Elastic Caching*

Anupam Gupta† Ravishankar Krishnaswamy‡ Amit Kumar§ Debmalya Panigrahi¶

Abstract

Motivated by applications in cloud computing, we study the classical online caching problem for a cache of variable size, where the algorithm pays a maintenance cost that monotonically increases with cache size. This captures not only the classical setting of a fixed cache size, which corresponds to a maintenance cost of 0 for a cache of size at most k and ∞ otherwise, but also other natural settings in the context of cloud computing such as a concave rental cost on cache size. We call this the *elastic caching* problem.

Our results are: (a) a randomized algorithm with a competitive ratio of $O(\log n)$ for maintenance cost that is an arbitrary function of cache size, (b) a deterministic algorithm with a competitive ratio of 2 for concave, or more generally submodular maintenance costs, (c) a deterministic n-competitive algorithm when the cost function is any monotone non-negative set function, and (d) a randomized constant-factor approximation algorithm for the offline version of the problem. Our algorithms are based on a configuration LP formulation of the problem, for which our main technical contribution is to maintain online a feasible fractional solution that can be converted to an integer solution using existing rounding techniques.

1 Introduction

In the classical caching problem, we are given a cache of fixed size, and the goal is to devise a page eviction strategy such that the number of page faults is minimized. This model is well-suited to traditional models of computation where the fast cache memory resides in the processor or in the RAM, and the size of the cache remains more-or-less fixed throughout the lifetime of a program. Managing a cache (i.e., deciding which pages to keep in the cache and which to evict) is a natural online problem. In fact, it was one of the first problems to be studied extensively in the online computation model, and led to the development of the widely popular *competitive analysis* framework and its extensions and refinements. In this competitive analysis framework, the basic caching problem is very well-understood: given a cache which can hold up to k pages, we know deterministic k-competitive and randomized H_k-competitive algorithms, and also that these results are best possible [16, 14]. Subsequently, there has been much work in understanding more general caching problems where pages can have non-uniform eviction costs (called weighted paging), and/or non-uniform sizes (called generalized caching). These problems are also by now well understood, with k-competitive deterministic algorithms and $\Theta(\log k)$-competitive randomized algorithms [18, 11, 5, 3] known. The latter results are based on recent developments using the online primal-dual framework.

In the past few years, however, the model of computation have undergone a paradigm shift. Vast amounts of computational and storage resources are available on the cloud, and one can rent varying amounts of these resources. Some well-known examples include the Amazon Web Service (AWS), Google Cloud and Microsoft Azure services. To accompany this new model of computation, the prevalent model of caching has also changed: essentially unlimited amounts of cache memory are available, and this cache memory has much lower latencies when compared to fetching data from typical data stores located on the cloud. Crucially, one can *rent varying amounts* of cache memory depending on one's requirements at any point of time. This service has come to be known as *elastic caching*, and is offered via products like Amazon's *Elasticache* [1] and Microsoft's *Azure Redis Cache* [2]. Of course, it would cost more to rent a larger cache, and so a user needs to trade off between the cache rental cost (which we call the "maintenance cost") and the latency cost when the desired page is not in the cache. For example, at the time of writing, Azure Cache offers individual cache units whose sizes range from 250 MB (at a cost of $0.055 per hour) to 53 GB (at a cost of $2.10 per hour). Indeed, finding this balance between minimizing maintenance costs and minimizing latency naturally leads to a new set of algorithmic challenges. In this paper, we initiate a theoretical study of such problems in a setting we call *elastic caching*.

Let us present our formal model. We have a slow memory of n pages, and some subset A_t of these pages are in the cache at time t. Requests to some of these n pages arrive over time. If a requested page is not present in the cache,

*This research was done under the auspices of the Indo-US Virtual Networked Joint Center on Algorithms under Uncertainty.

†Carnegie Mellon University, Pittsburgh PA 15217. Email: anupamg@cs.cmu.edu. Supported in part by NSF awards CCF-1536002, CCF-1540541, and CCF-1617790.

‡Microsoft Research India. Email: rakri@microsoft.com

§Department of Computer Science and Engg., IIT Delhi. Email: amitk@cse.iitd.ac.in

¶Duke University. Email: debmalya@cs.duke.edu. Supported in part by NSF grants CCF-1527084 and CCF-1535972, and an NSF CAREER Award CCF-1750140.

we fetch this page into the cache, and may also decide to evict one or more pages. Each page $i \in [n]$ has a weight w_i and a size s_i. There are two kinds of costs:

- *Eviction cost:* If we move from set A_{t-1} to A_t, we pay some cost for evicting the pages in $A_{t-1} \setminus A_t$. In this paper, we consider the case where the eviction cost is $d(A_{t-1}, A_t) := \sum_{i \in A_{t-1} \setminus A_t} w_i$. In other words, we pay w_i for evicting page i out of our cache.
- *Maintenance cost:* We are given a monotone function $c : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, and we pay a maintenance cost of $c(\sum_{i \in A_t} s_i)$ every time step. More generally, we also consider the case when the cost function is a non-negative set function, i.e., $C(A_t)$ is the cost of maintaining the set A_t for one unit of time. While in general this could be an arbitrary monotone function $C : 2^{[n]} \to \mathbb{R}_{\geq 0}$, we also consider the case when C has some structure, e.g., it is submodular.

The goal is to minimize the sum of the eviction and maintenance costs over all time. I.e.,

$$\min \sum_t \Big(d(A_{t-1}, A_t) + c(A_t) \Big). \tag{1.1}$$

Note that the regular fixed-cache-size paging problems are a special case of the above problem, where we set the maintenance cost function $c(s) = 0$ for $s \leq k$ and ∞ for $s > k$. Setting $w_i = s_i = 1$ gives the classical paging problem [16], having general weights but unit sizes gives the weighted paging problem [17], and having both being general gives the generalized caching problem [5].

1.1 Our Results.

Our main result is the following:

THEOREM 1.1. (MAIN THEOREM) *There is an $O(\log n)$-competitive randomized algorithm for elastic caching with general sizes and eviction weights when the maintenance cost only depends on the cache size.*

We also address the deterministic online complexity of the problem. In fact, we can even deal with arbitrary monotone maintenance cost functions.

THEOREM 1.2. (DETERMINISTIC ALGORITHM) *There is an n-competitive deterministic algorithm for the elastic caching problem with general eviction weights w_i, if the maintenance cost function $C : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is an arbitrary monotone function of the set of pages in the cache. In particular, this implies an n-competitive algorithm for the elastic caching with general weights and sizes.*

Since the paging problem with n pages and a fixed cache of size $k = n - 1$ has lower bounds of n and H_n for deterministic and randomized algorithms, these results are tight upto constant factors.

We then turn our attention to cost functions with some additional structure: Indeed, if the cost function is concave (i.e., the marginal cost per unit of additional cache decreases as we increase the cache size), or more generally, any monotone submodular function of the pages, we can give substantially improved results:

THEOREM 1.3. (SUBMODULAR COSTS) *There is a deterministic 2-competitive algorithm for the elastic caching problem with general eviction weights w_i, if the maintenance cost $C : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is a monotone submodular function of the set of pages in the cache.*

Finally, we consider the offline problem, and give a constant-factor approximation.

THEOREM 1.4. (OFFLINE PROBLEM) *There is a polytime $O(1)$-approximation algorithm for elastic caching with general sizes and eviction weights w_i where the maintenance cost only depends on the cache size.*

1.2 Our Techniques.

Our algorithms follow the online primal-dual paradigm: we write an LP relaxation for the problem, solve it fractionally, and then round the fractional solution in an online fashion. In the fixed-cache size setting, the LP relaxation is well-understood: for each page p and time t, we have a variable $x_{p,t}$ which indicates whether p is *outside* the cache or not. Now the eviction cost at time t is given by $\sum_p (x_{p,t} - x_{p,t-1})^+$; since there is a fixed cache size, there is no maintenance cost. This LP relaxation has a constant integrality gap when all pages are unit sized, and it forms the basis of very elegant $O(\log k)$-competitive randomized algorithms based on the primal-dual method for weighted paging [5]. For pages with varying sizes, one needs to augment this LP relaxation with the *knapsack-cover* inequalities [5], and in this case, both the online fractional algorithm and the online rounding are more involved [3].

Indeed, it is not difficult to extend the LP relaxation to the setting of elastic caching: for each time t and size s, we need to add indicator variables for whether a cache of size s is being used at time t—details are in Section 5. Moreover, for generalized caching, we can also strengthen the LP relaxation by adding the knapsack cover inequalities *for all possible sizes*. However, obtaining a primal-dual algorithm requires some need new ideas. For instance, since the size of the cache is not fixed, when do we stop raising the primal and dual variables when a new request arrives? In the fixed-size caching problems,

we can evict the pages other than the currently requested one using a multiplicative weights approach until the total fractional size of the non-evicted parts reaches the cache size k. It is not clear how to generalize this approach to our setting. A related question is how to raise the dual variables to get enough money to pay for the cache size we choose? (In fact, these questions are non-trivial even for the case of unit costs and weights.) Finally, the myriad Knapsack cover inequalities would only add to our woes!

A crucial insight to help address these issues (especially the last one) is to actually work with a stronger *configuration LP* relaxation for this problems. Indeed, the associated dual LP has much more structure that we leverage in our primal-dual updates. Using this much-larger LP, we show in Section 2 how to update the primal/dual variables, and how to get a stopping condition so that we can pay for both the eviction and maintenance costs to within an $O(\log n)$ factor of a feasible dual. At a high level, we use the tight constraints of the dual to decide a tentative cache size at each time step, and check if the fractional primal variables satisfy the knapsack cover inequalities w.r.t this tentative size. If not, we identify a violated constraint and monotonically increase the primal/dual variables corresponding to this set of pages. This will then change the tentative cache size, and the process would continue until we converge. Finally, we show how we can adapt (in a straight-forward manner) an elegant rounding scheme of [3] to convert our fractional solution into a randomized integer solution.

Using the configuration LP has another advantage: it allows us to cleanly generalize our results to settings when the maintenance cost of a cache is a *submodular* function of the pages in it. We design a simple primal-dual algorithm which actually recovers an integral solution which is also 2-competitive. The main idea here is that we can express the primal as a purely covering problem (submodularity means that we don't need the usual constraint which enforces that we pick at most one configuration at each time), and this observation greatly simplifies the algorithm.

The configuration LP is crucially needed when the maintenance cost is a monotone function of the set of pages in the cache. We show that it has integrality gap of at most n, and in fact, we can even give an on-line algorithm with this competitive ratio. We again rely on the dual of the configuration LP, but the dual growth process is more straightforward. Finally, to get our offline approximations, we show that one can solve the configuration LP up to a constant factor by relating it to a weaker LP relaxation which uses the Knapsack Cover inequalities. Then, given a fractional solution to the configuration LP, we use filtering based techniques to round it.

1.3 Related Work. The classical paging/caching problem with a fixed cache size has been widely studied since the mid-80s. When the cache holds k pages, Belady's offline algorithm (Farthest in the Future) performs an optimal number of evictions [7]. Sleator and Tarjan [16] and Fiat et al. [14] gave deterministic k-competitive and randomized $O(\ln k)$-competitive algorithms respectively, and showed that these were both optimal. In *weighted paging*, each page has a weight and the goal is to minimize the total weight of evicted pages. This is equivalent to the k-server problem on a star graph, so deterministic k-competitiveness follows from the algorithm of Chrobak et al. [12] for k-server on trees. Bansal et al. [4] settled the randomized complexity of this problem by showing a randomized $O(\ln k)$-competitive algorithm, illustrating the power of the primal-dual technique for these problems. In *generalized caching*, pages also have *sizes*, and the total size of the pages in the cache must not exceed the cache capacity k. (All sizes are integers.) The off-line problem is NP-hard and an $O(1)$-approximation was given by Bar-Noy et al. [6]. Cao and Irani [9] and Young [17] gave deterministic $(k+1)$-competitive algorithms. Bansal et al. [5] gave an $O(\log^2 k)$-competitive randomized algorithm using primal-dual techniques, which was subsequently improved to the tight $O(\log k)$-factor by Adamaszek et al. [3]. All these problems have a related (h,k)-counterpart, where the offline adversary is restricted by a cache of size $h < k$. Deterministic $k/(k-h+1)$-competitive and randomized $O(\ln k/(k-h+1)$-competitive algorithms are known for all these variants [18, 5, 3].

Recently, there has also been work on generalizing paging by considering settings where the set of pages in cache must be an independent set of a matroid [8], or the setting where there is a matroid where we need to maintain a basis but the maintenance cost function (which is linear over the ground elements) changes over time, and the objective is to minimize the sum of maintenance and eviction costs [15]. Another problem loosely related to ours is called generalized caching with rejections, where we can choose not to serve a page request by incurring a penalty [13].

1.4 Preliminaries and Notation. For a subset T of pages, we use $s(T)$ to denote the total size $\sum_{i \in T} s_i$ of pages in T. We also use S to denote the total size $\sum_i s_i$ of all the n pages. We use the small letter $c : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ to denote maintenance costs which depend only on the cache size, and the capital letter $C : 2^{[n]} \to R_{\geq 0}$ for general cost functions.

There are page requests at integer time steps, and we use the symbol i_t to denote the request at time t. For each

page i and $j \geq 1$, let $t_{i,j}$ be the time at which the j^{th} request for page i is made. For notational convenience, we divide the entire time horizon into *phases* for each page. For $j \geq 1$, the j^{th} phase for page i comprises of the time slots in the interval $[t_{i,j}, t_{i,j+1})$. Finally, for a page i and time t, we slightly abuse notation and use $j(i,t)$ to denote the current phase of page i at time t, i.e., $j(i,t) = \arg\max_{j'} t \geq t_{i,j'}$.

1.5 Roadmap. In Section 2, we describe the randomized $O(\log n)$-competitive algorithm for elastic caching for size-dependent maintenance cost. In Section 3, we give a 2-competitive algorithm for submodular maintenance costs. Then in Section 4 we consider deterministic algorithms for general monotone maintenance costs and finally, in Section 5, we consider the offline setting.

2 An $O(\log n)$-competitive Algorithm for Elastic Caching

In this section, we consider the setting where pages have arbitrary non-negative sizes and weights, and the cost for maintaining a set A of pages in cache is $c(s(A))$, and devise a randomized $O(\log n)$-competitive online algorithm using the online primal-dual framework.

2.1 The Primal and Dual Linear Programs. We begin by stating the primal and dual formulations.

$$\text{(LP1)} \qquad \min \sum_{i,j} w_i x_{i,j} + \sum_{\eta,t} c(s(\eta)) y_{\eta,t}$$

$$x_{i,j(i,t)} + \sum_{\eta: i \in \eta} y_{\eta,t} \geq 1 \qquad \forall i \neq i_t, \forall t \tag{2.2}$$

$$\sum_{\eta: i_t \in \eta} y_{\eta,t} \geq 1 \qquad \forall t \tag{2.3}$$

$$\sum_{\eta} y_{\eta,t} \leq 1 \qquad \forall t \tag{2.4}$$

$$x_{i,j} \geq 0 \qquad \forall i,j$$

$$y_{\eta,t} \geq 0 \qquad \forall \eta, t.$$

$$\text{(LPD1)} \qquad \max \sum_{i,t} \gamma_{i,t} - \sum_{t} \sigma_t$$

$$\sum_{t_{i,j}+1 \leq t < t_{i,j+1}} \gamma_{i,t} \leq w_i \qquad \forall i,j \tag{2.5}$$

$$\sum_{i \in \eta} \gamma_{i,t} - \sigma_t \leq c(s(\eta)) \qquad \forall \eta, t \tag{2.6}$$

$$\gamma_{i,t} \geq 0 \qquad \forall i,t$$

$$\sigma_t \geq 0 \qquad \forall t.$$

For the primal, we use a configuration LP (LP1), with variables $y_{\eta,t}$ for each configuration[1] η and time t. Recall that i_t is the requested page at time t, $t_{i,j}$ denotes the time at which the j^{th} request for page i is made, and $j(i,t)$ is the current phase for page i at time t. Here $x_{i,j}$ denotes the indicator variable whether the page i is evicted from the cache during phase j or not. For each time slot t, and each configuration $\eta \subseteq [n]$ of pages, there is an indicator variable $y_{\eta,t}$ for whether the set of pages in the cache at time t is η. For any configuration η, we use the expression $s(\eta)$ to denote $\sum_{i \in \eta} s_i$, the total size of configuration η.

For any request sequence, the total cost incurred is the total eviction cost plus the total maintenance cost, which is our LP objective function. Constraint (2.2) ensures that, at each time step, either a page is already evicted in the corresponding phase, or we must select a configuration which contains it; constraint (2.3) enforces that at each time t, the requested page be included in the cache; and constraint (2.4) enforces that we select at most one configuration at each time step. The following lemma is then easy to see.

LEMMA 2.1. *The optimal solution to (LP1) has objective value at most OPT.*

2.2 Online Algorithm for Fractional Solution. At any time step t, the relevant primal variables for our online algorithm are the $x_{i,j(i,t)}$ variables which intuitively correspond to the eviction probabilities of page i in its current phase $j(i,t)$. To make the overall intuition of our algorithm more transparent, the dual constraints (2.5) are called the *weight constraints*, and the dual constraints (2.6) the *size constraints*. When a new page request arrives at time t, we run Algorithm 2.1 to update the $x_{i,j(i,t)}, \gamma_{i,t}$ and σ_t variables monotonically at suitably chosen rates. We note that we do not directly compute the configuration y variables online, but instead identify a size $\mathfrak{s}_t^*$ of the cache at each time step such that the maintenance cost $c(\mathfrak{s}_t^*)$ can be charged to the increase in dual objective. Then using the x variables and the cache sizes computed, we infer a randomized eviction policy which ensures that the pages which remain in the cache have total size at most $\mathfrak{s}_t^*$.

The Intuition. At a high level, we raise the x variables for some subset of the pages at an exponential rate: since $x_{i,j}$ is the fraction of the page i that has been evicted in phase j, this increase gradually evicts these pages from the cache. Simultaneously, we raise the corresponding dual variable $\gamma_{i,t}$ linearly, such that the weight constraint for i becomes tight exactly when the primal variable reaches 1 (i.e., when page i has been fully evicted). But when do we stop raising the x variables? For the classical paging

[1] We shall use the term "configuration" to refer to a subset of pages.

problem, this is when the sum of $x_{i,j}$ values equals $n-k$, ensuring that $\leq k$ pages are in cache. But we are in the elastic caching setting, and the cache size is not fixed! Moreover, how do these x_{ij} variables interact with the choice of the configuration η at time t, and how do we account for the maintenance cost $c(\eta)$? We need some new ideas for this.

This is where the tight size constraints come into play. The intuition is that tight size constraints (2.6) indicate how large a cache we can maintain at various time steps. Indeed, if during the execution of the update algorithm, the size constraint becomes tight for some configuration η^* with $s(\eta^*) = s^*$, then the dual objective value collected until now at time t is $\sum_i \gamma_{i,t} - \sigma_t \geq \sum_{i \in \eta^*} \gamma_{i,t} - \sigma_t = c(s(\eta^*))$, enough to pay for a cache of size s^*. So as long we we ensure that the dual objective only increases monotonically during Algorithm 2.1's execution, we will be able to charge the maintenance cost of $c(s^*)$ to the increase in dual objective at time t. This is important to note because during the course of the execution of Algorithm 2.1, a size constraint may become tight at some point, and subsequently become slack because of the different rates in which our algorithm will update the $\gamma_{i,t}$ and σ_t variables.

Motivated by the preceding discussion, during the execution of Algorithm 2.1 at time step t, we maintain a *critical size*, denoted $\mathfrak{s}_t^*$. This critical size is equal to the maximum value of $s(\eta)$ over all configurations η such that the the size constraint corresponding to η became tight during the the execution of the update algorithm at time step t. Note that the value of $\mathfrak{s}_t^*$ increases monotonically over the course of the continuous update algorithm for time step t, and moreover, from the discussion in the preceding paragraph, we can charge the maintenance cost $c(\mathfrak{s}_t^*)$ to the increase in dual objective at time t.

It remains to figure out the subset of pages for which we increase the x and γ variables, and also when to stop the algorithm. To this end, we work with the goal that if the critical size at any instant of the update algorithm is $\mathfrak{s}_t^*$, the set of pages evicted from cache account for a size of at least $S - \mathfrak{s}_t^*$ (meaning that the remaining pages residing in cache can fit in a size of $\mathfrak{s}_t^*$). So one natural stopping point would be when $\sum_i s_i x_{i,j(i,t)}$ first exceeds $S - \mathfrak{s}_t^*$. However, the covering knapsack has a large integrality gap, and so we cannot translate this into a convex combination of feasible integer configurations which fit into a $\mathfrak{s}_t^*$-sized cache. This leads us to our final idea: we increase the x, γ and σ variables until the $x_{i,j(i,t)}$ variables satisfy the so-called *Knapsack Cover (KC) inequalities* for a covering requirement of $S - \mathfrak{s}_t^*$. While these inequalities also appear in previous algorithms for generalized caching [5], it is more subtle in our setting. Indeed, notice the cyclic nature of the process here: the currently violated KC-inequality will dictate how we update the primal and dual variables, which in turn will dictate how the critical size $\mathfrak{s}_t$ changes over time, which in turn dictates the subsequent KC-inequalities we need to satisfy.

To this end, we first define the Knapsack Cover inequalities, and then formally present the update algorithm. For clarity in presentation, we present the algorithm as a continuous-time procedure, and we can easily discretize it using standard techniques.

DEFINITION 1. *Given a set of pages T and a covering requirement of $R \leq s(T)$, a fractional solution $\{z_i\}_{\{i \in T\}}$ over the pages in T is said to* satisfy the knapsack cover (KC) inequalities *for the requirement of R if for all subsets $X \subseteq T$ such that $s(X) > s(T) - R$, we have*

$$\begin{aligned} & \sum_{i \in X} \min(s_i, (s(X) - (s(T) - R)))z_i \\ \geq \quad & (s(X) - (s(T) - R)). \end{aligned}$$

Equivalently, we want that for all subsets $Y \subseteq T$ such that $s(Y) < R$, we have $\sum_{i \in T \setminus Y} \min(s_i, (R - s(Y)))z_i \geq R - s(Y)$.

So to summarize, we update our $x_{i,j}, \gamma_{i,t}$, and σ_t variables such that the total eviction cost is comparable to the dual increase, and moreover, the x variables at the end of the continuous process satisfy the KC inequalities for a requirement of $S - \mathfrak{s}_t^*$. More accurately, we enforce that out of the pages in $[n] \setminus \{i_t\}$, we evict a total size of at least $(S - s_{i_t}) - (\mathfrak{s}_t^* - s_{i_t})$, so that even with including s_i in the cache, the total size would be at most $\mathfrak{s}_t^*$.

The Formal Algorithm. The algorithm for maintaining the fractional solution is shown below; we set the parameters $\alpha = \ln(1 + 1/\delta)$ and $\delta = 1/n$. The set A_t denotes the set of pages i for which we can potentially raise the $\gamma_{i,t}$ variables. In Step 6, we choose a suitable subset η of pages, and raise the dual variables corresponding to these in Steps 7–9. We will show that η happens to be a subset of A_t. In Step 11, we raise σ_t to ensure that the dual constraints (2.6) are satisfied. Observe that the actual increase in Steps 7–11 is infinitesimal, and then we go back to the **while** loop condition in Step 4. A more "discrete" way of stating this is that we increase these dual variables till (i) the set A_t changes, or (ii) the set η as defined in Step 6 changes.

We now prove that the Algorithm 2.1 maintains a feasible primal-dual pair of low cost.

CLAIM 1. *Consider the execution of Algorithm 2.1 at time t. The set η defined in Step 4 is a subset of A_t.*

Algorithm 2.1 ELASTICCACHE(t, i_t)

1: Initialize $A_t \leftarrow \{i : x_{i,j(i,t)} < 1\}$
2: Initialize $\gamma_{i_t,t} = c(s_i)$, $\gamma_{i,t} = 0$ for all $i \neq i_t$, $\sigma_t = 0$, and $\mathfrak{s}_t^* \leftarrow s_i$
3: Let $T \leftarrow [n] \setminus \{i_t\}$
4: **while** $\{x_{i,j(i,t)}\}_{\{i \in T\}}$ does not satisfy KC inequalities for T and $R := S - \mathfrak{s}_t^*$ **do**
5: Let $\mathcal{T}$ denote the set of all tight size constraints at this moment
6: Let $\eta \subseteq T$ correspond to a minimally violated KC inequality
7: **for all** $i \in \eta$ **do**
8: Increase $\gamma_{i,t}$ at rate $\min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t})$
9: Increase $x_{i,j(i,t)}$ at rate $\frac{\alpha}{w_i} \cdot \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}) \cdot (x_{i,j(i,t)} + \delta)$
10: Increase $\gamma_{i_t,t}$ at rate $\min(s_{i_t}, s(\eta) - \mathfrak{s}_t^* + s_{i_t})$
11: Increase σ_t at rate $\max_{\eta' \in \mathcal{T}} \sum_{i \in \eta' \cap (\eta \cup \{i_t\})} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t})$
12: Update $A_t \leftarrow \{i : x_{i,j(i,t)} < 1\}$
13: Update $\mathfrak{s}_t^* \leftarrow \max(\mathfrak{s}_t^*, \max_{\eta' \in \mathcal{T}} s(\eta'))$

Proof. Suppose the KC inequality for the requirement R and set T as defined in Steps 3–4 are not satisfied by the subset η of pages. In other words:

$$\begin{aligned} & \sum_{i \in \eta} \min(s_i, (s(\eta) - (s(T) - R)))x_{i,j(i,t)} \\ < \quad & (s(\eta) - (s(T) - R)). \end{aligned}$$

Further assume that there is a page $i^\star \in \eta$ for which $x_{i^\star,j(i^\star,t)}$ is 1. If

$$\min(s_{i^\star}, (s(\eta) - (s(T) - R))) = s(\eta) - (s(T) - R),$$

then the above inequality is no valid. So it must happen that $\min(s_{i^\star}, (s(\eta) - (s(T) - R)))$ is $s_{i^\star}$. Now, subtracting $s_{i^\star}$ from both sides in the inequality above, we see that the KC inequality for the subset $\eta \setminus \{i^\star\}$ is also violated, contradicting the minimality property of η.

LEMMA 2.2. *At all times during the execution of the algorithm, the dual variables remain feasible.*

Proof. We prove this by induction over time. Indeed, suppose the statement holds for all times up to $t-1$, and consider the algorithm execution after the arrival of page i_t at time t.

Firstly, we initialize the algorithm with $\gamma_{i_t,t} = c(s_i)$ and $\sigma_t = 0$. Since the variable $\gamma_{i_t,t}$ does not appear in any weight constraint, and $c(\cdot)$ is a monotonically increasing function, all size constraints are feasible after this initialization. Now, consider the weight constraints (2.5) for some $i \neq i_t$. Our update rules for $x_{i,j(i,t)}$ and $\gamma_{i,t}$ variables imply that $x_{i,j(i,t)} = \delta\left[\exp(\alpha\Gamma/w_i) - 1\right]$ where $\Gamma := \sum_{t_{i,j}+1 \leq t' < t_{i,j+1}} \gamma_{i,t'}$ is the total increase for the dual γ variables contributing to the dual weight constraint during its current phase. Hence, when Γ reaches w_i, we have $x_{i,j(i,t)}$ reaches 1 as well, and the page is no longer in the set A_t. Claim 1 shows that we will not raise $\gamma_{i,t}$ any more, and so this ensures feasibility of the weight constraints.

The size constraints (2.6) for time t are independent of previous times, so consider the current time step t. By the definition of the rate of increase of σ_t, we ensure that if size constraint η' is tight at any instant of this continuous update, the rate of increase of $\sum_{i \in \eta'} \gamma_{i,t}$ is at most the rate of increase of σ_t, and hence the size constraint is never violated.

Next, we show that the first term in the primal objective can be bounded against the dual objective, for which the following claim will serve useful.

CLAIM 2. *At each instant of the update algorithm, if $\mathfrak{s}_t^*$ denotes the critical size at this instant, and if η denotes the minimally violated KC constraint identified at this instant, then the rate of increase of dual objective at this instant is at least $s(\eta) - \mathfrak{s}_t^* + s_{i_t}$.*

Proof. Using the notation in Algorithm 2.1, we see that the rate of increase of dual objective is given by

$$\begin{aligned} & \sum_{i \in \eta \cup \{i_t\}} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}) \\ - \quad & \max_{\eta' \in \mathcal{T}} \sum_{i \in \eta' \cap (\eta \cup \{i_t\})} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}). \end{aligned}$$

Indeed the first term is the rate at which $\sum_i \gamma_{i,t}$ increases and the second term is the rate of increase of σ_t. Therefore it suffices to show that for any $\eta' \in \mathcal{T}$,

$$\begin{aligned} & \sum_{i \in \eta \cup \{i_t\}} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}) \\ & - \sum_{i \in \eta' \cap (\eta \cup \{i_t\})} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}) \\ \geq \quad & s(\eta) - \mathfrak{s}_t^* + s_{i_t}. \end{aligned}$$

This is equivalent to needing:

$$\sum_{i \in (\eta \cup \{i_t\}) \setminus \eta'} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}) \geq s(\eta) - \mathfrak{s}_t^* + s_{i_t},$$

which would in turn follow from:

$$\sum_{i \in (\eta \cup \{i_t\}) \setminus \eta'} s_i \geq s(\eta) - \mathfrak{s}_t^* + s_{i_t},$$

which would be implied by:

$$\mathfrak{s}_t^* \geq \sum_{i \in (\eta \cup \{i_t\}) \cap \eta'} s_i.$$

But this last inequality is true, since $\mathfrak{s}_t^*$ is the maximum size of a configuration corresponding to a tight constraint and η' is a tight configuration.

LEMMA 2.3. *At any instant of the update algorithm, the rate of increase of $\sum_i w_i x_{i,j}$ is upper bounded by 2α times the the rate of increase of the dual objective.*

Proof. Let us first calculate the rate of increase of $\sum_i w_i x_{i,j}$ at any instant of the update algorithm. By definition, this is equal to $\alpha \sum_{i \in \eta} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}) \cdot (x_{i,j(i,t)} + \delta)$ which is upper bounded by $\alpha(T_1 + T_2)$ where $T_1 = \sum_{i \in \eta} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t}) x_{i,j(i,t)}$ and $T_2 = \delta \sum_{i \in \eta} \min(s_i, s(\eta) - \mathfrak{s}_t^* + s_{i_t})$. We now bound T_1 and T_2 separately by the rate of growth of the dual objective.

Note that T_1 is precisely the left hand side of the minimally violated KC inequality. Therefore

$$T_1 \leq s(\eta) - (s(T) - R) = s(\eta) + s_{i_t} - \mathfrak{s}_t^*,$$

which is at most the rate of growth of dual objective by Claim 2. As for T_2, note that it is at most $\delta n(s(\eta) - \mathfrak{s}_t^* + s_{i_t})$, which is at most the dual growth rate by using $\delta = \frac{1}{n}$ and from Claim 2.

Another implication of Claim 2 is the following:

COROLLARY 2.1. *At any time step, the dual objective is non-decreasing over the execution of the algorithm.*

Proof. At any instant, the dual growth rate is at least $s(\eta) - \mathfrak{s}_t^* + s_{i_t}$ by Claim 2. But note that for η to be a violated KC inequality, it must be the case that $s(\eta) > \mathfrak{s}_t^* - s_{i_t}$, by plugging in the values of $T = [n] \setminus \{i_t\}$ and $R = S - \mathfrak{s}_t^*$ in definition 1.

LEMMA 2.4. *During the course of the execution of Algorithm 2.1, the quantity $c(\mathfrak{s}_t^*)$ is at most the increase in dual objective at time step t, i.e., $\sum_i \gamma_{i,t} - \sigma_t$.*

Proof. If $\mathfrak{s}_t^*$ is updated at any moment of the algorithm's execution, it means that there exists a tight size constraint η^* such that $\mathfrak{s}_t^* = s(\eta^*)$. In particular, at this moment, we have $\sum_i \gamma_{i,t} - \sigma_t \geq \sum_{i \in \eta^*} \gamma_{i,t} - \sigma_t = c(s(\eta^*)) = c(\mathfrak{s}_t^*)$. This observation, along with Corollary 2.1 which asserts that the dual is always increasing, completes the proof.

LEMMA 2.5. *When Algorithm 2.1 terminates for a time step t, it will maintain primal variables $x_{i,j}$ that satisfy the KC inequalities for a requirement of $R = S - \mathfrak{s}_t^*$.*

Proof. This follows just from the fact that if there is a violated constraint, then the algorithm updates the primal and dual variables until all KC inequalities are satisfied.

We now conclude this section by summarizing the output of our algorithm in the following lemma.

THEOREM 2.1. *Using Algorithm 2.1, we can compute in an online manner a sequence of fractional vectors $\mathbf{x}_t$ and a critical size $\mathfrak{s}_t^*$ for each time t such that:*

(a) the total fractional eviction cost $\sum_{i,t} w_i |x_{i,t} - x_{i,t-1}|$ is $O(\alpha)OPT$,
(b) at each time t, the $x_{i,t}$ variables satisfy the KC inequalities for the set of pages $[n] \setminus \{i_t\}$ and requirement $S - \mathfrak{s}_t^*$,
(c) $x_{i_t,t} = 0$ for each t and the requested page i_t, and
(d) the total maintenance cost $\sum_t c(\mathfrak{s}_t^*)$ is $O(\alpha)OPT$.

Proof. Recall that for every page i, the configuration LP defines variables $x_{i,j}$, where j corresponds to the phase defined by $[t_{i,j}, t_{i,j+1})$ for page i. We will like to extend this variable to $x_{i,t}$ for all (integer) time t. Again the interpretation should be that $x_{i,t}$ is 1 iff i is not in the cache at time t.

During our primal-dual algorithm, for each i and j, the variable $x_{i,j}$ remains 0 till time t reaches $t_{i,j}$. Then from time $t_{i,j} + 1$ onwards till time $t_{i,j+1}$ the algorithm raises $x_{i,j}$ and when the phase ends, it does not change $x_{i,j}$. Therefore, for a time $t \in [t_{i,j}, t_{j+1})$, we can simply define $x_{i,t}$ as the value of $x_{i,j}$ at time t during our primal-dual procedure.

So we can simply output the vector $\mathbf{x}_t$ defined by $x_{i,t}$ for all pages i along with the critical sizes $\mathfrak{s}_t^*$ computed when the Algorithm 2.1 terminates at each time step. Since $x_{i,t+1} \geq x_{i,t}$ unless a phase for page i ends at time t (in which case $x_{i,t}$ is reset to 0), the following observation is easy to see:

$$\sum_{i,t} w_i |x_{i,t+1} - x_{i,t}| \leq 2 \sum_{i,j} w_i x_{i,j}. \tag{2.7}$$

Therefore, property (a) above follows by combining this observation along with Lemma 2.3 and 2.2. Property (b)

follows by the termination condition of the while loop in Algorithm 2.1. Property (c) follows because we do not increase the $x_{i,j(i,t)}$ at the beginning of a new phase, i.e., at time $t = t_{i,j}$, and hence $x_{i_t,t} = 0$ for all requested pages (since they define the beginning of a new phase). Finally, property (d) follows from Lemma 2.4.

2.3 Online Rounding. Finally, we show how to round the fractional solution promised by Theorem 2.1. Indeed, this procedure is in fact a *direct adaptation* of a very elegant rounding scheme of Adamaszek et al. [3] for generalized caching. Our main observation is that the rounding algorithm of [3] essentially does not use the fact that the cache size is a fixed parameter over all time steps, and constructs the rounded solution *only using the knowledge of the fractional solution* $\mathbf{x}_t$ over time.

THEOREM 2.2. *Suppose we have a sequence of vectors* $\mathbf{x}_t$ *and sizes* $\mathfrak{s}_t$ *satisfying the properties of Theorem 2.1. Then, the online rounding algorithm of [3], which, when given this sequence of vectors* $\mathbf{x}_t$*, efficiently computes a sequence of distributions* $\mathcal{D}_t$ *over subsets of evicted pages such that:*

- *every set in the support of* $\mathcal{D}_t$ *evicts a total size of at least* $S - \mathfrak{s}_t$ *pages,*
- *the request page* i_t *is not evicted in any set in the support of* $\mathcal{D}_t$*, and*
- *the total earthmover distance* $EMD(\mathcal{D}_{t-1}, \mathcal{D}_t)$ *is at most* $O(1)\sum_i w_i|x_{i,t} - x_{i,t-1}|$.

Proof. In what follows, we give a proof sketch of Theorem 2.2. We describe the rounding of Adamaszek et al. at a high-level while deferring the complete details to [3]. Importantly, we will abstract out the features of their rounding which will be most relevant to our setting and use the other ingredients in [3] in a black-box manner.

Recall that by running the online primal-dual Algorithm 2.1, we can compute in an online manner, a sequence of fractional vectors $\mathbf{x}_t$ and a critical size $\mathfrak{s}^*_t$ for each time t such that:

(a) the total fractional eviction cost $\sum_{i,t} w_i|x_{i,t} - x_{i,t-1}|$ is $O(\alpha)OPT$,
(b) at each time t, the $x_{i,t}$ variables satisfy the KC inequalities for the set of pages $[n] \setminus \{i_t\}$ and requirement $S - \mathfrak{s}^*_t$,
(c) $x_{i_t,t} = 0$ for each t and the requested page i_t, and
(d) the total maintenance cost $\sum_t c(\mathfrak{s}^*_t)$ is $O(\alpha)OPT$.

We now sketch the rounding procedure, which will convert the $\mathbf{x}_t$ vector into a distribution of integer cache states, such that each valid cache state evicts at least $S - \mathfrak{s}_t$ size, and moreover, i_t is never evicted from any of them. Most importantly, successive distributions for time steps t and $t+1$ are "close" to each other.

We first introduce some notation before proceeding with the details. Let us partition the pages into different *size classes*, with size class $\mathcal{S}_\ell := \{i : 2^{\ell-1} < s_i \leq 2_\ell\}$.

Ideally, we want our cache states in our distribution to be close in some manner to the $\mathbf{x}_t$ vector, and it turns out that the correct characterization of close is the following.

DEFINITION 2. *Given a fractional vector* $\mathbf{z} \in [0,1]^n$*, let* L *denote the prefix of pages (sorted by non-increasing order of size) such that the total* z *value of pages in* L *first exceeds* 1 *(or* $L = [n]$ *if no such prefix exists). We say that a subset* $E \subseteq [n]$ *of pages* mirrors *the vector* $\mathbf{z}$ *if:*

- *for all pages* i*,* $z_i = 0$ *implies that* $i \notin E$*,*
- *for all pages* i*,* $z_i = 1$ *implies that* $i \in E$*,*
- *for each integer* ℓ*, we have* $\lfloor \sum_{i \in \mathcal{S}_\ell} z_i \rfloor \leq |E \cap \mathcal{S}_\ell|$*, and*
- *we have* $|E \cap L| \geq 1$.

Intuitively, the definition requires that within each size class, the total number of pages in E should be at least the total number of fractional pages in $\mathbf{z}$. Moreover, for some technical reasons which will crop up in the proof of Lemma 2.6, we also need that from the prefix of pages with fractional z value adding to 1, the set E must select at least one page. From our perspective, it is crucial that the definition of when a set E mirrors a solution $\mathbf{z}$ does not depend on the cache size, and only depends on the the structure of the solution $\mathbf{z}$ in different size classes and the prefix of 1 unit of mass.

Using this definition, the following structural lemma (a restatement of Lemma 3.3 in [3] to handle time-dependent cache sizes) shows that mirroring a fractional solution is sufficient to be a valid cache state at any time step.

LEMMA 2.6. *Let* $\mathbf{x}_t$ *be a fractional solution which satisfies the KC inequalities for requirement* $S - \mathfrak{s}^*_t$*, and suppose* $x_{i_t,t} = 0$*. Define* $z_{i,t} = \min(1, 16 \cdot x_{i,t})$*. Then if a set* E *of pages mirrors the vector* $\mathbf{z}_t$*, then* E *is a feasible integral solution for the covering knapsack problem with requirement* $S - \mathfrak{s}^*_t$*, and* E *does not contain* i_t*. Equivalently the complementary pages* $[n] \setminus E$ *describes a valid cache state containing* i_t*, and having pages of total size at most* $\mathfrak{s}^*_t$.

After this point, the algorithm in [3] *ignores the cache size altogether*, and simply seeks to maintain a distribution $\mathcal{D}_t$ over pages such that every set with positive probability in the distribution $\mathcal{D}_t$ *mirrors* the fractional solution $\mathbf{z}_t$ where $z_{i,t} = \min(1, 16 \cdot x_{i,t})$. Moreover, $\mathcal{D}_t$ can be computed by making bounded changes from $\mathcal{D}_{t-1}$. More formally, they show the following result (Section 3.2

of [3]) which is also their main contribution, and which we use in a black-box manner.

THEOREM 2.3. *The online rounding algorithm of [3], which, when given this sequence of vectors $\mathbf{z}_t$, efficiently computes a sequence of distributions $\mathcal{D}_t$ over subsets of evicted pages such that:*

- *every set in the support of $\mathcal{D}_t$ mirrors $\mathbf{z}_t$ pages, and*
- *the total earthmover distance $EMD(\mathcal{D}_{t-1}, \mathcal{D}_t)$ is at most $O(1)\sum_i w_i|z_{i,t} - z_{i,t-1}|$.*

Combining Lemma 2.6 with Theorem 2.3 completes the proof.

Combining this with Theorem 2.1 then proves Theorem 1.1.

3 Elastic Caching with Submodular Costs

We now consider the Submodular Elastic Caching problem, where the maintenance cost for a set $\eta \subseteq [n]$ of pages in the cache is given by a monotone submodular function $C(\eta)$. We now show that the primal-dual approach gives a deterministic 2-competitive algorithm in this case. The configuration LP is written in a similar manner as LP1, given below. As compared to LP1, the main simplification here is that we do not have any constraint corresponding to (2.4) here, i.e., we do not need to explicitly enforce that we only pick one configuration at any time instant. Indeed, if any solution picks multiple configurations, say, $\eta_1, \eta_2, \ldots, \eta_\ell$ to satisfy the constraint (3.8), then $\eta = \cup\eta_i$ will also satisfy all the constraints, with only lesser holding cost as $C(\eta) \leq \sum C(\eta_i)$ due to submodularity of $C(\cdot)$.

$$\text{(LP2)} \qquad \min \sum_{i,j} w_i x_{i,j} + \sum_{\eta,t} C(\eta) y_{\eta,t}$$

$$\text{(3.8)} \qquad x_{i,j(i,t)} + \sum_{\eta: i \in \eta} y_{\eta,t} \geq 1 \qquad \forall i \neq i_t, \forall t$$

$$\text{(3.9)} \qquad \sum_{\eta: i_t \in \eta} y_{\eta,t} \geq 1 \qquad \forall t$$

$$x_{i,j} \geq 0 \qquad \forall i, j$$

$$y_{\eta,t} \geq 0 \qquad \forall \eta, t.$$

$$\max \sum_{i,t} \gamma_{i,t}$$

$$\text{(3.10)} \qquad \sum_{t_{i,j}+1 \leq t < t_{i,j+1}} \gamma_{i,t} \leq w_i \qquad \forall i, j$$

$$\text{(3.11)} \qquad \sum_{i \in \eta} \gamma_{i,t} \leq C(\eta) \qquad \forall \eta, t$$

$$\gamma_{i,t} \geq 0 \qquad \forall i, t$$

The interpretation of variables $x_{i,j}$ remains as in LP1. The variable $y_{\eta,t}$ is the indicator variable which is 1 iff η is the set of pages in the cache at time t. The second term in the objective function captures the maintenance cost. The constraints (3.8) and (3.9) are similar to the constraints (2.2) and (2.3) in LP1.

We now describe the update algorithm when a new page is requested. Indeed, at any time t, say that page i is *active* if the dual constraint (3.10) is not tight for page i and the current phase at time t, i.e., $j = j(i,t)$. We also maintain a set F_t of frozen variables, which is initialized to $\emptyset$. In the update at time t, the algorithm begins with all dual variables $\gamma_{i,t} = 0$ and raises all the dual variables for *active, unfrozen* pages (i.e., those in $A_t \setminus F_t$) uniformly at rate 1, until one of the following event happens:

(a) There is a page i for which the dual constraint (3.10) becomes tight. This page is then removed from A_t, and gets evicted from the cache.
(b) One of the dual constraints (3.11) for a set η becomes tight. In this case, all pages in η gets frozen, i.e., we set $F_t := F_t \cup \eta$.

This process continues till we reach a stage when $\sum_{i,t} \gamma_{i,t}$ becomes larger than $C(A_t)$ – using submodularity, we will show later that such a stage will always occur. The update at time t then stops, and we keep all of A_t in the cache. The algorithm is described formally in Algorithm 3.1.

We now analyze this algorithm.

LEMMA 3.1. *The variables $\gamma_{i,t}$ form a feasible solution to the dual LP.*

Proof. Feasibility of (3.11) is easy to see – whenever this constraint becomes tight, we add η to the set F_t, and then we never $\gamma_{i,t}$ for any $i \in \eta$ thenceforth. Feasibility of (3.10) also follows from a similar argument. Indeed, if (3.10) becomes tight for any page i and phase j, the page i gets removed from the active set A_t, and is also evicted from cache. Hence, the $\gamma_{i,t}$ variables are no longer updated for all subsequent time steps in this phase for page i.

We now show that the algorithm is well-defined, i.e., the while loop terminates. The following structural claim, which is essentially an uncrossing argument, will be useful for this.

CLAIM 3. *If constraint 3.11 is tight for two sets η_1 and η_2, then it is tight for $\eta_1 \cup \eta_2$.*

Proof. Observe that

$$C(\eta_1 \cup \eta_2) + C(\eta_1 \cap \eta_2) \geq \sum_{i \in \eta_1 \cup \eta_2} \gamma_{i,t} + \sum_{i \in \eta_1 \cap \eta_2} \gamma_{i,t}$$

Algorithm 3.1 Online Primal-Dual Update at time t

Initialize A_t as set of all pages i for which (3.10) is not tight for the current phase $j(i,t)$.
Initialize $\gamma_{i,t} = 0$ for all i, and initialize set $F_t = \emptyset$
while $\sum_i \gamma_{i,t} < C(A_t)$ **do**
  for all $i \in A_t \setminus F_t$ **do**
    increase $\gamma_{i,t}$ at rate $\frac{\partial}{\partial t}\gamma_{i,t} = 1$
  Update A_t as set of all pages i for which (3.10) is not tight for the current phase $j(i,t)$.
  for all sets η for which constraint (3.11) is tight **do**
    Update $F_t := F_t \cup \eta$

$$= \sum_{i\in\eta_1} \gamma_{i,t} + \sum_{i\in\eta_2} \gamma_{i,t} = C(\eta_1) + C(\eta_2)$$
$$\geq C(\eta_1 \cup \eta_2) + C(\eta_1 \cap \eta_2).$$

The first inequality follows because of dual feasibility, the last inequality follows from submodularity, and the second equality follows because constraint 3.11 is tight for two sets η_1 and η_2. So all inequalities have to be satisfied at equality above, and in particular, the constraint (3.11) has to be tight for $\eta_1 \cup \eta_2$ and $\eta_1 \cap \eta_2$.

COROLLARY 3.1. *At any moment of the update algorithm, we have* $C(F_t) \leq \sum_{i\in F_t} \gamma_{i,t}$

Proof. By nature of when elements get added to F_t, we have that at any moment, F_t is a union of tight η_i's. Hence from Claim 3 we get that the dual constraint 3.11 is tight for F_t.

Using the above corollary we show that the algorithm is well-defined.

LEMMA 3.2. *For every time t, the* **while** *loop in Algorithm 3.1 terminates. Moreover, the requested page i_t is in A_t when the loop terminates.*

Proof. Fix some time t. We now show that if $\sum_i \gamma_{i,t} < C(A_t)$, then $A_t \setminus F_t \neq \emptyset$ (hence steps 11 and 12 are executed). Indeed, suppose not, i.e., $A_t \subseteq F_t$. Then Corollary 3.1 says that $\sum_i \gamma_{i,t} \geq \sum_{i\in F_t} \gamma_{i,t} \geq C(F_t) \geq C(A_t)$, which contradicts our assumption that $\sum_i \gamma_{i,t} < C(A_t)$. As for the current request i_t, observe that a new phase begins for this page, moreover, the variable $\gamma_{i_t,t}$ does not contribute to its constraint (3.10). Hence it will not get removed from A_t in this time step.

We can now bound the total cost of the on-line algorithm.

LEMMA 3.3. *The total eviction cost and the cache maintenance cost of the on-line algorithm is at most twice the optimal cost.*

Proof. We first bound the eviction cost. Any time we evict a page i in some phase j, it must be that $\sum_{t_{i,j}<t<t_{i,j+1}} \gamma_{i,t} = w_i$. So we can simply charge the eviction cost w_i of page i in phase j to the term $\sum_{t_{i,j}<t<t_{i,j+1}} \gamma_{i,t}$. Summing over all i and j, we get that the total eviction cost is at most $\sum_{i,t} \gamma_{i,t}$. As for the maintenance cost, we are guaranteed by the termination condition of the while loop, that at each time, the total holding cost $C(A_t)$ is at most $\sum_i \gamma_{i,t}$. Summing over all t, we get that the total holding cost is at most $\sum_{i,t} \gamma_{i,t}$. The proof then follows by weak duality and from the fact that γ is a feasible dual solution (Lemma 3.1).

4 Deterministic Algorithms for Elastic Caching with Monotone Maintenance Cost

In this section, we consider the Elastic Caching problem where the maintenance cost is an *arbitrary non-negative monotone set function* $C(\eta)$ of the configuration η of pages in the cache, i.e., $C(\eta) \geq 0$ for all η, and $C(\eta) \leq C(\eta')$ for $\eta \subseteq \eta'$. We give a simple deterministic n-competitive algorithm for this problem using the primal-dual method. Note that this also implies an n-competitive algorithm for the basic Elastic Caching problem studied in Section 2 when the maintenance cost is a function of the cache size.

One can write a configuration LP which is identical to (LP1), except that $c(s(\eta))$ is replaced by $C(\eta)$ in the objective function. Our algorithm will need to work with the dual LP, which is again very similar to the dual LP (LPD1). The dual LP is as follows:

$$\text{(LPD2)} \qquad \max \sum_{i,t} \gamma_{i,t} - \sum_t \sigma_t$$
$$\text{(4.12)} \qquad \sum_{t_{i,j}+1\leq t<t_{i,j+1}} \gamma_{i,t} \leq w_i \qquad \forall i,j$$
$$\text{(4.13)} \qquad \sum_{i\in\eta} \gamma_{i,t} - \sigma_t \leq C(\eta) \qquad \forall \eta, t$$
$$\gamma_{i,t} \geq 0 \qquad \forall i,t$$
$$\sigma_t \geq 0 \qquad \forall t.$$

The algorithm is described in Algorithm 4.1. At each time t, the algorithm maintains a set A_t of active pages

– these are the pages i for which the constraint (4.12) for $j = j(i,t)$ is not tight (i.e., satisfied with equality). Any page which becomes inactive gets evicted from the cache (if it was in this cache at this moment). The algorithm uniformly raises all the $\gamma_{i,t}$ variables for $i \in A_t$. We add a configuration η to a set $\mathcal{T}$ if the constraint (4.13) for η, t gets satisfied with equality at some point of the execution. To ensure feasibility of (4.13), we raise σ_t at a rate equal to the maximum number of active pages in a configuration $\eta \in \mathcal{T}$. The process continues as long as all the active pages are not contained in a single configuration in $\mathcal{T}$. All active pages when the algorithm terminates remain in cache at time t.

Termination of the algorithm follows from the fact that either the set $\mathcal{T}$ grows, or the set of active pages A_t shrinks over the course of execution. Also note that the page i_t will never get removed from A_t because $\gamma_{i_t,t}$ does not appear in any constraint of the form (4.12), and hence $i_t \in A_t$ when the algorithm terminates. Finally, we note that it is possible that constraint (4.13) for some η gets slack after becoming tight. However, once η enters $\mathcal{T}$, it continues to remain in it.

We now analyze the algorithm.

CLAIM 4. *The dual variables are feasible, and the rate of increase of dual objective function is always at least 1. Further,*

$$\sum_{i,t} \gamma_{i,t} - \sum_t \sigma_t \geq \frac{1}{n} \sum_{i,t} \gamma_{i,t}.$$

Proof. Feasibility of dual variables is easy to see – whenever (4.12) gets tight, we stop raising the corresponding $\gamma_{i,j(i,t)}$ variable, and whenever (4.13) gets tight, we raise σ_t so that it never gets violated.

The rate of increase of $\sum_i \gamma_{i,t}$ is $|A_t|$, while that for σ_t is $|A_t \cap \eta|$ for some $\eta \in \mathcal{T}$. By the termination condition of while loop, it must be the case that $|A_t \cap \eta| < |A_t|$. Therefore the rate of increase of $\sum_i \gamma_{i,t} - \sigma_t$ is at least 1. Since the rate of increase of $\sum_i \gamma_{i,t}$ is at most n, the last statement in the claim also follows.

We now bound the cost of the algorithm. A page i gets evicted during phase j if the corresponding constraint (4.12) becomes tight. Therefore, the total eviction cost can be bounded by

$$\sum_{i,j} \sum_{t_{i,j}+1 \leq t \leq t_{i,j+1}} \gamma_{i,t} \leq \sum_{i,t} \gamma_{i,t} \leq n \left(\sum_{i,t} \gamma_{i,t} - \sum_t \sigma_t \right),$$

where the last inequality is given by Claim 4. Thus, the total eviction cost is at most n times the cost of a feasible dual solution.

Now we account for the maintenance cost. Fix a time t. The cache contains the set A_t at the end of the update step in Algorithm 4.1. Note that there is a configuration $\eta \in \mathcal{T}$ containing A_t. By monotonicity, it follows that $C(A_t) \leq C(\eta)$. The fact that η was added to $\mathcal{T}$ implies that there was a moment during the update algorithm when the constraint (4.13) for η, t became tight, i.e., when

$$\sum_{i \in \eta} \gamma_{i,t} - \sigma_t = C(\eta).$$

Since the LHS is at most the total dual raised during time t (using Claim 4), we can pay for the maintenance cost by the total dual objective function. This completes the proof of Theorem 1.2.

5 Offline Algorithm for Elastic Caching

In this section we describe a constant factor polynomial-time approximation algorithm for elastic caching when the maintenance cost is a monotone function of the cache size. Indeed, one approach would be to solve the configuration LP (LP1) in polynomial time, and round it. However, notice that finding a separation oracle for (2.6) in the dual LP is equivalent to solving several knapsack problems, one for each possible size of the cache. While this can be done in time polynomial in the total size of the n pages, we would like to design strongly polynomial-time algorithms. Moreover, since the dual constraint involves difference of dual variables, it is not clear if an approximation algorithm (even an FPTAS) for the knapsack problem is useful as a separation oracle.

As a result, we present a different formulation in Section 5.1, based on the Knapsack Cover inequalities. We first show a structural theorem that proves that the new formulation and the configuration LP are within a factor of 2 of each other. Moreover, the proof will be "constructive", the meaning of which will become clear soon. So we can use the structure theorem to approximately solve the configuration LP in polynomial time. Once we do this, we do some simple filtering steps to decide the appropriate cache size at each time t, and then simply deploy our online rounding algorithm!

5.1 A Different LP relaxation for Elastic Caching.

We now describe the new LP formulation for Elastic Caching, which we call *knapsack cover* LP, in Figure 5.1.

Recall here that S denotes the total size of all pages. For each page i and phase j corresponding to page i, the variable $\alpha_{i,j}$ is an indicator variable for whether page i is evicted during phase j—this is same as the variable $x_{i,j}$

Algorithm 4.1 Monotone ELASTICCACHE(t, i_t)

Initialize A_t to be the set of pages i for which (4.12) for $i, j(i,t)$ is not tight.
Let $\mathcal{T}$ denote the set of all η for which (4.13) for η, t is tight.
while there is no η in $\mathcal{T}$ containing A_t **do**
    for all $i \in A_t$ **do**
        Raise $\gamma_{i,t}$ at a uniform rate of 1.
    Increase σ_t at rate $\max_{\eta \in \mathcal{T}} |A_t \cap \eta|$.
    Update A_t as the set of pages for which (4.12) for $i, j(i,t)$ is not tight.
    If a constraint (4.13) for a configuration η, t gets tight, add η to $\mathcal{T}$.
The cache at time t contains all the pages in A_t.

$$(LP_{KC}) \qquad \min \sum_{i,j} w_i \alpha_{i,j} + \sum_{s,t} c(s) \beta_{s,t}$$

$$\sum_{s} \beta_{s,t} \leq 1 \qquad \forall t \tag{5.14}$$

$$\alpha_{i,j(i,t)} + \sum_{s} \gamma_{s,i,t} \geq 1 \qquad \forall i \neq i_t, t \tag{5.15}$$

$$\sum_{s} \gamma_{s,i_t,t} \geq 1 \qquad \forall t \tag{5.16}$$

$$\gamma_{s,i,t} \leq \beta_{s,t} \qquad \forall s, i, t \tag{5.17}$$

$$\sum_{i \notin A} \min(S - s(A) - s, s_i)(\beta_{s,t} - \gamma_{s,i,t}) \geq (S - s(A) - s)\beta_{s,t} \qquad \forall s, t, A : S - s(A) - s > 0 \tag{5.18}$$

$$\alpha_{i,t}, \beta_{s,t}, \gamma_{s,i,t} \geq 0 \qquad \forall i, t, s$$

Figure 5.1: The knapsack cover LP

in the configuration LP. The indicator variable $\beta_{s,t}$ is 1 if the cache has size s at time t. The objective function is self-explanatory. Constraint (5.14) states that at most (in fact, exactly) one of variables $\beta_{s,t}$ will be 1 for any time t. We also have indicator variables $\gamma_{s,i,t}$ for page i, time t, and size s, which is set to 1 if the page i is in the cache at time t, and the cache at time t has size s. The constraint (5.15) states that if a page i does not get evicted during its phase j, then it must be in the cache at all times during this phase. If the page i happens to be same as i_t, then it must be in the cache – this is specified by (5.16). Constraint (5.17) says that if $\gamma_{s,i,t} = 1$, then $\beta_{s,t} = 1$. The final set of constraints (5.18) correspond to knapsack cover inequalities, and are defined for every subset A of pages, every size s and every time t. To check the feasibility of these constraints for a 0-1 solution, ignore A for a moment, and suppose $\beta_{s,t}$ is 1 for some size s and time t. (If $\beta_{s,t} = 0$, then the constraint holds trivially). Then the total size of pages which are not in the cache must be at least $S - s$, and so the variables $\beta_{s,t} - \gamma_{s,i,t}$ are a valid $0/1$ solution to the covering Knapsack problem of requirement $S - s$. Hence these variables satisfy all the Knapsack cover inequalities for the associated covering problem. We therefore get:

LEMMA 5.1. *For any instance $\mathcal{I}$ of Elastic Caching, the optimal value of $\mathcal{I}$ is at least the optimal value of the knapsack cover linear program LP_{KC} for this instance.*

Note that as stated, the size of this program is exponentially large in the input size (there are exponentially many KC-inequalities for *each size* s). In the next section, we first show an approximate equivalence between the Knapsack Cover LP and the configuration LP (LP1). Moreover, the proof will be constructive, in the sense that, either it recovers a feasible solution to the configuration LP, or it will identify a violated constraint, which can serve as the separation oracle. So even though we will not solve the above LP efficiently, we are still OK due to the so-called round-or-cut framework.

5.2 Approximate Equivalence of LP_{KC} and LP1. We now show that the two LPs are equivalent up to a factor of 2. First, we show the easy direction. Again, let $\mathcal{I}$ be an instance of Elastic Caching.

LEMMA 5.2. *For any feasible solution (x, y) to the configuration LP for $\mathcal{I}$ with objective value T, there is a feasible solution (α, β, γ) for (LP_{KC}) with objective value is at most T.*

Proof. Consider such a solution (x, y) to the configuration LP, and define:

$$\alpha_{i,j} = x_{i,j},\ \beta_{s,t} = \sum_{\eta: s(\eta)=s} y_{\eta,t},\ \gamma_{s,i,t} = \sum_{\eta: s(\eta)=s, i\in\eta} y_{\eta,t}.$$

It is easy to verify that the solution (α, β, γ) is feasible for the knapsack cover LP.

Now, we show the other direction, which is also the more useful direction for our rounding algorithm.

THEOREM 5.1. *For any candidate solution (α, β, γ) to (LP_{KC}) with objective value T, there is an algorithm which either*

- *finds a feasible solution (x, y) for the configuration LP with objective value is at most $2T$, or*
- *identifies a constraint in (LP_{KC}) which the solution (α, β, γ) violates.*

Moreover, the algorithms runs in time polynomial in the number of (s, t) tuples with $\beta_{s,t} > 0$.

Proof. Fix size s and time t such that $\beta_{s,t} > 0$, and consider the constraints (5.18) corresponding to s, t. These constraints can be interpreted as knapsack cover inequalities for a knapsack of size $S - s$, with variables $a_{i,s,t} := (1 - \frac{\gamma_{s,i,t}}{\beta_{s,t}})$ denoting whether page i of size s_i is in the knapsack. Then we can invoke Theorem 2.1 from [10] for the fractional solution $a_{i,s,t}$ to efficiently (in polynomial time) do the following: either identify some KC-inequality which is violated by the solution (α, β, γ), or compute variables $z_{\bar{\eta},s,t}$, where η corresponds to the configurations which are in the cache (and so its complement $\bar{\eta}$ corresponds to pages in the knapsack), such that:

$$2a_{i,s,t} \geq \sum_{\bar{\eta}: i\in\bar{\eta}} z_{\bar{\eta},s,t} \quad \forall i \tag{5.19}$$

$$\sum_{\eta} z_{\bar{\eta},s,t} = 1 \tag{5.20}$$

$$z_{\bar{\eta},s,t} \geq 0 \quad \forall \eta$$

$$z_{\bar{\eta},s,t} = 0 \quad \forall \eta : s(\bar{\eta}) < S - s. \tag{5.21}$$

In other words, we are expressing the variables $a_{i,s,t}$ as a near-convex combination of the configuration variables $z_{\bar{\eta},s,t}$. It is now natural to define the variable $y_{\eta,t}$ for the configuration LP as a weighted sum of all such $z_{\bar{\eta},s,t}$ variables, i.e.,

$$y_{\eta,t} = \sum_{s} z_{\bar{\eta},s,t} \cdot \beta_{s,t}$$

To define the $x_{i,j}$ variables, intuitively these should be close to the $\alpha_{i,j}$ variables. Since we lose a factor 2 in the convex combination above, we define

$$x_{i,j} = \min(2\alpha_{i,j}, 1).$$

Having defined these variables, it now remains to check that the desired conditions are satisfied. First, we bound the objective function of the solution (x, y) for the configuration LP. The first term in the objective is easy to bound:

$$\sum_{i,j} w_i x_{i,j} \leq 2 \sum_{i,j} w_i \alpha_{i,j}.$$

For the second term, we fix s, t and proceed as follows:

$$\begin{aligned}\sum_{\eta: s(\eta)\geq s} y_{\eta,t} &= \sum_{\eta: s(\eta)\geq s} \sum_{s'} z_{\bar{\eta},s',t} \cdot \beta_{s',t} \\ &= \sum_{s'} \beta_{s',t} \left(\sum_{\eta: s(\eta)\geq s} z_{\bar{\eta},s',t} \right)\end{aligned}$$

Consider the terms in the summation above for which $s' < s$. Constraint (5.21) imply that for such values of s', $z_{\bar{\eta},s',t}$ is 0 because $s(\bar{\eta}) \leq S - s < S - s'$. Therefore, we get

$$\begin{aligned}\sum_{\eta: s(\eta)\geq s} y_{\eta,t} &\leq \sum_{s'\geq s} \beta_{s',t} \left(\sum_{\eta: s(\eta)\geq s} z_{\bar{\eta},s',t} \right) \\ &\leq \sum_{s'\geq s} \beta_{s',t} \left(\sum_{\eta} z_{\bar{\eta},s',t} \right) = \sum_{s'\geq s} \beta_{s',t},\end{aligned}$$

where the last equality follows from (5.20).

By monotonicity of $c(\cdot)$, we can now claim that:

$$\sum_{\eta} c(s(\eta)) y_{\eta,t} \leq \sum_{s} c(s) \beta_{s,t}.$$

Next, we show feasibility of the solution (x, y) for the configuration LP. Fix t. We first check the constraints (2.4):

$$\begin{aligned}\sum_{\eta} y_{\eta,t} &= \sum_{\eta} \sum_{s} z_{\bar{\eta},s,t} \cdot \beta_{s,t} \\ &= \sum_{s} \beta_{s,t} \left(\sum_{\eta} z_{\bar{\eta},s,t} \right) \\ &\overset{(5.20)}{=} \sum_{s} \beta_{s,t} \overset{(5.14)}{\leq} 1\ .\end{aligned}$$

Now, we verify constraints (5.15) for each page i and time t lying between $t_{i,j}$ and $t_{i,j+1}$. First, we bound the second term in the LHS of this constraint:

$$\sum_{\eta: i\in\eta} y_{\eta,t} = \sum_{\eta: i\in\eta} \sum_{s} z_{\bar{\eta},s,t} \cdot \beta_{s,t}$$

$$
\begin{aligned}
&= \sum_s \beta_{s,t} \left(\sum_{\eta: i \in \eta} z_{\bar{\eta},s,t} \right) \\
&= \sum_s \beta_{s,t} \left(\sum_{\eta} z_{\bar{\eta},s,t} - \sum_{\eta: i \notin \eta} z_{\bar{\eta},s,t} \right) \\
&\overset{(5.21)}{=} \sum_s \beta_{s,t} \left(1 - \sum_{\eta: i \notin \eta} z_{\bar{\eta},s,t} \right) \\
&\overset{(5.19)}{\geq} \sum_s \beta_{s,t} \left(1 - 2 \left(1 - \frac{\gamma_{s,i,t}}{\beta_{s,t}} \right) \right) \\
&= \sum_s \beta_{s,t} \left(2 \cdot \frac{\gamma_{s,i,t}}{\beta_{s,t}} - 1 \right) \\
&\overset{(5.14)}{\geq} 2 \cdot \sum_s \gamma_{s,i,t} - 1.
\end{aligned}
$$

Now, we add the first term $x_{i,j}$. If $x_{i,j} = 1$, then the corresponding constraint (5.15) is already satisfied. Hence, we assume $x_{i,j} = 2\alpha_{i,j}$. Adding this term, we get:

$$
\begin{aligned}
x_{i,j} + \sum_{\eta: i \in \eta} y_{\eta,t} &\geq 2\alpha_{i,j} + 2 \cdot \sum_s \gamma_{s,i,t} - 1 \\
&= 2 \cdot \left(\alpha_{i,j} + \sum_s \gamma_{s,i,t} \right) - 1 \overset{(5.15)}{\geq} 1.
\end{aligned}
$$

Thus, we have shown that the variables (x, y) satisfy all the constraints in the configuration LP.

5.3 Approximation Algorithm for Elastic Caching. In this section, we describe a constant-factor approximation algorithm for Elastic Caching. As before, we shall use OPT to denote the optimal value of an instance $\mathcal{I}$ of this problem. As a first step, we scale terms so that the minimum cache cost is 1, and the maximum cache maintenance cost is $C_{\max}$. Next, we identify $L = O(\log C_{\max})$ cache sizes where the maintenance cost doubles. Then we formulate the Knapsack Cover LP LP_{KC} only over these L possible cache sizes. So far we have lost a factor of 2 in the approximation ratio due to this restriction on the cache sizes.

Next, we start with an arbitrary solution (α, β, γ) and invoke Theorem 5.1, which will either output a valid solution to the configuration LP LP1 within a constant factor of OPT, or identify a violated constraint, i.e., a separation oracle. The running time of this procedure will be polynomial in n, T and L. Then, by the ellipsoid method, after polynomially many applications of this procedure, we will identify a feasible solution to the configuration LP. We denote this solution by $(x^\star, y^\star)$.

5.3.1 Rounding the Configuration LP solution. Our rounding proceeds in two steps: first, we apply a *filtering* technique to ensure that at each time t, all configuration η s.t. $y_{\eta,t} > 0$ have a fixed size $\hat{\mathfrak{s}}_t$, and moreover, $\sum_t c(\hat{\mathfrak{s}}_t)$ is at most the LP objective. Then, we argue that the x-vector is therefore a feasible solution to the KC-inequalities for the Knapsack problem of evicting at least a total of $S - \hat{\mathfrak{s}}_t$ size of pages. Then we can simply feed these x vectors along with the sizes $\hat{\mathfrak{s}}_t$ to our online rounding procedure Theorem 2.2 to complete our overall rounding algorithm.

Filtering Step. We first modify the solution $(x^\star, y^\star)$ to a feasible solution $(\hat{x}, \hat{y})$ which has the nice property that $\hat{y}$ is supported on configurations of a uniform cache size for any fixed time t.

LEMMA 5.3. *There is a polynomial time algorithm to modify $(x^\star, y^\star)$ to a feasible solution $(\hat{x}, \hat{y})$ of cost at most twice that of $(x^\star, y^\star)$ such that it has the following properties:*

- *For every time t, there is a size $\hat{\mathfrak{s}}_t$ such that if $\hat{y}_{\eta,t} > 0$ for some configuration η, then $s(\eta) = \hat{\mathfrak{s}}_t$.*
- *The constraints (2.2), (2.3) and (2.4) are satisfied with equality.*

Proof. Fix a time t. Let $\hat{\mathfrak{s}}_t$ be the smallest size s such that $\sum_{\eta: s(\eta) \leq s} y^\star_{\eta,t} \geq 1/2$, and let γ_t denote this quantity. Define

$$
\hat{y}_{\eta,t} = \begin{cases} y^\star_{\eta,t}/\gamma_t & \text{if } s(\eta) \leq \hat{\mathfrak{s}}_t \\ 0 & \text{otherwise} \end{cases}
$$

We scale the x variables as well. Define

$$
\hat{x}_{i,j} = \min(1, 2x^\star_{i,j}).
$$

We now show feasibility of this new solution. It is easy to see that constraints (2.4) and (2.3) are satisfied with equality. To check feasibility of (2.2), fix a page $i \neq i_t$ and time t. Let $S_{i,t}$ denote the set of configurations η containing i such that $s(\eta) \leq \hat{\mathfrak{s}}_t$, and let $L_{i,t}$ denote those configurations containing i whose size is more than $\hat{\mathfrak{s}}_t$. Notice that $\sum_{\eta \in L_{i,t}} y^\star_{\eta,t} \leq 1 - \gamma_t$. Therefore, by feasibility of $(x^\star, y^\star)$,

$$
x^\star_{i,j(i,t)} + \sum_{\eta \in S_{i,t}} y^\star_{\eta,t} \geq \gamma_t.
$$

If $\hat{x}_{i,j(i,t)}$ becomes 1, there is nothing to prove. So assume $\hat{x}_{i,j(i,t)} = 2x^\star_{i,j(i,t)} \geq x^\star_{i,j(i,t)}/\gamma_t$. Further, if $\eta \in S_{i,t}$, then $\hat{y}_{\eta,t} = y^\star_{\eta,t}/\gamma_t$. Therefore,

$$
\hat{x}_{i,j(i,t)} + \sum_{\eta \in S_{i,t}} \hat{y}_{\eta,t} \geq 1.
$$

This proves the feasibility of $(\hat{x}, \hat{y})$. Now, if $\hat{y}_{\eta,t} > 0$, we know that $s(\eta) \leq \hat{\mathfrak{s}}_t$. For each such configuration, we perform the following modification of $\hat{y}_{\eta,t}$ – augment η by extra pages such that it's size becomes equal to $\hat{\mathfrak{s}}_t$. Let η' denote this configuration. Now assign the value $\hat{y}_{\eta,t}$ to $\hat{y}_{\eta',t}$ and then set $\hat{y}_{\eta,t}$ to 0. Thus, we have ensured that if $\hat{y}_{\eta,t} > 0$ then $s(\eta) = \hat{\mathfrak{s}}_t$. Clearly, this solution remains feasible and constraints (2.4) and (2.3) are satisfied with equality. To ensure equality of (2.2) constraints, we reduce the values $\hat{x}_{i,j(i,t)}$ till we get equality.

Now we bound the cost of this new solution. Clearly,

$$\sum_{i,j} \hat{x}_{i,j} \leq 2 \sum_{i,j} x^{\star}_{i,j}.$$

Further, it is easy to see that

$$\sum_{\eta,t} \hat{y}_{\eta,t} = \sum_{t} \hat{\mathfrak{s}}_t \leq 2 \sum_{\eta,t} y^{\star}_{\eta,t}.$$

This completes the proof of the lemma.

Rounding. The above lemma states that for any time t, the vector $\hat{\mathbf{x}}_t$ can be written as convex combination of characteristic vectors of configurations of pages which are not in the cache, where each configuration evicts at least $S - \hat{\mathfrak{s}}_t$ size. Since each such characteristic vector satisfies KC-inequalities for a knapsack of size $S - \hat{\mathfrak{s}}_t$ (see Definition 1), the overall vector $\hat{\mathbf{x}}_t$ also satisfies the KC-inequalities. So we can simply feed the sequence of vectors $\hat{\mathbf{x}}_t$, and $\hat{\mathfrak{s}}_t$ to the Theorem 2.2, and obtain a distribution over feasible solutions of cost at most a constant factor of optimal. This completes the proof of Theorem 1.4.

References

[1] Amazon Elasticache. `https://aws.amazon.com/elasticache/`, 2018.

[2] Microsoft Azure Cache. `https://azure.microsoft.com/en-in/services/cache/`, 2018.

[3] A. Adamaszek, A. Czumaj, M. Englert, and H. Räcke. An $O(\log\ k)$-competitive algorithm for generalized caching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1681–1689, 2012.

[4] N. Bansal, N. Buchbinder, and J. Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.

[5] N. Bansal, N. Buchbinder, and J. Naor. Randomized competitive algorithms for generalized caching. *SIAM J. Comput.*, 41(2):391–414, 2012.

[6] 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.

[7] L. A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[8] N. Buchbinder, S. Chen, and J. S. Naor. Competitive algorithms for restricted caching and matroid caching. In A. S. Schulz and D. Wagner, editors, *Algorithms - ESA 2014*, pages 209–221, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[9] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *1st USENIX Symposium on Internet Technologies and Systems, USITS'97, Monterey, California, USA, December 8-11, 1997*, 1997.

[10] R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 106–115, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

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

[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] L. Epstein, C. Imreh, A. Levin, and J. Nagy-György. Online file caching with rejection penalties. *Algorithmica*, 71(2):279–306, Feb. 2015.

[14] 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.

[15] A. Gupta, K. Talwar, and U. Wieder. Changing bases: Multistage optimization for matroids and matchings. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming*, pages 563–575, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[16] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[17] 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.

[18] N. E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002. Internet algorithmics.