CS 224: Advanced Algorithms Spring 2017

Lecture 10 — 23 February 2017

Prof. Jelani Nelson *Scribe: Alexander Wei*

1 Paging (cont.)

1.1 Randomized Paging

For randomized paging, recall the Mark algorithm [3], which is 1-bit LRU with the modification that every time we evict, we remove a *random* unmarked page from the cache.

Theorem 1. *The Mark algorithm is $2H_k$-competitive, where H_k is the k-th harmonic number.*

Proof. We break up the request sequence into phases based on when the Mark algorithm unmarks its entire cache, just as we did in our analysis of 1-bit LRU. We consider the element that caused the unmarking as the first element of the next phase. Call a request for page p **clean** if p has not yet been requested in the current phase and p was not requested in the last phase either. Call a request **stale** if p has not yet been requested in the current phase and p was requested in the last phase. We can think of the clean requests as definite cache misses—there is not much we can do to improve them. For stale requests, however, we can hope that randomization improves performance because the cache is filled with stale pages at the beginning of each phase.

Let L_i be the number of clean requests in phase i, and let D_i be the number of items in the cache of the optimal algorithm (OPT) at the beginning of phase i that are not in the cache of the Mark algorithm. As usual, we assume the two algorithms start off with the same initial cache; i.e. $D_1 = 0$.

We prove the following claims:

1. In phase i, OPT pays at least $L_i - D_i$.
2. In phase i, OPT pays at least D_{i+1}.

These imply OPT pays at least the average of these two, which is $\frac{1}{2}(L_i + D_{i+1} - D_i)$. Thus, OPT pays at least

$$\sum_i \frac{1}{2}(L_i + D_{i+1} - D_i) = \frac{1}{2}\sum_i L_i + \frac{1}{2}D_{\text{final}} + \frac{1}{2}D_1 \geq \frac{1}{2}\sum_i L_i$$

overall, where first equality follows from a telescoping sum. We additionally show

3. In phase i, the expected cost of the Mark algorithm is at most $H_k \cdot L_i$.

The theorem follows immediately from the above claims, since

$$\frac{\mathbb{E}[C_{\text{Mark}}]}{C_{\text{OPT}}} \leq \frac{H_k \sum_i L_i}{\frac{1}{2}\sum_i L_i} = 2H_k.$$

We start by proving the first claim. Observe that each of the L_i clean requests causes a page fault for the Mark algorithm. Since OPT has at most D_i clean request pages cached, OPT has to pay for at least the remaining $L_i - D_i$ clean pages in the i-th phase. This gives us the first claim.

For the second claim, observe that by the end of the i-th phase, at least k distinct pages have been brought into cache in the i-th phase. (These are exactly the marked pages in the Mark algorithm.) By the definition of D_{i+1}, OPT has brought at least D_{i+1} other pages into its cache. Thus, during the i-th phase, OPT has had at least $k + D_{i+1}$ distinct pages in its cache. Because OPT only has a cache of size k, OPT had to have at least D_{i+1} page faults. Thus we have the second claim.

We now move on to the third claim. In the Mark algorithm, observe that only clean and stale requests can result in page faults. Each clean request causes a page fault, so the Mark algorithm incurs a cost of L_i from those. To analyze stale requests, we focus on a particular stale request: Let c and s be the number of clean requests so far in the current phase, and let s be the number of stale requests so far in the current phase. There are s marked elements in the cache from the stale requests. Furthermore, there have been c clean requests that have clobbered away some of our stale pages. Because the cache has k pages in total, the probability that the current request results in a page fault is thus $\frac{c}{k-s}$.

This probability lets us bound the total expected cost from stale requests. The number of stale requests in a phase is at most $k - L_i$. We thus have

$$\begin{aligned}
\mathbb{E}[\#\text{ of page faults from stale requests}] &\leq \sum_{s=0}^{k-L_i-1} \frac{L_i}{k-s} \\
&\leq L_i \cdot \sum_{s=L_i+1}^{k} \frac{1}{s} \\
&= L_i(H_k - H_{L_i}) \\
&\leq L_i(H_k - 1).
\end{aligned}$$

Combining our values for clean and stale requests, we obtain

$$\mathbb{E}[\#\text{ of page faults}] \leq L_i + L_i(H_k - 1) = L_i \cdot H_k,$$

which is the statement of the third claim. □

1.2 Recent Results

In addition to the paging model that we have discussed, we can also consider the **weighted paging problem**, where each page p has a weight w_p which is the cost of bringing it into cache. This problem models the real life scenario in which different pages are stored in different locations, resulting in varying costs for fetching.

Theorem 2 (Bansal, Buchbinder, Naor [1])**.** *There exists an* $O\left(\log\left(\frac{k}{k-h+1}\right)\right)$*-competitive randomized algorithm for weighted paging with resource augmentation.*

Bansal et al. designed this algorithm with the primal/dual framework that we are about to see. A possible open problem for the final project is to optimize the constant inside the big-O, since their paper only focused on obtaining the big-O and may not have particularly optimized constants.

Another related problem is the k-**server problem**, which is even more general. In this problem, k servers live in an n-point metric space, and each request is for a point in the metric space. To service a request, a server must move to the point being queried. Moving a server from point p to point q costs $d(p, q)$.

To see that the k-servers problem is a generalization of the weighted paging problem, consider the scenario in which $d(p, q) = \frac{1}{2}(w_p + w_q)$ (where each page corresponds to a point). We have each server represent a space in our cache. Initially, the servers start off at a null page that has weight 0. Then, as servers move between pages, each pays half the cost as it enters a page and half the cost as it leaves. We also have to add a final dummy request for the null page to return all the servers to their starting points. (But this doesn't quite work!!)

Theorem 3 (Koutsoupias, Papdimitriou [4])**.** *There exists a deterministic* $(2k-1)$*-competitive algorithm for the* k*-servers problem. In addition, there does not exist a* k*-competitive deterministic algorithm.*

Note also that for randomized approaches to the k-servers problem, we cannot hope for an algorithm that is $o(\log k)$-competitive, because paging is a special case of this problem. An open problem is whether there exists a polylog(k)-competitive for the k-servers problem. The progress so far (from Bansal, Buchbinder, Madry, Naor [2]) is a $O(\log^2 k \log^3 n \log\log n)$-competitive algorithm. A weakness of this bound is its dependence on n—the number of points in the space.

2 Online Primal/Dual

Online primal/dual is a framework for developing online algorithms with good competitive ratio via linear programs. On Tuesday, we'll see an example for online ski rental that achieves the competitive ratio we saw earlier as well as a randomized version that has a competitive ratio of $e/(e-1)$. Next lecture, we'll also develop an algorithm for online set cover using this framework.

2.1 Linear Programming Review

A **linear program** (LP) consists of an objective $\min c^\top x$ and a set of constraints

$$\begin{aligned} \sum a_{1i}x_i &\geq b_1 \\ \sum a_{2i}x_i &\geq b_2 \\ &\vdots \\ \sum a_{ni}x_i &\geq b_n \\ x &\geq \mathbf{0}. \end{aligned}$$

In other words, we want to minimize $c^\top x$ such that $Ax \geq b$ and $x \geq \mathbf{0}$. It is not difficult to show that more general linear programs (e.g. with a maximization objective, $\leq$'s in the constraints, or negative x_i), can be converted into this form. We call a linear program of this form **primal**. If every entry of A, b, and c is non-negative, then the linear program is said to be **covering**. For a linear program P, let OPT(P) denote the optimal value of the objective.

2.2 LP Duality

Given a linear program $\min c^\top x$ subject to $Ax \geq b$ and $x \geq \mathbf{0}$, suppose we want to prove a lower bound $c^\top x$. One way to do this is to play around with the constraints: We can consider multiplying the i-th constraint by y_i, where $y \geq \mathbf{0}$. Summing the results gives us $b^\top y \leq y^\top Ax = (A^\top y)^\top x$. Then, if $A^\top y \leq c$, we have that $b^\top y \leq c^\top x$. Note that the greater $b^\top y$ is, the stronger our bound.

In this process, we have created another linear program, namely $\max b^\top y$ subject to the constraint $A^\top y \leq c$. This linear program is known as the **dual** of our primal LP $\min c^\top x$ such that $Ax \geq b$. If $A, b, c \geq 0$, then such a dual LP is a **packing** LP. That is, the dual of a covering LP is a packing LP. The definition of the dual immediately gives us the following theorem:

Theorem 4 (Weak Duality Theorem)**.** *Suppose P is a primal LP and D is its dual. Then*

$$\mathrm{OPT}(P) \geq \mathrm{OPT}(D).$$

There is also another version this theorem that we'll prove (and use) later in the course:

Theorem 5 (Strong Duality Theorem)**.** *If the primal P is bounded and feasible and D is its dual, then*

$$\mathrm{OPT}(P) = \mathrm{OPT}(D).$$

Let's define the two terms in the above theorem statement. The **feasible region** of a linear program is the subset of $\mathbb{R}^n$ consisting of the x that satisfy all the constraints. A linear program is **feasible** if exists at least one $x \in \mathbb{R}^n$ satisfying the constraints and is **bounded** if its feasible region is bounded (i.e. the feasible region is an n-dimensional polytope). There are also results that say if the primal is unbounded, then the dual is unfeasible, etc.

2.3 Online Linear Programs

Consider an online linear program, where x starts off as $\mathbf{0} \in \mathbb{R}^n$ and the constraints on x come in online. (By constraint, we mean a single row of A.) Furthermore, assume the linear program defined always remains a covering LP. When a constraint comes in, we want to adjust x so that the new constraint is satisfied. We are allowed to adjust x by increasing any subset of its coordinates. (In particular, we cannot decrease any coordinates of x.)

One approach is to maintain primal/dual feasible solutions x and y such that $c^\top x \leq \gamma \cdot b^\top y$. We know by weak duality that

$$c^\top x \leq \gamma \cdot b^\top y \leq \gamma \cdot \mathrm{OPT}(D) \leq \gamma \cdot \mathrm{OPT}(P),$$

where P is the primal LP and D is its dual. Therefore, if we maintain such feasible x and y, then we have a γ-competitive solution for the primal.

Theorem 6 (Approximate complementary slackness)**.** *Suppose we have a primal LP $\min c^\top x$ subject to $Ax \geq b$ for which x and y are primal and dual feasible solutions, respectively. If there exist $\alpha, \beta \geq 1$ such that*

1. *$x_i > 0$ implies $c_i/\alpha \leq (A^\top)_i y \leq c_i$*

2. $y_i > 0$ *implies* $b_i \leq A_i x \leq \beta \cdot b_i$,

then $c^\top x \leq \gamma \cdot b^\top y$ *for* $\gamma = \alpha\beta$.

Proof. We'll prove this on the next problem set. □

By this theorem, if we can maintain x and y online so that they satisfy the two properties above, then we have a γ-competitive algorithm for the linear program. Next week, we'll see how this can be applied to obtain competitive algorithms for other online problems.

References

[1] Nikhil Bansal, Niv Bucbinder, Joseph (Seffi) Naor. A Simple Analysis for Randomized Online Weighted Paging. *FOCS '07*.

[2] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, Joseph Naor. A Polylogarithmic-Competitive Algorithm for the k-Server Problem. *J. ACM*, 52(5): 40:1-40:49, 2015.

[3] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, Neal E. Young. Competitive Paging Algorithms. *J. Alg.*, 12:685–699, 1991.

[4] Elias Koutsoupias, Christos Papdimitriou. On the k-Server Conjecture. *J. ACM*, 42(5):971–983, 1995.