CS 591, Lecture 14

Jared Saia
University of New Mexico

Class Outline

- Intro to Online Algorithms
- Ski Rental Problem
- Intro to Randomized Online Algorithms
- Paging

Online Algorithms

- An online algorithm is one where the input is received in an online manner and the output must be produced online
- Example 1: Renting Skis
- Example 2: Accessing Items in a List
- Example 3: Paging

Competitive Ratio

- Optimization problem: want to find feasible solution which optimizes some quantity (e.g. knapsack problem)
- For a given optimization problem input, I, we will let OPT *(I)* be the optimum solution
- For a given algorithm, ALG, and input sequence, I, let ALG *(I)* be the cost incurred be ALG on input I
- An online algorithm is said to be c-*competitive* if there is a constant α such that for all finite input sequences I,

$$ALG(I) \leq c * OPT(I) + \alpha.$$

Ski Rental Problem

- A set of skis costs x dollars per day to rent
- A set of skis costs kx dollars to buy
- Each day your kid decides either to 1) ski or 2) end her ski career forever
- Each day you decide whether to 1) rent skis or 2) buy skis
- Question: What's the best online algorithm for minimizing the money you spend?

The Algorithm

- There is a 2-competitive algorithm.
- The online algorithm: Rent until you've spent kx dollars on skis, and then buy them.

Analysis

Assume your kid skis for n days

- Case 1: $n \leq k$: OPT is nx and ALG is $n * x$, so the competitive ratio in this case is 1.
- Case 2: $n > k$: OPT is kx and ALG is $2 * kx$, so the competitive ratio in this case is 2.

The Paging Problem

- Memory: stores all possible pages
- Cache: can store a fixed set of k pages, (k is small)
- Given a request, for page p, p must be made available in the cache
- If p is already in the cache (a hit), we need do nothing, if p is not in the cache (a miss), we incur a page fault and must copy p into the cache
- On a page fault, we might need to "evict" another page to make room for p

The Paging Problem

- The pages that are requested arrive in an online manner
- We need to make decisions about which page to evict in an online manner
- We assume there is a cost of 1 to bring a page into the cache (a miss)
- We want an online algorithm which incurs the least number of misses
- This problem mirrors many real-world problems, including: efficiently utilizing a L1 cache, efficiently utilizing main memory, and caching web pages efficiently

Some Paging Algorithms

- LRU: When eviction is necessary, replace the page whose most recent request was earliest
- FIFO: Replace the page that has been in the cache longest
- FWF: On page fault, evict *all* pages from cache
- LFU: Replace the page that has been requested the least since moving into the cache

Theoretical Analysis

- LRU: k is the competitive ratio
- FIFO: k
- FWF: k
- LFU: *not competitive*

Upperbound

- We'll show the bound of k for FWF
- Divide the request sequence, σ, into *phases* as follows:
- Phase 0 is the empty sequence
- Phase i is the maximal sequence following phase $i-1$ that contains at most k distinct requests

Upperbound

Two important facts:

- FWF incurs no more than k faults per phase
- OPT incurs at least one fault per phase

This gives us a competitive ratio of k.

Lower Bound

- In fact, k is the best possible competitive ratio for any algorithm:
- Assume we've got $k + 1$ pages and for any algorithm, ALG, consider the sequence σ which always requests the one page ALG doesn't have in the cache
- ALG faults at every request on this sequence
- OPT faults at most once every k requests. (How?)

Empirical Analysis

- p. 41 in "Online Computation and Competitive Analysis" book
- Request sequences were generated by page request for "X windows" programs, a makefile program and a "disk copy" program
- $700,000$ requests, cache sizes ranged from 10 to 600
- Algorithm Rankings (Best to Worst): LRU, FIFO, FWF
- Both LRU and FIFO had competitive ratios less than 4 even as the cache size grew!

Problems

- k is not too impressive a competitive ratio
- Particularly since in practice, the competitive ratio seems to be much better
- Q: Can we do better if we use randomness?

Randomized Online Algorithms

- For a randomized online algorithm, we typically assume that the input sequence is created without knowledge of the random bits used by the algorithm
- Typically the input sequence is thought of as being created by an "adversary" who doesn't know the random bits
- An adversary who doesn't know the random choices made by our algorithm is called an *Oblivious Adversary*
- An online algorithm is said to be c-*competitive* if there is a constant α such that for all finite input sequences, I:

$$E(ALG(I)) \leq c * OPT(I) + \alpha.$$

Algorithm MARK

We will consider the randomized algorithm MARK defined as follows:

- Initially all pages are marked
- If there is a request for a page p which is in the cache but unmarked, then p is marked
- Otherwise if p is not in the cache, p is brought *marked* into the cache
- If all the pages in the cache are marked when p is brought in, they are all unmarked first
- The page that is evicted is chosen uniformly at random from all unmarked pages currently in the cache.

MARK Analysis

- Defn: $H_k = 1 + 1/2 + 1/3 + \cdots + 1/k$
- Fact: $\ln k < H_k < 1 + \ln k$
- *Theorem:* The competitive ratio of Mark is $2 * H_k$

MARK Analysis

- Fix a request sequence σ and partition it into phases of k unique requests as before
- For each phase i, call the pages in the cache just before the start of the phase the *old* pages and call all other pages requested in the i-th phase *new* pages
- Note that each request in the i-th phase is to either an old or a new phase
- Note also that repeated requests to a new or old phase don't contribute to our cost since each page requested is kept in the cache until the end of the phase

MARK Analysis

- Let m_i be the number of new pages requested in the i-th phase
- The worst order for the MARK algorithm is if all requests to new pages precede requests to old pages
- In this order, the requests to new pages incur m_i faults
- What about the remaining $k - m_i$ requests to old pages?

MARK Analysis

- What do we want?: We want to compute the expected number of misses on the $k - m_i$ old pages
- We can get this using linearity of expectation. We need to answer: What is the probability that the j-th old page requested will not be in the cache? (i.e. it will generate a fault)
- This probability is $m_i/(k-(j-1))$. Why?

Mark Analysis

Hence the number of faults incurred by MARK is:

$$
\begin{aligned}
m_i + \sum_{j=1}^{k-m_i} \frac{m_i}{k-j+1} &= m_i(H_k - H_{m_i} + 1) && (1)\\
&\leq m_i H_k && (2)
\end{aligned}
$$

Bound for OPT

It's not hard to bound OPT for this sequence:

- During the i-th and $i+1$-th phase, at least $k+m_i$ distinct pages have been requested
- Thus for each $i \geq 2$, the number of faults made by OPT in the i-th and $i+1$-th phase is m_i
- The number of faults made by OPT in the first phase is m_1
- Thus the total number of faults made by OPT is at least $1/2 * \sum_i m_i$

Take Away

- So MARK has no more than $H_k * \sum_i m_i$ faults
- OPT has at least $\sum_i m_i$ faults
- Hence the competitive ration is H_k