6.854 Advanced Algorithms	
Lecture 20: 11/01/2004	Lecturer: David Karger
Scribes: Tushara C. Karunaratna	

Online Algorithms (continued)

20.1 Introduction

In the previous lecture, we defined the term *competitive ratio* for deterministic online algorithms. We studied a 2-competitive algorithm for ski-rental. We also observed that our greedy algorithm for load balancing is a 2-competitive online algorithm.

In this lecture, we will study online algorithms for paging, finance, and k-server on a line.

20.2 Paging

Paging is an important problem in computer systems design. We model a machine's memory as consisting of two parts: an unlimited number of pages of *slow* memory, and a cache consisting of k pages of *fast* memory. On a page request, if the requested page is not in the cache (we call this a *cache miss* or a *fault*), then a page in the cache must be *evicted* and the requested page must be brought in. A *paging strategy* specifies the choice of which page to evict on a cache miss. The goal is to minimize the number of cache misses.

Here are the some of the commonly used paging strategies.

- LRU: evict the least recently used page. We will prove that this strategy is k-competitive.
- Random: evict a random page. It is known that this strategy is k-competitive.
- FIFO: evict the earliest fetched page. It is known that this strategy is k-competitive.
- Frequency counts: evict the least frequently used page. It is known that this strategy is *not* competitive.

20.2.1 LRU is k-competitive

To analyse the performance of LRU, we partition the input sequence into *phases*. The first phase begins immediately after LRU first faults. A phase ends immediately after LRU has faulted k times since the start of the phase; the next phase begins at this point.

We now prove that OPT faults at least once per phase.
Consider any phase such that LRU faults *twice* on some page p in this phase. We know that at least k other distinct pages must have been requested in between the two requests of p (because otherwise p would not have been evicted by LRU). Hence, there are at least $k+1$ distinct pages requested in this phase, and thus OPT faults at least once in this phase.
On the other hand, consider any phase such that LRU faults on k *distinct* pages in this phase. Let p be the last fault of the previous phase. Note that even if p is one of the k faults in this phase, at least k other distinct pages must have been requested in this phase (because otherwise p would not have been evicted by LRU). Since p was in OPT's cache at the start of this phase, OPT faults at least once in this phase.

Therefore, LRU is k-competitive.

20.2.2 Lower bound for deterministic online paging

Consider any deterministic online paging algorithm A.

We construct a request sequence involving $k+1$ pages; at each step, we request the page that is *not* in A's cache. Note that for this sequence, A faults on every request.

Consider the offline algorithm that, on a cache miss, evicts the page that is requested furthest in the future. When this offline algorithm faults, the page that it evicted won't be requested until at least k steps in the future. Therefore, this offline algorithm faults only once every k requests for our above adverserial sequence.

Therefore, no deterministic online paging algorithm can be better than k-competitive.

20.3 Finance

We own some amount of stock. We have no information on how the price per unit of the stock may fluctate in the future, but we know that the peak price is M and that the minimum price is m. We want to sell our stock so as to maximize our return.

20.3.1 Reserve price algorithm

The stategy is to sell all our stock the first time that the price beats r.
We claim that $r = \sqrt{Mm}$ is a good choice:

- Case 1: We never meet the reserve price. Then, our regret ratio is at most r/m.
- Case 2: We meet the reserve, and it eventually rises to M. Then, our regret ratio is at most M/r.

$\max\{r/m,\ M/r\}$ is minimized by setting $r/m = M/r$. This gives us $r = \sqrt{Mm}$. With this choice of r, the regret ratio is $\sqrt{M/m}$.

20.3.2 Selling in fractions

We assume that $M = 2^k m$ for some integer k.
Let OPT denote the maximum price that is reached (of course, we do not know this in advance).

Consider the following strategy: for each i where $1 \leq i \leq k$, sell $1/k$ of our stock the first time that the unit price falls in the interval $[2^i m, 2^{i+1} m)$.

Let j be the largest integer such that $2^j m \leq$OPT.
Note that $2^j m > OPT/2$.
Therefore, we sell $1/k$ of our stock at price at least $OPT/2$.
So our regret ratio is $\frac{OPT}{OPT/(2k)} = 2k = 2 \lg (M/m)$.

20.3.3 A randomized reserve-price strategy

We pick i uniformly at random from $1, 2, \ldots, k$. We set the reserve price to $2^i m$.

Let j be the largest integer such that $2^j m \leq$OPT.
With probability at least $1/k$, we have a return of $2^j m > OPT/2$.
Therefore, our expected return is at least $OPT/2k$.
So our expected regret ratio is $2k = 2 \lg (M/m)$.

20.4 k-server on a line

We have k servers on a line. A *request* specifies a point on the line to which a server must be moved. The cost of serving a request sequence is measured by the total distance travelled by servers. Thus, the goal is to be clever in the way servers are moved, such that the total distance travelled by the servers is minimized.

20.4.1 A greedy strategy that isn't competitive

A *greedy* strategy for this problem is to simply move the server that is closest to the requested point.

However, we can easily see that this greedy strategy is not competitive. Suppose that we have two servers A and B that are initially at points x and y, respectively, with $x < y$ Suppose that our requests repeatedly alternate between the points $y - (y - x)/4$ and $y + (y - x)/4$. Then, according to the greedy strategy, all the requests will be served by B, and the total distance travelled will be infinite. A better strategy would be to first move A and B to our two request points, in which case the total distance travelled would be only constant.

20.4.2 Double coverage

The *double coverage*(DC) strategy is

- If the request falls between two servers, then move both towards the request at the same rate until one reaches it.
- Else (in which case the request falls "outside"), just move the closest server to the request.

Now, we show that DC is k-competitive.

We compare the moves of DC to those of an optimal offline algorithm OPT. We can assume that OPT satisfies a request by moving *exactly* one server; we can convert any offline strategy into a strategy that moves *exactly* one server per request, without increasing the cost, by differing moves to the future.

Let M be the cost of the minimum cost matching between DC's and OPT's servers.
Let S be the sum of the pairwise distances of DC's servers.
Let $\Phi = kM + S$. This is our potential function.

We consider the requests one by one. For each request, we first observe OPT's move, and then we observe DC's move. We compute the ammortized cost incurred by DC for each move.

- **OPT moves**distance d. The distance from the moved server to the moved server's match increases by at most d. Hence, M increases by at most d. Of course, S is unchanged.
  So $\Delta\Phi \leq kd$.
  The real cost incurred by DC is 0.
  So the ammortized cost incurred by DC is $\leq kd$.
- **DC moves** There are two cases to consider.
  - **The request falls between two servers** Suppose that each of the two moved servers moved a distance d. The pairwise distance between the two moved servers decreases by $2d$. The changes in the other pairwise distances cancel out. So S decreases by $2d$.
    One of the moving servers has its match on its destination. The other moving server might be moving further from its match, but only at the same rate. Therefore, M does not increase.
    So $\Delta\Phi \leq -2d$.
    The real cost incurred by DC is $2d$.
    So the ammortized cost incurred by DC is ≤ 0.
  - **The request falls "outside"** Suppose that the moved server moved a distance d. The moved server has its match on its destination. So M decreases by at least d.
    Each of the $k-1$ pairwise distances involving the moved server increases by d. So S increases by $(k-1)d$.
    So $\Delta\Phi \leq -kd + (k-1)d = -d$.
    The real cost incurred by DC is d.
    So the ammortized cost incurred by DC is ≤ 0.

Note that positive ammortized cost is incurred by DC only when OPT moves. When OPT moves, the ammortized cost incurred by DC at most k times the distance moved by OPT's server.

Thus, the total real cost incurred by DC is at most k times that incurred by OPT (plus some additional constant if the potential wasn't initially 0).

So, DC is k-competitive.