < N - g(Q). \tag{10}$$

For the second expression, let $S' \in (\mathcal{P} - p_t) \setminus Q$ such that (i) $g_Q(S') \geq N - g(Q)$ and (ii) S' is of minimum cardinality of all such sets. Clearly, there is such S' as $S'' = (\mathcal{P} - p_t) \setminus Q$ satisfies the first condition (p_t is feasible alone in cache). Since S' satisfies the cover constraints it holds that $f(\mathcal{P} \setminus (S' \cup Q) \leq k$; thus, by the definition of μ it holds that $|\mathcal{P} \setminus (S' \cup Q| \leq \mu$.

Therefore,

$$\begin{aligned}\sum_{p\in(\mathcal{P}-p_t)\setminus Q}\frac{1}{\mu} &= \frac{|(\mathcal{P}-p_t)\setminus Q|-1}{\mu}\\ &= \frac{|S'|+|\mathcal{P}\setminus(S'\cup Q)|-1}{\mu}\\ &\le \frac{N-g(Q)+\mu-1}{\mu}\\ &\le N-g(Q)+1 \le 2\cdot(N-g(Q))\,.\end{aligned} \tag{11}$$

The first inequality holds since $|S'| \le N - g(Q)$, as S' is the minimum cardinality set that covers the demand of $N - g(Q)$; thus, the marginal contribution of any page in S' to the cover is at least 1 implying the inequality. The first inequality also uses $|\mathcal{P} \setminus (S' \cup Q| \le \mu$ as explained above. For the last inequality, note that $N - g(Q) \ge 1$ since we assume that $y_t(\mathcal{P} \setminus Q)$ increases at this time, implying that the corresponding constraint of t and S is not trivially satisfied. Therefore, by (9), (10), and (11),

$$\begin{aligned}dx &\le \ln(\mu+1)\cdot dy_t(\mathcal{P}\setminus Q)\cdot\left(\sum_{p\in(\mathcal{P}-p_t)\setminus Q}x_p(r(p,t))+\sum_{p\in(\mathcal{P}-p_t)\setminus Q}\frac{1}{\mu}\right)\\ &= \ln(\mu+1)\cdot dy_t(\mathcal{P}\setminus Q)\cdot 3\cdot(N-g(Q))\\ &= O(\log(\mu))\cdot dy.\end{aligned} \tag{12}$$

Thus, by (12), every increase in y incurs an increase of at most a factor $O(\log(\mu))$ in x. Finally, note that if $x_p(j) \ge \frac{1}{2}$ then we immediately increase $x_p(j)$ to 1; this increase the total cost of x by a factor of 2 w.r.t. the value of y. ◀

To conclude, by Algorithm 2 and Algorithm 2 we can trivially bound the cost of z by a constant factor of the cost of x.

▶ **Observation 19.** *The cost of z is bounded by 4 times the cost of x.*

Finally, using the above we summarize the properties of z.

▶ **Lemma 20.** *Algorithm 2 returns a feasible primal solution z to (6) such that the following holds.*

1. *For all $p \in \mathcal{P}$ and $j \in [n_p]$ it holds that either $z_p(j) \in \{0, 1\}$ or that $z_p(j) \in \left[\frac{1}{4\cdot N\cdot\mu}, \frac{1}{2}\right]$.*
2. *The cost of z is bounded by $O(\log(\mu))$ times the cost of* OPT.

3.3 Randomized Rounding

In this section, we construct a randomized algorithm for supermodular paging based on an online rounding scheme of the solution z to the LP (6) obtained by Algorithm 2. Let

$$C = \max_{p,q\in\mathcal{P}}\frac{c(p)}{c(q)}$$

be the maximum ratio of costs taken over all pairs pages; we assume without the loss of generality that all costs are strictly positive. As a scaling factor for our algorithm, let $\alpha = \log\left(4\cdot C\cdot N^2\cdot\mu^2\right)$ and let z' be the solution obtained by augmenting z by a factor of α. That is, for all $p \in \mathcal{P}$ and $j \in [n_p]$ define $z'_p(j) = \min(1, \alpha \cdot z_p)$.

Our algorithm computes z' in an online fashion. At every time t, after updating z', the algorithm evicts each page $p \in \mathcal{P}$ from the cache with probability chosen carefully so that the total probability that p is missing from cache after this moment is exactly $z'_p(r(p,t))$. If the cache is not feasible after this randomized rounding procedure, we evict pages that increase the total cover until we reach feasibility. The pseudocode of the algorithm is given in Algorithm 3.

Algorithm 3 Randomized Rounding.

for $t \in T$ **do**
  If p_t is missing from cache: fetch p_t.
  Initialize $\Delta_{p_t} \leftarrow 0$.
  Update the solution z according to Algorithm 2.
  Define $z'_p(r(p,t)) \leftarrow \min(1, \alpha \cdot z_p)$ for all $p \in \mathcal{P}$.
  for $p \in \mathcal{P}$ **do**
    Evict p from cache with probability $\frac{z'_p(r(p,t)) - \Delta_p}{1 - \Delta_p}$.
    Update $\Delta_p = z'_p(r(p,t))$.
  end
  Let $\mathcal{F}$ be the pages outside of the cache (part of the cover).
  while $g(\mathcal{F}) < N$ **do**
    Evict a page $p \in \mathcal{P} \setminus \mathcal{F}$ such that $g_{\mathcal{F}}(p) > 0$.
  end
end
Return the integral solution.

Observe that we evict a page p at time t with probability $\frac{z'_p(r(p,t)) - \Delta_p}{\Delta_p}$, conditioned on the event that p is still in the cache. Thus, the probability that p belongs to the cache at the beginning of time t is Δ_p, and is $z'_p(r(p,t))$ after Algorithm 3. Thus, we have the following observation.

▶ **Observation 21.** *For all $p \in \mathcal{P}$ and $t \in T$ the probability that p is missing from the cache at time t is at least $z'_p(r(p,t))$.*

To analyze the performance of the algorithm, we use the following lemma. The proof follows from Lemma 2.5 in [20] combined with Observation 21.

▶ **Lemma 22.** *Let $\mathcal{F}$ be the set of pages outside cache at Algorithm 3 at time t. Then,*

$$\mathbb{E}[g(\mathcal{F})] \geq N - e^{-\alpha} \cdot N \geq N - \frac{1}{2 \cdot \mu^2 \cdot C \cdot N}.$$

Using Lemma 22, we bound the expected cost of the algorithm.

▶ **Lemma 23.** *Algorithm 3 returns a feasible integral solution for* supermodular paging *with expected cost $O(\log(\mu) \cdot \alpha) \cdot \mathrm{OPT} = O\left(\log^2(\mu) \cdot \log(C \cdot N)\right) \cdot \mathrm{OPT}$.*

The proof of Theorem 6 follows from Lemma 23. Moreover, the proof of Theorem 9 follows using the "round-or-separate" approach of [20].

References

1 Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An o (log k)-competitive algorithm for generalized caching. *ACM Transactions on Algorithms (TALG)*, 15(1):1–18, 2018.

2 Noga Alon, Baruch Awerbuch, and Yossi Azar. The online set cover problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 100–105, 2003.

3 Bo Bai, Li Wang, Zhu Han, Wei Chen, and Tommy Svensson. Caching based socially-aware d2d communications in wireless content delivery networks: A hypergraph framework. *IEEE Wireless Communications*, 23(4):74–81, 2016.

4 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)*, 59(4):1–24, 2012.

5 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM Journal on Computing*, 41(2):391–414, 2012.

6 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM (JACM)*, 30(3):479–513, 1983.

7 John K Bennett, John B Carter, and Willy Zwaenepoel. Adaptive software cache management for distributed shared memory architectures. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 125–134, 1990.

8 Pei Cao and Sandy Irani. {Cost-Aware}{WWW} proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems (USITS 97)*, 1997.

9 Tao Cheng, Kuochen Wang, Li-Chun Wang, and Chain-Wu Lee. An in-switch rule caching and replacement algorithm in software defined networks. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

10 Christian Coester, Roie Levin, Joseph Naor, and Ohad Talmon. Competitive algorithms for block-aware caching. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 161–172, 2022.

11 Peter J Denning. Virtual memory. *ACM Computing Surveys (CSUR)*, 2(3):153–189, 1970.

12 Mianxiong Dong, He Li, Kaoru Ota, and Jiang Xiao. Rule caching in sdn-enabled mobile access networks. *IEEE Network*, 29(4):40–45, 2015.

13 Ilan Doron-Arad, Guy Kortsarz, Joseph Naor, Baruch Schieber, and Hadas Shachnai. Approximations and hardness of packing partially ordered items. *In proc. WG*, 2024.

14 Ilan Doron-Arad and Joseph Naor. Non-linear paging, 2024. `arXiv:2404.13334`.

15 Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

16 Samoda Gamage and Ajith Pasqual. High performance parallel packet classification architecture with popular rule caching. In *2012 18th IEEE International Conference on Networks (ICON)*, pages 52–57. IEEE, 2012.

17 Peixuan Gao, Yang Xu, and H Jonathan Chao. Ovs-cab: Efficient rule-caching for open vswitch hardware offloading. *Computer Networks*, 188:107844, 2021.

18 Yue Gao, Meng Wang, Zheng-Jun Zha, Jialie Shen, Xuelong Li, and Xindong Wu. Visual-textual joint relevance learning for tag-based social image search. *IEEE Transactions on Image Processing*, 22(1):363–376, 2012.

19 Gourab Ghoshal, Vinko Zlatić, Guido Caldarelli, and Mark EJ Newman. Random hypergraphs and their applications. *Physical Review E*, 79(6):066118, 2009.

20 Anupam Gupta and Roie Levin. The online submodular cover problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1525–1537. SIAM, 2020.

21 Huawei Huang, Song Guo, Peng Li, Weifa Liang, and Albert Y Zomaya. Cost minimization for rule caching in software defined networking. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1007–1016, 2015.

22 Jin Huang, Rui Zhang, and Jeffrey Xu Yu. Scalable hypergraph learning and processing. In *2015 IEEE International Conference on Data Mining*, pages 775–780. IEEE, 2015.

23 Simon Korman. On the use of randomization in the online set cover problem. *Weizmann Institute of Science*, 2, 2004.

24 He Li, Song Guo, Chentao Wu, and Jie Li. Fdrc: Flow-driven rule caching optimization in software defined networking. In *2015 IEEE International Conference on Communications (ICC)*, pages 5777–5782. IEEE, 2015.

25 Rui Li, Yu Pang, Jin Zhao, and Xin Wang. A tale of two (flow) tables: Demystifying rule caching in openflow switches. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–10, 2019.

26 Rui Li, Bohan Zhao, Ruixin Chen, and Jin Zhao. Taming the wildcards: Towards dependency-free rule caching with freecache. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.

27 Shijing Li and Tian Lan. Hotdedup: Managing hot data storage at network edge through optimal distributed deduplication. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*, pages 247–256. IEEE, 2020.

28 David J Lilja. Cache coherence in large-scale shared-memory multiprocessors: Issues and comparisons. *ACM Computing Surveys (CSUR)*, 25(3):303–338, 1993.

29 Qingshan Liu, Yuchi Huang, and Dimitris N Metaxas. Hypergraph with sampling for image retrieval. *Pattern Recognition*, 44(10-11):2255–2262, 2011.

30 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

31 Keisuke Murakami and Takeaki Uno. Efficient algorithms for dualizing large-scale hypergraphs. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–13. SIAM, 2013.

32 Rob Patro and Carl Kingsford. Predicting protein interactions via parsimonious network history inference. *Bioinformatics*, 29(13):i237–i246, 2013.

33 Seyed Hamed Rastegar, Aliazam Abbasfar, and Vahid Shah-Mansouri. Rule caching in sdn-enabled base stations supporting massive iot devices with bursty traffic. *IEEE Internet of Things Journal*, 7(9):8917–8931, 2020.

34 Ori Rottenstreich, Ariel Kulik, Ananya Joshi, Jennifer Rexford, Gábor Rétvári, and Daniel S Menasché. Cooperative rule caching for sdn switches. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pages 1–7. IEEE, 2020.

35 Ori Rottenstreich and János Tapolcai. Optimal rule caching and lossy compression for longest prefix matching. *IEEE/ACM Transactions on Networking*, 25(2):864–878, 2016.

36 Nicolò Ruggeri, Martina Contisciani, Federico Battiston, and Caterina De Bacco. Community detection in large hypergraphs. *Science Advances*, 9(28):eadg9159, 2023.

37 Jang-Ping Sheu and Yen-Cheng Chuo. Wildcard rules caching and cache replacement algorithms in software-defined networking. *IEEE Transactions on Network and Service Management*, 13(1):19–29, 2016.

38 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

39 Michael Stonebraker, Anant Jhingran, Jeffrey Goh, and Spyros Potamianos. On rules, procedure, caching and views in data base systems. *ACM SIGMOD Record*, 19(2):281–290, 1990.

40 G Edward Suh, Larry Rudolph, and Srinivas Devadas. Dynamic partitioning of shared cache memory. *The Journal of Supercomputing*, 28(1):7–26, 2004.

41 Shulong Tan, Jiajun Bu, Chun Chen, Bin Xu, Can Wang, and Xiaofei He. Using rich social media information for music recommendation via hypergraph model. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 7(1):1–22, 2011.

42 Ze Tian, TaeHyun Hwang, and Rui Kuang. A hypergraph-based learning algorithm for classifying gene expression and arraycgh data with prior knowledge. *Bioinformatics*, 25(21):2831–2838, 2009.

43 Andrew W Wilson Jr. Hierarchical cache/bus architecture for shared memory multiprocessors. In *Proceedings of the 14th annual international symposium on Computer architecture*, pages 244–252, 1987.

44 Laurence A Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

45 Bo Yan, Yang Xu, and H Jonathan Chao. Adaptive wildcard rule cache management for software-defined networks. *IEEE/ACM Transactions on Networking*, 26(2):962–975, 2018.

46 Bo Yan, Yang Xu, Hongya Xing, Kang Xi, and H Jonathan Chao. Cab: A reactive wildcard rule caching system for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 163–168, 2014.

47 Jialun Yang, Tao Li, Jinli Yan, Junnan Li, Chenglong Li, and Baosheng Wang. Pipecache: High hit rate rule-caching scheme based on multi-stage cache tables. *Electronics*, 9(6):999, 2020.

48 Neal E Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

49 Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19, 2006.