Algoritmos e Incerteza **(PUC-Rio INF2979, 2018.2)**

Lecture 3: Online Algorithms

2018-08-28

Lecturer: Marco Molinaro *Scribe: Leonardo Oliveira*

In this class we analyze the paging problem a little further, and see how the Least Recently Used (LRU) algorithm compares to the optimal offline solution. We also see the AdWords problem.

1 The paging problem

In the paging problem we have a small cache memory where only *k* pages can be fit and a sequence of pages ranging from 1 to $\mathcal{K}$. Whenever a page is requested and the cache is full, one page must be removed and the algorithm pays a cost of 1.

Simple example In a first toy example, let's consider an instance where

$$\mathcal{K} = k + 1$$

And the following sequence is requested:

$$\{1, 2, 3, ..., k, k + 1, 1, 2, ...k\}$$

In the optimal solution, the furthest in future (see [4] for complete proof of this claim) algorithm will select the page k to replace the (k+1)-th request. After that, it will have another miss only when the page k is request (cost 2). The Least Recently Used (LRU) algorithm will always evict the oldest page in cache. So, it will remove the first page (page 1) in cache when the page k+1 is requested. But the instance will request it in the next round, forcing another miss. The total cost will be k+1. Thus, the LRU is, at least, $\frac{k}{2} - competitive$

Lemma 1.1. *Any deterministic algorithm is, at least, k-competitive*

Proof. The optimal solution is at least as good as the strategy that evicts the page that will be requested further in future. The optimal solution has a miss after at least k steps (given that $\mathcal{K}$ = k + 1): let p be the page that was just evicted in a arbitrary step. As the algorithm was designed, p is the further page to be requested in the future in the cache of the previous time step. So, there is at least one request for one of the other k-1 pages in the cache between now and the next time p will be requested. Therefore, there are $\frac{n}{k}$ misses in the optimal solution (given n is the number of pages in the request sequence) The general worst-case scenario for a deterministic algorithm will be that instance that requests exactly the page that was just evicted. In this case, the general algorithm will have n - K misses Comparing this general algorithm with the optimal one, we have

$$\frac{n-k}{\frac{n}{k}} = \frac{k*(n-k)}{n} = k*(1-\frac{k}{n})$$

$$\lim_{n\to\infty} k * (1 - \frac{k}{n}) = k$$

therefore, the general algorithm is, at lease, k-competitive □

1.1 Some definitions

Definition 1.2. A *phase* is defined to be the greatest prefix of a sequence that contains k distinct pages, since the start of the sequence or the end of the previous phase. The first phase starts at the beginning of the sequence and there are as many phases as needed. The second phase starts at the end of the first and is the prefix of the rest of the sequence that contains k distinct pages. A phase chance can be detected in a online way, when the (k+1)-th distinct element appears in the phase.

Example: Consider $\mathcal{K}$ = 10 and k = 2. The phases in the sequence below are

$$(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)$$

Phases Algorithm : The Phases Algorithm is designed to evict all pages when a new phase is detected. So, it has a total cost of $k * [numberofpages]$. Note that all pages pages in a phase fit into the cache and no cache miss occurs.

Definition 1.3. An algorithm is said to be *marking* if any page that it loads into cache in a phase stays in cache until the end of the phase. Note that a page that was already in cache when the phase started can be evicted once and then reloaded. But once reloaded, it must stay there until the end of the phase.

Phases Algorithm is marking : As the algorithm is designed, all pages during a phase will only be loaded to cache, so no misses will happen and therefore no page will be evicted during the phase.

Lemma 1.4. *The OPT algorithm will have a cost of, at least,* $\frac{([numberofphases]-1)}{2}$

Proof. Consider two consecutive phases. By the definition of phase, when the second phase starts, any solution must evict a page to make room for the k+1 page that is being requested. So, the optimal solution must have at least 1 miss per page. In the end it will have a cost of, at least, $\frac{([numberofphases]-1)}{2}$ □

LIFO algorithm is not marking . Considering the algorithm that evicts the most recently page, it is not a marking algorithm. This can be seen in a instance where $\mathcal{K}$ = 3 and k = 2. The following sequence is grouped in this manner:

$$(1, 2), (3, 2, 3, 2, 3, ...)$$

When the second phase begins, it will evict the 2 and load the first 3. When the first 2 of the sequence is requested, it will evict the 3 that was loaded in the current phase. This is not a marking algorithm.

Lemma 1.5. *Any marking algorithm has a total cost of, at most, k * [number of pages]*

Corollary 1.6. *Marking is approximately k-competitive*

Proof. The proof the straightforward as we use the definition of a marking algorithm. As any page will not be evicted after loaded during a phase, there can be no more than k evictions by phase. In total then there are k * [number of pages]. Comparing with the OPT solution Using 1.4 and 1.5:

$$\frac{k * [numberofpages]}{[numberofpages] - 1} \approx k$$

□

1.2 LRU Algorithm

Theorem 1.7. *LRU is marking*

Proof. Consider a phase. Let $\mathcal{P}$ be the set of all k pages in the phase, let S_t be the set of all pages added to the cache during this phase. We show by induction on $t \in phase$ that $S_t \subseteq cache_t$

Base case For t, assume $S_t \subset \mathcal{P}$ For the induction step, there are two cases

Case 1 There is no miss in t+1. In this case,

$$Cache_{t+1} = Cache_t$$

$$S_{t+1} = S_t$$

Since

$$S_t \subset Cache_t$$

then

$$S_{t+1} \subset Cache_{t+1}$$

Case 2 There is a miss in t+1. When this miss occurs, the page p_{t+1} is added to the cache. So, $p_{t+1} \in \mathcal{P}$, but also $p_{t+1} \notin S_t$. This, implies that there at least 1 element of difference between the sets and that $|S_t| < |\mathcal{P}| \leq k$. So there are other pages in the $cache_t$ that can be removed and were added in previous phases, i.e. pages in $cache_t \setminus S_t$. By the definition of the LRU, it will remove the oldest in cache. As all pages in S_t were added in the current phase, there is a page older than all pages in S_t that can be removed (the page p_t). So, $cache_{t+1}$ contains both S_t and p_{t+1} and we can say that

$$S_t + p_{t+1} \subseteq cache_{t+1}$$

$$S_{t+1} \subseteq cache_{t+1}$$

□

1.3 Randomized algorithm

As seen in previous sections, there is no deterministic algorithm that is better than k-competitive. One way to improve it is to add some randomization. In [1], the author show the 1-bit LRU algorithm that can achieve $log(k) - competitive$. The general idea is to add some random way to evict a page. The general algorithm works like this: when a page is loaded to cache, mark the page as new. If a page must be evicted, chooses at random any page that is not marked as new. If all pages are marked as new, un-mark all and choose any one at random. The way this version of LRU works is to use a single bit (if the page is marked or not) to separate older pages from new ones.

2 Ad-words Problem

The problem is a simpler version of the real ad-words from Google. In this problem, it is assumed to exists an advertiser *i* that has interest in some words and a maximum overall budget. The advertiser *i* can pay up to b(i,k) for the word *k*. The maximum amount spent during the day cannot exceed B(i). To simplify the notation, let use the following notation for the amount of money earned in time *t* for the advertiser *i* and word *k*

$$\tilde{b}(i,k) = min(b(i_t, k_t), B(i) - [\text{money spent by i up to t}])$$

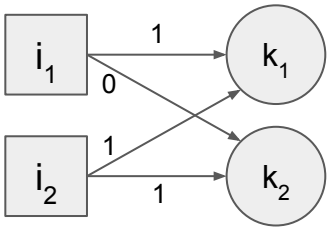
In this version, it is assumed that only one word is searched at a time. Also, if a word is search many times, each occurrence of it will be considered a different word. Only one ad can be shown by word

Observation 2.1. *The offline version of the AdWords is NP-hard. But if the values of b are assumed to be small (compared to B), it can be shown that it has a cost of* $1 - \epsilon$*, with* ϵ *small.*

2.1 The Game

This problem can be understood as a game in which one player chooses a word *k* of interest and the other chooses which ad will be displayed for that word. For the chosen ad *i*, the second player earns $\tilde{b}(i,k)$. The goal is to maximize the total money earned. Note that in order to maximize the total money earned it is necessary to choose the ads in a balanced way so the advertiser does not run out of budget too quickly. Another interesting fact about this game is that any association of words and ads is valid. The difference from one solution to another is how much money is earned in the end.

Example. In the following example we show how much each of two advertisers would pay for two words.



Consider the budgets are $B_1 = B_2 = 1$. There are (at least) two possible solutions when the first word k_1 arrives. If i_1 is paired with k_1 and i_2 with k_2, we receive a total of 2. If i_2 is paired with k_1, that leaves i_1 with k_2, with total of 1. We can see that future information is important to determine the optimal solution. Another solution would be to pair i_2 with both k_1 and k_2. In that case, if the total budget were $B_2 \geq 2$, then the total received would also be 2. But, if $B_2 = 1.5$, the total received would be $1 + \min(1, 1.5 - 1) = 1 + 0.5 = 1.5$

2.2 Greedy algorithm

An online solution for this problem is a greedy algorithm that chooses the ad from advertiser *i* that maximizes $\tilde{b}(i, k)$.

Theorem 2.2. *The greedy algorithm is $\frac{1}{2}$-competitive*

Observation 2.3. *If a budget B_i is decreased of ϵ, the total amount of money earned by a solution is decreased, at most, by ϵ*

Proof of Theorem 2.2. Consider the choices made by the greedy algorithm when the first word arrives. Let's assume that the ad chosen by the greedy algorithm was $\bar{i}$ and the optimal solution would choose i^*. We can create a new solution of the problem called I' by taking the original instance I and removing the first keyword (k_1) and decreasing the budget of the selected ad by $\tilde{b}(\bar{i}, k_1)$. Because of observation 2.3, we can say that

$$OPT(I) \leq OPT(I') + \tilde{b}(i^*, k_1) + \tilde{b}(i^*, k_1) \tag{1}$$

The first term is due the value of selecting the ad i^* and the second one is due the budget decrease. Note the sign is $\leq$. As the greedy algorithm will select the ad that maximizes $\tilde{b}$, then we can apply a lower bound for OPT(I) as

$$OPT(I') \geq OPT(I) - 2 \times \tilde{b}(\bar{i}, k_1) \tag{2}$$

If we apply an induction in the size of I, we can see that $OPT(I') \leq 2 \times Greedy(I')$. Applying the lower bound 2 in this induction, we can show that

$$OPT(I) \leq 2 \times (\tilde{b}(\bar{i}, k_1) + Greedy(I'))$$

$$OPT(I) \leq 2 \times (Greedy(I))$$
$$Greedy(I) \geq \frac{1}{2} \times OPT(I)$$

□

Example. (Tight example for greedy) Suppose there are 2 keywords and 2 advertisers, both with budgets $B_1 = B_2 = 1$. The first keyword gives value 1 to each advertiser. Suppose the first advertiser is matched to this keyword. Then the second keyword gives value 1 to advertiser 1, and 0 to advertiser 2; the algorithm cannot obtain any value from it. Thus, the algorithm obtains total value 1, while OPT gets value 2.

3 Further Readings

3.1 Better algorithm for Ad-words

The greedy algorithm is not the best algorithm for this problem. In [5] the authors show an algorithm named Balance that achieves competitive ratio of $1 - \frac{1}{\epsilon}$. Without going in too many details, this algorithm is also greedy, but instead using the simple value b defined in our example, it uses the following trade-off function

$$\phi(x) = 1 - \epsilon^{-(1-x)} \tag{3}$$

which represents how much an advertiser has its budget already occupied by previous words. One observation is that the competitive ratio holds only if $b_i \ll B_i$.

A better algorithm Although this limitation is real from an engineering point of view, in [2], the author shows an algorithm that achieves the same bound but without the limitation.

Multiple ads by word In [6] there is a chapter exclusive for the Ad-words problem where the authors detail other algorithms and, in special, they show a version where you can have multiple ads shown for the same word.

General framework The primal-dual framework is a general way of thinking not only the Ad-words problem but also the Paging problem and the Ski rental problem. In [3], the author show examples on how to solve this problem with this framework.

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat. Heat and dump: competitive distributed paging. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 22–31, Nov 1993.

[2] N. Buchbinder and J. Naor. Improved bounds for online routing and packing via a primal-dual approach. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 293–304, Oct 2006.

[3] N. Buchbinder and J. Naor. *The Design of Competitive Online Algorithms via a Primal-Dual Approach*. now, 2009.

[4] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2006.

[5] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized on-line matching. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 264–273, Oct 2005.

[6] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.