CS 224: Advanced Algorithms Fall 2014

Lecture 8 — Sept 25, 2014

Prof. Jelani Nelson *Scribe: Jean Pouget-Abadie*

1 Online Algorithms - Overview

In this lecture, we are going to cover 'Online algorithms' which study decision-making in the face of uncertainty. We suppose there is some cost to be paid for making a decision. At the end of a sequence of requests, we want to make sure we didn't spend much more than the least we could've spent in hindsight.

Definition 1. *Let OPT be the algorithm that makes the best sequence of decisions one could make if we knew the series of requests in advance. Denote by $\sigma(\sigma_1, \sigma_2, \ldots, \sigma_n)$ a series of requests. An decision-making algorithm A is* **r-competitive** *if*

$$\forall \sigma, cost_A(\sigma) \leq r \cdot cost_{OPT}(\sigma) + O(1)$$

2 Examples

2.1 Ski rental

Suppose you are going on a ski trip with friends. Each day, your friends will either decide to stay and ski on more day or decide to leave, such that you do not know ahead of time how long you will keep on skiing. The ski shop gives you two options: you can either rent skis for 1$/day, or you can buy skis for B$ after which you do not need to rent skis anymore.

The OPT algorithm is omniscient and knows exactly how long days N you will ski. If $N > B$, the OPT algorithm decides to buy skis, and rent skis every day otherwise. Without knowing N ahead of time, one algorithm is to rent skis for B days, then buy skis on the $B + 1^{st}$ day. We show that this algorithm A is 2-competitive.

1. If $N \leq B$, then $\text{cost}_{OPT} = N$ and $\text{cost}_A = N$.
2. $If N > B$, then $\text{cost}_{OPT} = B$ and $\text{cost}_A = 2B$

2.2 Free pizza

Suppose you are in the center of a very long hallway, with $\frac{n}{2}$ evenly-spaced rooms on either side. You heard from you friend that one of the rooms in the hallway has free pizza. However, your phone has run out of battery and you must peek into the rooms to find the free pizza. Also assume that walking past a room is enough to decide whether there is free pizza inside, such that you pay exactly the number of rooms you walk past.

Index rooms from $-\frac{n}{2}$ to $\frac{n}{2}$ from left to right, placing yourself at 0 and suppose the room with the free pizza is at P. The OPT algorithm knows exactly where the pizza is and walks past $|t|$ rooms. Algorithm A consist of visiting 1 room on your right, then doubling back and visiting all the rooms until you reach room number -2, doubling back again to visit room number 4 such that you zigzag from left to right, doubling back at $1, -2, 4, -8, 16 \ldots$. We claim that algorithm A is 9-competitive.

Each time you visit a room, you pay the price to reach that room (equal to the room number) and the price to double-back and reach zero, such that in the worse case, you pay

$$S := (1+1) + (2+2) + (4+4) \ldots (2^m + 2^m) + t$$

Notice that $2^m \leq 2t$ since in the worse case, you double-back at $\pm 2^m$ before going to $t = \pm 2^{m+1}$. Therefore $S \leq t + 2(1 + 2 + 4 + \ldots 2t) \leq t + 2(4t) = 9 \times \text{cost}_{OPT}$.

3 List Update

We are going to study two mechanisms taken from Sleator and Tarjan's paper [1]: List Update and Paging. The latter is covered in the next section.

Suppose we can hold up to n items in a linked list, which supports the following operations :

1. **access(x)** : Pointer starts at beginning of linked list and goes to item x.
2. **insert(x)** : Puts item x at the end of the list
3. **delete(x)** : Deletes node x from the list

After accessing an item, you are allowed to move it to any position that is closer to the front of the list at no cost. We further suppose that accessing item i costs us i. Sleator and Tarjan discuss several heuristics in [1]:

1. **Move-to-Front (MF)**: always move the accessed item to the front of the list
2. **Transpose**: transpose accessed item with the item immediately before it in the list
3. **Frequency count**: keep items in sorted order of frequence of accesses (most to least frequent).

Bentley, McGeoch prove in [2] the following theorem,

Theorem 2. *Let s be a sequence of access requests. If no insertions are allowed and items are initially sorted by time of first access, then*

$$\forall s, cost_{MF}(s) \leq 2cost_{FC}(s)$$

We can think of this result as a static-optimality property for the Move-to-Front heuristic. Sleator and Tarjan [1] prove in the same year the two following results:

Theorem 3. *Move-to-Front is dynamically optimal: its competitive ratio is $O(1)$*

Theorem 4. *Let A be any algorithm and we assume that at the beginning of access sequence s, A and MF have all items in the same order, then*

$$\forall s \text{ s.t. } |s| = m,\ cost_{MF}(s) \leq 2cost_A(s) + X_A(s) - F_A(s) - m$$

where X_A denotes the paid exchanges and F_A denotes the free exchanges.

Proof. Let Φ be the number of inversions in MF's list using the ordering specified by A's list. For example, if an item i comes before j in MF's list but after J in A's list, then we count one inversion. By definition of the optimal cost and the chosen potential function Φ, we only need to bound the amortized cost.

$$\text{Total cost} = \text{Amortized cost} + \Phi_{init} - \Phi_{final} \qquad \leq \text{Amortized cost} + \Phi_{init} \text{ (with } \Phi_{init} = 0)$$

By definition of the amortized cost, 'Amortized cost' ='Actual Cost' $+\Delta\Phi$ Suppose queried item x is the i^{th} item for A and the k^{th} item for MF. Then we have 'Actual Cost' $= k$. What of $\Delta\Phi$? Suppose there are t items before x in MF's list that are after x in A's list. Accessing x and moving it to the front undoes t inversions and creates $k - t - 1$ new inversions:

$$\Delta\Phi = k - t - 1 - t = k - 2t - 1$$

Notice that there are at most $i - 1$ items before x in A's list but at least $k - t - 1$ items, therefore $k - t \leq i$. It follows that

$$\text{Amortized cost} = k + (k - 2t - 1) = 2(k - t) - 1 \leq 2i - 1$$

which concludes our proof. The analysis of insertion is similar. For deletion we create no new inversions and thus have amortized cost $k - t \leq i$.

Free exchanges by A have amortized cost -1, giving the additive $-F_A(s)$ term. Paid exchanges by A have amortized cost at most 1, giving the additive $X_A(s)$ term. □

4 Paging

What if accessing the i^{th} item in the list costs $f(i)$ and not just i? In the paging model, we suppose that we have K lines of cache, which cost nothing to access, and n items split between RAM and cache. Fetching an item from RAM costs 1. If a queried item is not in cache, we must choose to place it in the cache (and therefore evict an item from the cache). In this setup, $f(i) = 0$ if $i \leq K$ and 1 otherwise.

There are several replacement policy heuristics:

- **Least Recently Used (LRU)** : Evict the least recently used item from the cache (equivalent to Move-to-Front)
- **Least Frequently Used (LFU)** : Remove the least frequently used item from cache.
- **First in First out (FIFO)** : remove the most recently queried item from cache

Belady shows in [3] that the optimal (omniscient) algorithm is the one selecting the farthest queried item in the future. The followin theorem is due to Sleator and Tarjan [1]:

Theorem 5. *LRU and FIFO are both K-competitive. Furthermore, no online algorithm can have a competitive ratio* $< K$

Proof. We prove the second claim. Assume the total number of pages is $K+1$. If we always query the item that A just evicted, A will fault everytime. However, OPT will evict the furthest item in the future and faults only at most every K queries. □

We now describe another class of heuristics : **1-bit LRU**. Initially, all pages are unmarked. When an access comes in, if we must evict an item from cache, we evict an unmarked item (randomly or according to a certain procedure). Every time an item is accessed, we mark it. If all pages are marked, then once we need to evict, we clear all marked bits.

Theorem 6. *1-bit LRU is K-competitive*

Proof. Notice that since LRU is a special case of 1-bit LRU, we have: 1-bit LRU is K-competitive implies that LRU is K competitive. As such, by proving Theorem 6 we prove the second claim of Theorem 5. Break the sequence into phases which are seperated by the 'clear all marked bits' operation. Within a phase, 1-bit LRU has at most K faults. OPT has at least 1 fault. □

5 Circumventing Theorem 5

There are several methods for circumventing the lower bound on the competitivity ratio indicated by Theorem 5:

5.1 Resource Augmentation

If we allow ourselves to have K slots in cache but only give $K' < K$ slots to the OPT algorithm, then Sleator and Tarjan show in [1] that LRU and FIFO are $\frac{K}{K-K'+1}$–competitive.

5.2 Randomization

Suppose that the cache replacement policy can make random decisions. We can distinguish three types of adversaries:

1. Omniscient: knows all random coins flipped, in past and future
2. Adaptative: takes into account past random coins flipped, but doesn't know future randomness.
3. Oblivious: chooses access sequence s at the beginning, before any coins are flipped.

Let us define a $r - competitive$ randomized algorithm.

Definition 7. *Let s be a sequence of access queries. A* **randomized** *algorithm A is* **r-competitive** *if*

$$\forall s, \mathbb{E}(cost_A(s)) \leq r\, cost_{OPT}(s) + O(1)$$

The following theorem is due to Fiat, Karp, Luby, McGeoch, Sleator, and Young [4]:

Theorem 8. *There is a randomized algorithm with a competitive ratio of $H_K := \sum_{j=1}^{K} \frac{1}{j} = \ln(K) + O(1)$. Furthermore, any randomized algorithm has a competitive ratio $\Omega(\log k)$*

Consider the following **Marking** algorithm The marking algorithm is a special case of 1-bit LRU, where the evicted unmarked page is chosen uniformly at random.

Theorem 9. *The marking algorithm is $2H_k$-competitive, where H_k is defined as above.*

Proof. We break the access sequences into phases. We will call a page **clean**, if it was not accessed in the last phase and not yet accessed in this phase. We call a page **stale** if it was accessed in the last phase but not yet in this phase. We will show (next time) that there are L clean accessed in a phase. In this case, OPT pays at least $\frac{k}{2}$ and M pays at most $2LH_K$ in expectation. □

References

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

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

[3] László Bélády. A Study of Replacement Algorithms for Virtual-Storage Computer *IBM Systems Journal*, 5(2):78–101, 1966.

[4] Amos Fiat, Richard Karp, Michael Luby, Lyle McGeoch, Daniel Sleator, Neal Young. Competitive Paging Algorithms *J. Algorithms* 12(4):685–699, 1991.