CS880: Approximation and Online Algorithms	**Scribe:** Mikhail Nedbai
Lecture 19: Caching	**Date:** 11/01/2019

19.1 Caching Problem

A prblem of caching concerns maintaining a cache of size k of distinct pages from universe Ω. The input to the problem is an ordered sequence $\sigma \subseteq \Omega$. Upon the arrival of σ_i, the algorithm incurrs a cost if $\sigma_i \notin B$, and no cost otherwise. The algorithm then has to find a *victim* page inside of it's buffer, that will replaced by page σ_i, or *evicted*. An algorithm for serving these requests is often reffered to as a *page replacement policy*.

19.2 Unweighted caching

We first consider the case where the cost of evicting any page is 1. Given any determinstic algorithm, there exists a lower bound on it's performance for the caching problem, describd below.

Theorem 19.2.1 *No deterministic algorithm for the caching problem achieves a competitive ratio less than k*

The proof of this theorem will be omitted here. There are many intuitive policies that come to mind, when considering reasonable solutions for the caching problem, such as **FIFO**, a policy which will evict the page that has been brought into the cache the earliest, or **LRU**, a policy which will evict the least recently accessed page. Both of these policies, as well as other similar approaches achieve the same competitve ratio of k as described below

Theorem 19.2.2 *Both **FIFO** and **LRU** algorithms achieve a competitive ratio of k for the caching problem.*

We can improve this result by introducing randomness into our approaches. This modification will allow us to do better,but still will not yield a constant competitive ratio.

Theorem 19.2.3 *No randomized or deterministic algoritm for the caching problem can achieve a competitive ratio better than* $o(\log(k))$.

We present an algorithm, **Marker**, which achieves a logarithmic competitive ratio.

Definition 19.2.4 ***Marker Algorithm***
Each page maintains a bit b_i. *On initialization* $b_i = 0 \quad \forall i \in [k]$ *The algorithm proceeds in phases. A new phase begins whenever* $b_i = 0$ *for all pages. For every phase*

- *when a new page arrives*
  - *if* $b_i = 1 \quad \forall i \in [k]$, *then* $b_i = 0 \quad \forall i \in [k]$, *start new phase*
  - *otherwise, evict a uniformly random page i, such that* $b_i = 0$, *set* $b_i = 1$
- *when a page i present in cache arrives, set* $b_i = 1$

In order to properly analyze the competitive ratio of **Marker**, we need to introduce a hindsight optimal algorithm for the caching problem. We observe, that the hindsight optimal algorithm will always evict the page, that is requested farthest in the future. We can now proceed to analyze the **Marker** algorithm, which is also called **1 bit LRU**.

Theorem 19.2.5 *Algorithm **Marker** achieves a competitive ratio of $O\log(k)$ for the caching problem.*

Proof: We begin with an observation, that in any phase, plus the first request from the next phase, there are exactly $k+1$ distinct pages requested. Assume for instance a cache of size 3:

$$\overbrace{\underset{\times}{1},\underset{\times}{3},\underset{\cdot}{1},\underset{\times}{2}}^{\text{phase 1}},\overbrace{\underset{\times}{4},\underset{\times}{1},\underset{\times}{5}}^{\text{phase 2}},\overbrace{\underset{\times}{2},\underset{\cdot}{2},\underset{\times}{3},\underset{\cdot}{1},\underset{\cdot}{3},}^{\text{phase 3}}\overbrace{\underset{\times}{4}}^{\text{phase 4}}$$

Indeed, a beginning of a new phase indicates, that $b_i = 1 \quad \forall i \in [k]$, meaning that the cache is full with k distinct pages that have been accessed, additionally, the first request of the next phase must not have been present in the cache for the new phase to begin. This observation allows us to conclude that any algorithm must incurr the cost of 1 for every phase and one more request. Now, since the cache is of size k, deterministic **LRU** as well as **Marker** will incurr a cost of at most k for each phase, thus bringing the competitive ratio to k. While this bound is tight for **LRU**, a more carefull analysis of **Marker** can help us show, that this algorithm in expectation will do better, than incurring k misses each phase.

For phase i let $m_i =$ number of distinct pages accessed during phase i, that are not present in the cache in the beginning of the phase. Let OPT_i denote the number of misses incurred by **OPT** in phase i. It then follows from the argument above that

$$\begin{aligned} &\forall i > 1 \ : \ OPT_{i-1} + OPT_i \geq m_i \\ &\implies OPT \geq \frac{1}{2}\sum_i m_i \end{aligned}$$

We now attempt to provide a bound on the expected number of misses that **Marker** incurrs during each phase.

First observe, that the worst case scenario for **Marker** occurs when every new page arrives at the beginning of a phase, since it that case the algorithm will be forced to evict m_i pages from the cache, incurring the most potential misses from acesses to old pages. We then analyze sequences which start with requests to new pages only. Now consider the first request to an old page. The probability that it was evicted from the cache is $P[\text{first old page evicted}] \leq \frac{m_i}{k}$, since there are m_i new pages that could have evicted it. When the second old page arrives, the probability it was evicted from the cache is $P[\text{second old page evicted}] \leq \frac{m_i}{k-1}$, since the first page was already placed in the cache. We can generalize this result to conclude that the probability that i^{th} old page is evicted at the beginning is $P[\text{i old page evicted}] \leq \frac{m_i}{k-i+1}$. Thus the expected cost of the algorithm during a single phase is

$$\mathbb{E}[\text{ cost of } \textbf{Marker} \text{ in phase } i] \leq m_i + \sum_j \frac{m_i}{k-j+1} = m_i + m_i H_k$$

This allows us to calculate the total cost of the algorthm and its competitive ratio.

$$\mathbb{E}[\textbf{Marker}] \leq H_k \sum_i m_i$$

$$\frac{\mathbb{E}[\textbf{Marker}]}{\textbf{OPT}} \leq 2H_k$$

■

19.3 Weighted paging

The approach desribed above only works for the case, where the cost of each page is 1. We might imagine a more general case, where the cost of evicting a page i is w_i. For a period of time the solution to this more general version presented a significant challenge, but it was eventually overcome by utilizing Primal-Dual techniques. At a high level, we can construct a fractional LP for the caching problem by looking at it as a covering problem. Consider a problem with a $\Omega = \{1, 2, 3, 4, 5\}$ and a cache of size 3. Since the universe of pages in our case has 2 more pages, we maintain the invariant that at any point some two pages must be not present in the cache, and thus *covered* by our solution. We will then assign intervals to each page in the universe between each pair of requests to that page. When a new request arrives, the interval for the corresponding page ends, and it can no longer act as a cover, so another interval must be chosen. More formally, let p_t denote the page requested at time t. Let $t(p, j)$ denote the j^{th} time page p is requested. Let $x(p, j)$ indicate whether p was evicted from the cache at time $[t(p, j) + 1, t(p, j + 1) - 1]$. Let $r(p, t)$ denote the number of requests to p up to and including time t. Let $B(t) = \{p | r(p, t) \geq 1\}$ denote the set of pages requested up to and including t.

At time t, we are obliged to ensure that p_t is present in the cache, anf there are at most k pages occupying the cache. Therefore $B(t) \setminus \{p_t\}$ is to contain at most $k - 1$ pages. We can write that down as

$$\sum_{p \in B(t) \setminus \{p_t\}} [1 - x(p, r(p, t))] \leq k - 1$$

We can then present an exact LP formulation for the caching problem.

$$\begin{array}{ll} \text{minimize} & \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} w_p x(p, j) \\ \text{subject to} & \sum_{p \in B(t) \setminus \{p_t\}} [1 - x(p, r(p, t))] \leq k - 1, \forall t \\ & x(p, j) \in \{0, 1\}, \quad \forall p, j \end{array}$$

Relaxing the last constraint of the problem yields a fractional version of the caching probem. The fractional version can be solved using a primal-dual technique to obtain a $O \log(d)$ competitive solution, where d is the maximum number of intervals, or sets that each page belongs to. Since at each step there are at most $n - 1$ intervals that can cover a page, $d \leq n - 1$