Algorithms and Uncertainty, Winter 2017/18 Lecture 2 (4 pages)

Online Paging

Instructor: Thomas Kesselheim

Today, we will consider a very classical problem in online algorithms called *Paging*. We have a system with a large, slow memory and a small, fast memory (cache). In the large memory, there are n pages stored, the small memory has space for only $k \ll n$ pages. Our algorithm will have to decide which pages to keep in the small memory.

In our model, this means that initially we start with a cache of k arbitrary pages. Our algorithm is then presented a sequence of pages $\sigma_1, \sigma_2, \ldots \in \{1, \ldots, n\}$. If σ_t is not contained in the cache at time t, a page fault occurs and our algorithm incurs a cost of 1. It then has fetch σ_t from the slow memory and choose which one of the k pages in the cache it should replace.

Example 2.1. *In this example, the cache has size $k = 4$ and there are $n = 5$ pages overall.*

pages in cache	*page requested*	*page evicted*	*cost*
1, 2, 3, 4	*1*	—	*0*
1, 2, 3, 4	*5*	*4*	*1*
1, 2, 3, 5	*5*	—	*0*
1, 2, 3, 5	*4*	*2*	*1*

Again, we will analyze this problem in competitive analysis. An algorithm is c-competitive if we have

$$\mathbf{E}\left[\text{cost}(\text{ALG}(\sigma))\right] \leq c \cdot \text{cost}(\text{OPT}(\sigma)) + b \qquad \text{for any sequence } \sigma \tag{1}$$

for some constant b. (Note that we now have a minimization problem, therefore the inequality is switched.)

1 Algorithm LRU

Our algorithm has to decide which page to evict in case of a page fault. Let us consider the very natural strategy *least recently used* (LRU). Here, we evict the page from the cache that has not been requested for the most number of rounds by now.

Theorem 2.2. *LRU is k-competitive.*

To show that LRU is k-competitive, we will devise an auxiliary algorithm *1-bit LRU*, which is underspecified because we do not define how to deal with ties. No matter how ties are broken, it will be k-competitive and LRU will correspond to one particular tie-breaking rule.

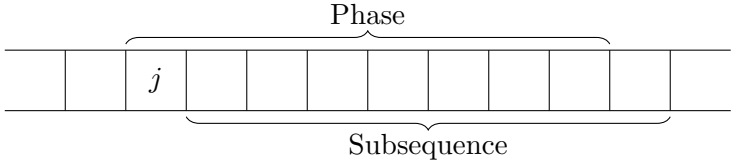
- If the requested page is in the cache, mark it
- Else (not in cache), if there is some unmarked page in the cache, evict some unmarked page and mark the new page
- Else (not in cache, no unmarked), unmark all pages, evict some unmarked page and mark the new page
- Initialize with all pages marked

Lemma 2.3. *1-bit LRU is k-competitive for any way of breaking the ties.*

Proof. Divide time into phases. Each time the algorithm unmarks all pages, a new phase starts. We will compare the cost incurred by the algorithm to the offline optimum in all phases separately.

In every phase, the algorithm incurs cost at most k. This is because any eviction coincides with a page being marked. In the entire phase, exactly k pages get marked.

We now show that the offline optimum has to incur cost 1 per phase except the last one. To this end, consider an arbitrary phase and the subsequence that is defined by removing the first request from this phase and adding the first request from the following phase. We claim that the offline optimum incurs cost at least 1. Let j be the page requested at the beginning of the phase, that is, right before the subsequence. We know that in the subsequence exactly k other pages are requests because the phase ends as the $k+1$-st distinct page is requested since its beginning. As j has to be in the cache when the subsequence starts, there has to be one eviction.



□

Observe that LRU simply applies one particular tie-breaking rule to "evict some unmarked page". It is therefore k-competitive.

2 Lower Bound for Any Deterministic Algorithm

Next, we will prove a lower bound on the competitive ratio of any deterministic algorithm. To this end, observe that Equation (1) has to hold for any sequence σ. If the algorithm is deterministic, then $\sigma_1, \ldots, \sigma_t$ fully determine the cache state after round t.

Therefore, the following perspective is equivalent to our model and helpful to give an impossibility result. We have a round-based game[1], in which the algorithm and an adversary take turns. The adversary always chooses the next request in the sequence, whereas the algorithm chooses an action how to deal with this. The respective choices may depend on what both players have done up to this point. The adversary's goal is to make the algorithm perform as poorly as possible with respective to the competitive ratio.

Theorem 2.4. *There is no deterministic algorithm for the paging problem that is c-competitive for $c < k$.*

Proof. We consider a universe of $k+1$ pages. Let $\mathcal{A}$ be an arbitrary deterministic algorithm and let the initial state of the cache be arbitrary. We now define the sequence σ step by step. Consider any point in time. There is always exactly one page that is not in the algorithm's cache. This is the page that the adversary will request. By this construction, the algorithm incurs a cost of 1 in every single round. Therefore, in a sequence of length T, $\text{cost}(\text{ALG}(\sigma)) = T$.

We now have to argue that one could do on any sequence σ that the adversary presents. Namely, we will show that $\text{cost}(\text{OPT}(\sigma)) \leq \lceil T/k \rceil$. Split the sequence into blocks of k rounds. We show that the offline optimum has cost at most 1 per block. Pretend that before the first block, there is a block in which exactly the pages that are initially in the cache at the start of the sequence.

[1] In game-theoretic terms, it is a zero-sum game.

Consider any block. In this block, the pages that are requested are the same as in the preceding block with at most one exception because there are only $k+1$ pages overall. $\mathrm{OPT}(\sigma)$ can make sure that at the end of a block, its cache contains exactly the pages requested in this block. If in the two blocks the same pages are requested, it can keep the cache as it is and will have no page fault. Otherwise, it replaces the one page that is not requested in this block by the one that is requested now but was not in the previous one.

It is now easy to verify that there is no $c < k$ for which there is a b such that $\mathrm{cost}(\mathrm{ALG}(\sigma)) \leq k \cdot \mathrm{cost}(\mathrm{OPT}(\sigma)) + b$ for all sequences σ. □

3 A Randomized Algorithm

We now consider a randomized variant of 1-bit LRU. We follow the above template. If it says "evict some unmarked page", then we drawn one unmarked page uniformly at random.

Theorem 2.5. *The randomized algorithm is H_k-competitive, where $H_k = \sum_{\ell=1}^{k} \frac{1}{k} = \Theta(\log k)$ is the k-th harmonic number.*

To analyze the algorithm, observe that the tie-breaking, that is, which unmarked page is evicted, does not affect how time divides into phases as described above. Also, the state of the cache at the end of a phase dos not depend on the tie-breaking—it contains exactly the k distinct pages requested in this phase. Therefore, both are independent of the randomization.

For phase i, let m_i be the number of pages that are requested in phase i but not in phase $i-1$. We set m_1 to be the number of pages that are requested in the first page but are not initially in the cache.

Lemma 2.6. *The cost of the offline optimum is at least $\frac{1}{2}\sum_i m_i$.*

Proof. Let OPT_i denote the cost incurred by the offline optimum in the i-th phase. We claim that $\sum_i \mathrm{OPT}_i \geq \frac{1}{2}\sum_i m_i$. It does not hold $\mathrm{OPT}_i \geq \frac{1}{2}m_i$, therefore we have to be a bit more clever here. We will use a *potential function*. Let Φ_i be the number of pages that are in the cache of the offline optimum but not in the memory of LRU at the end of phase i. Let $\Phi_0 = 0$ because the online algorithm and the offline optimum start with the same cache.

We have $\mathrm{OPT}_i \geq m_i - \Phi_{i-1}$. This is because LRU has exactly the pages requested in the previous phase in its cache. There are m_i missing that will be requested in this phase. The offline optimum can have at most Φ_{i-1} of them in its cache, so at least $m_i - \Phi_{i-1}$ are missing.

Furthermore, we have $\mathrm{OPT}_i \geq \Phi_i$. Note that at the end of the phase the algorithms's cache contains exactly the pages requested in this phase. Each of these pages was in the optimum's cache during this phase at some point but has potentially been evicted again. Φ_i is a lower bound on the number of these evictions.

So, overall

$$\sum_i \mathrm{OPT}_i \geq \sum_{i=1}^{p} \left(\frac{1}{2}(m_i - \Phi_{i-1}) + \frac{1}{2}\Phi_i\right)$$
$$= \frac{1}{2}\sum_{i=1}^{p} m_i - \frac{1}{2}\left(\sum_{i=0}^{p-1}\Phi_i + \sum_{i=1}^{p}\Phi_i\right) = \frac{1}{2}\sum_{i=1}^{p} m_i - \frac{1}{2}\left(\Phi_0 + \Phi_p\right) \geq \frac{1}{2}\sum_{i=1}^{p} m_i \ ,$$

because $\Phi_0 = 0$ and $\Phi_p \geq 0$. □

Lemma 2.7. *The algorithm has expected cost at most $\sum_i m_i H_k$.*

Proof. There are m_i pages requested in phase i that did not occur in the previous phase and consequently $k - m_i$ pages that also occurred in the previous phase. In the last phase, it might be even less than $k - m_i$ but this only helps us.

The cost incurred by the pages that did not occur in the previous phase is exactly m_i. For the others, let $X_1, \ldots, X_{k-m_i}$ be the random variables such that X_ℓ is the cost due to the ℓth page from the previous phase. The overall cost of phase i is therefore $C_i = m_i + X_1 + \ldots + X_{k-m_i}$.

We claim that $\mathbf{E}[X_\ell] \leq \frac{m_i}{k-\ell+1}$. To observe why this is true, let us first consider X_1. At the beginning of this phase the requested page was in the cache. However, it might have been evicted by the pages that were not requested in the previous phase. Up to this point, at most m_i new pages have been requested. They each evict one unmarked page from the cache. So overall, the probability is at most $\frac{m_i}{k}$ that our page has been evicted.

More generally, for X_ℓ, the probability that is evicted only increases if me mark the pages $1, \ldots, \ell - 1$. This means there are only unmarked $k - \ell + 1$ pages, which can be used to fit in the new pages. So, the probability is at most $\frac{m_i}{k-\ell+1}$ that the page has been evicted.

This means that for phase i, we get an expected cost of

$$\mathbf{E}[C_i] = \mathbf{E}[m_i + X_1 + \ldots + X_{k-m_i}] = m_i + \frac{m_i}{k} + \ldots + \frac{m_i}{k - m_i + 1} \leq \sum_{\ell=1}^{k} \frac{m_i}{\ell} = m_i H_k \ ,$$

and overall

$$\mathbf{E}\left[\sum_i C_i\right] = \sum_i \mathbf{E}[C_i] \leq \sum_i m_i H_k \ .$$

□

4 Discussion

In two lectures on competitive analysis, we have seen that it allows us to argue about why simple strategies are reasonable in the presence of uncertainty. We were able to derive rather simple, clean, and formal proofs. We used ideas of worst-case analysis. Namely, our guarantees hold without any further assumptions on the input. One should remark here that this worst-case perspective does not apply to the random bits used by the algorithm. This is because they algorithm can control them whereas it does not have control over the sequence.

However, we have also seen deficiencies of competitive analysis. Most importantly, the adversarial perspective is incredibly pessimistic. Deterministic algorithms only appear to be worse than randomized ones because it is harder to fool them. Practical instances usually are not designed to fool our algorithms.

References

- A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Chapters 3 and 4