CS 224: Advanced Algorithms Spring 2017

Lecture 9 — Feb. 21, 2017

Prof. Jelani Nelson *Scribe: Gavin McDowell*

1 Overview

Today: office hours 5-7, not 4-6.

We're continuing with online algorithms. In this lecture:

1. List update problem
2. Paging

2 List Update

Parameters:

- n items in a linked list
- σ query sequence to touch items

Cost Function:

- in general, cost to touch ith item in list is $f(i)$
- for default cost function, $f(i) = i$
- also common: paging cost $f(i) = \begin{cases} 0 & i \geq k \\ 1 & i > k \end{cases}$ for a cache of size k

Goal: minimize sum of costs in processing the query sequence σ

Allowed Operations:

- free exchanges of accessed item with any item toward the front, i.e. bring item arbitrarily closer to front for free on query
- "paid exchanges", where we can transpose any adjacent items (not just the queried item) which are closer to the front than the queried item for a cost of 1.

Some other operations may be allowed, such as **insert(x)** at tail for a cost of the length of the list, and **delete(i)**, which is cost of item index, but these will not be used in the heuristics discussed.

Note: only one heuristic uses paid exchanges and it's not an important one

2.1 Heuristics

1. **MF** (move to front): when someone says touch item x, walk the list, move this to front of the list. aka LRU in paging cost function
2. **Transpose**: move accessed item one position closer to the front. gradually gain importance to move toward the front.
3. **FC** (Frequency Count): keep items sorted in decreasing order of frequency
4. **DF** (Decreasing Frequency): statically optimal strategy–keep items in sorted order decreasing by final frequency (must know future)

Note on Optimality:

- **static** optimality: strategy not allowed to change state
- **dynamic** optimality: strategy allowed to do free/paid exchanges

Notation:

- $C_A(\sigma)$ is the total cost of strategy A on sequence σ

First paper which proved a competitive ratio is due to Bentley, McGeoch, CACM '85 [1], who showed:

Theorem 1. *If no insert/delete operations are allowed, then $\forall$ access sequences σ, $C_{MF}(\sigma) \leq 2C_{DF}(\sigma)$ (if items in MF are initially sorted by time of first access)*

We won't prove this.

Notes:

- This is pitting **MF** against **static OPT**. i.e. you can move things but OPT cannot.
- This paper also showed that **Transpose** isn't good in a static setting, and that even **FC** can be forced to have unbounded competitive ratio in dynamic setting.

We will prove a theorem due to Sleator, Tarjan, JACM, '85 [2], which states

Theorem 2. $\forall\sigma, \forall A, C_{MF}(\sigma) \leq 2C_A(\sigma) + X_A(\sigma) - F_A(\sigma) - m$, *where A is some strategy (which might know the future), $X_A(\sigma)$ is the number of paid exchanges by A on σ, $F_A(\sigma)$ is the number of free exchanges, $C_A(\sigma)$ is total cost, not counting paid exchanges, and $m = |\sigma|$. Note that the actual cost is $2C_A(\sigma) + X_A(\sigma)$.*

Proof. **Potential function** argument. Assume that A and MF start with same state (either same order, or empty list, requiring insertions). Define potential Φ to be the number of inversions in **MF's** list using the sorted order of **A's** list.

Recall for potential functions:

$$\text{(total amortized cost)} = \text{(total actual cost)} + \Phi_{final} - \Phi_{init}$$

$$\Rightarrow \text{(actual cost of MF)} = \text{(amortized cost of MF)} + \Phi_{init} - \Phi_{final}$$

Initial potential is 0, and final potential is nonnegative, so this is $\leq$ (amortized cost). Thus it's enough to bound the amortized cost (which we can do for every single operation, not just sum)

Bound on amortized cost:

search(x): Suppose x is at position i in A's list, k in MF's list. We have

$$\text{(am cost)} = \underset{actual\,cost}{k} + \Delta\Phi$$

Suppose also that t items precede x in MF's list, but are after x in A's list. Thus $k-t-1$ items precede x in both lists (all the things remaining before x except those k).

By moving x, **MF**

- creates $k-t-1$ new inversions (for those items above)
- undoes t inversions

Thus, $\Delta\Phi = k - 2t - 1$ so

$$\text{(am cost)} = 2k - 2t - 1 = 2(k-t) - 1$$

Since x is the ith item in A's list. We know that $k-t-1 \leq i-1$ because x is at i in A's list, and there are $k-t-1$ items preceding x in both lists. Thus,

$$\text{(am cost)} \leq 2i - 1$$

Because any free exchanges that are performed by A move x closer to the front, it can only undo some number of inversions (all including x) as it doesn't otherwise change the order. Thus any inversions A performs will be compensated by the potential function and thus this upper bound includes any free inversions, so we can neglect $F_A(\sigma)$. Any paid exchanges will increase the value of the RHS of the statement we wish to prove. Thus we take the worst case of this, where $X_A(\sigma) = 0$, providing the tightest bound, and can see that the bound we have proved is in fact bounded by the RHS of the equation above.

Since $C_A(i) = i$ know that in the sum over all m terms, we have the amortized cost of $C_{MF}(\sigma) \leq \sum_\sigma (2i-1) \leq 2C_A(\sigma) + X_A(\sigma) - F_A(\sigma) - m$.

□

Note: there are short (almost 1-line examples) to break **FC** (i.e. demonstrate unbounded competitive ratio). Left as an exercise.

3 Paging

A more cared-about problem. There are n items, and we have a cache of size k. If it's in cache, we can fetch for free. If not, have to bring it in from memory (evicting someone else from the cache). Evictions cost 1. How do you pick who to evict?

1. σ is a sequence of item requests
2. if some item is already in in cache, pay 0
3. else pay 1 and must bring item into cache (pick evicted member if the cache is full)

Must come up with an **eviction policy**.

3.1 Eviction Policies:

1. **LRU**: Least Recently Used (equivalent to MF)
2. **FIFO**: First-in First-out (evict member that's been there the longest by timestamp)
3. **LFU**: Least Frequently Used (almost like Freq Count, but requires accessed item to be brought into the cache)
4. **MIN**: MIN algorithm evicts the page that will be requested furthest in the future (Belady, IBM Sys.J., '66 [3]), which is OPT

Another set of theorems, all due to [2]

Theorem 3. *1.* **LFU** *is terrible (unbounded compet. ratio)*

2. **LRU** *is k-competitive*

3. **FIFO** *is k-competitive*

4. No (deterministic–can get $\ln k$ is the best possible compet. ratio w/ randomization) strategy can have competitive ratio $< k$

We will prove (2) and (4).

Proof. of (4): Can't beat k-competitive determinstic. Let A be our strategy, and choose $n = k + 1$. In order to make it is difficult as possible, our adversary can keep requesting the one page in the universe which is not currently in A's cache. In this case, $C_A(\sigma) = |\sigma|$ because we have to pay 1 for every single request.

Note that we can't play this game with **MIN** because **MIN** has to know all of σ, whereas in this case we are letting our adversary pick requests in sigma dynamically, making it as hard for A as possible.

Thus, A will fault on every page, but **MIN** (which is OPT) will only fault once every k requests, because it will evict the page requested furthest in the future, meaning that all k other pages will be requested before it is required again.

□

Proof. of (2): LRU is k-competitive. Will show **1-bit LRU** is k-competitive, which suffices because LRU is an implementation of 1-bit LRU.

Definition of 1-bit LRU:

- initially, all k pages in cache are unmarked
- when p is requested, mark p (whether or not it was already in the cache)
- if no room in cache, evict any unmarked page
  - if all pages marked, unmark all
  - evict any unmarked page
- can think about σ in "phases", where a "phase" starts each time we unmark all pages

Actual LRU is an implementation of 1-bit LRU. Since we're not allowed to evict marked person, and unmarked people have not touched in this phase (touched in last phase or earlier), the least recently used person must be unmarked. We're allowed to evict LRU person, so we can choose to evict LRU person in our implementation.

Note: easy randomization gets you $2 \ln k$ (pick random unmarked page to evict)

Analysis of 1-bit LRU: will center around the number of page faults per phase for different strategies.

For **1-bit LRU**, we have k distinct pages accessed in a phase. Since we have at most k distinct pages, at most k of them were not in the cache in this phase. Thus, we can do no worse than k page faults per phase.

For **OPT**, in order for a phase to end, all pages must be marked, and a page which isn't in the cache has to be requested (to start the new page). Thus, in this phase (counting the one which starts the following phase), we have accessed $k + 1$ distinct pages, which means that OPT must fault ≥ 1 per phase.

Thus, if **OPT** faults at least once per phase, and **1-bit LRU** faults at most k times perm phase, we have a competitive ratio of k. □

Question: how much better is LRU than other 1-bit LRU implementations?

Open Question: for randomized paging with resource augmentation: what's the best competitive ratio?

Note: This is pretty bad! If k is huge (like megabytes) this only gets us within a factor of millions of OPT.

3.2 Resource Augmentation

Definition 4. Resource Augmentation *[2] or "Bicriteria approximation". LRU has a cache of size k, but only allow OPT to have a cache of size h. Would like to be competitive against weakened OPT.*

Theorem 5. *FIFO + LRU both achieve competitive ratio of $\leq \frac{k}{k-h+1}$ in the resource augmentation model.*

Question: What about randomized strategies?

Answer: Depends on the allowed adversary

3.3 Types of adversaries:

a. **all-powerful**: knows your random coins. (e.g. in LRU, adversary knows which random page you're going to evict) deterministic to adversary, so can't beat k-competitive

b. **adaptive**: can choose next request based on machine state (what's in the cache, not in cache)–gets to see all parts of machine at all times, so knows cache state at all times (could also weaken this, only knowing page faults, not knowing which pages are evicted)

c. **oblivious**: must fix σ in advance–knowing code but not random coins

Due to (Fiat, Karp, Luby, McGeoch, Sleator, Young, J.Alg '91 [4]):

Theorem 6. *$\exists$ algorithm "Mark" which is $2H_k$-competitive against oblivious adversaries, where $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} = \ln k + O(1)$ is the kth harmonic number.*

$$\forall \sigma, \mathbb{E}C_{Mark}(\sigma) \leq 2H_k \cdot C_{OPT}(\sigma) + O(1)$$

Notes:

- With radnomized algorithm, competitive ratio refers to expected cost of algorithm, as performance is a random variable.
- This analysis doesn't work with an adaptive adversary.

Proof. Note that $\Omega(\ln k)$ is necessary even with randomization.

Again pick $n = k + 1$, and let $m = |\sigma|$. Fix algorithm $A = A_s$, where s is its random seed. This is because randomized algorithms can be viewed as deterministic algorithms with an extra input (the seed). We also pick a uniformly random sequence σ. Then we have

$$\mathop{\mathbb{E}}_{\sigma,s} C_A(\sigma) = \mathop{\mathbb{E}}_{s}\mathop{\mathbb{E}}_{\sigma} C_{A_s}(\sigma) = \mathop{\mathbb{E}}_{s,\sigma} \sum_{\text{operations } i \in \sigma} \mathbb{E}I(\text{page } i \text{ not in cache}) = \frac{m}{k+1}$$

Since we have here the sum of independent bernoulli random variables. With high probability, this is very close to the real value (for large m).

OPT evicts the page it'll see last, so how long does it take to see all of these pages? This is the **coupon collector** problem. It turns out to be something like $(k+1)\ln(k+1)$. Thus OPT fails a $1/((k+1)\ln(k+1))$ fraction of the time, while A_S fails like $1/(k+1)$ fraction of the time. By union bound, with high probability, there's a single σ that's hard for both A and OPT.

Pick random σ, which has positive probability of being hard for both. Therefore $\exists\sigma$ which is hard for both at the same time. Thus there's an access sequence which gives us exactly $\ln k$, with a factor of 2 gap between our upper and lower bounds. □

Proved Later:

- H_k-competitive "Partition" alg. (McGeoch, Sleator, Algorithmica '91 [5])
- H_k-competitive "Equitable" alg. (Achlioptas, Chrobak, Noge, Theor. C.S. '00 [6])
- $2\ln\frac{k}{k-h+1}$-comp. possible, $\ln\frac{k}{k-h+1}$-comp. impossible (Young, '91 [7])

References

[1] Jon L. Bentley, Catherine C. McGeoch. Amortized Analyses of Self-organizing Sequential Search Heuristics. *Commun. ACM*, 28(4):404–411, 1985.

[2] Daniel Sleator, Robert Tarjan. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM*, 28(2):202–208, 1985

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

[4] Amos Fiat, Richar Karp, Mike Luby, Lyle McGeoch, Daniel Sleator, Neal E. Young. Competitive Paging Algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[5] Lyle McGeoch, Daniel Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816–825, 1991.

[6] Dimitris Achlioptas, Marek Chrobak, John Noga Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234:203–218, 2000.

[7] Neal Young Competitive paging and dual-guided algorithms for weighted caching and matching. (Thesis) *Computer Science Dept., Princeton University*, 1991.