Advanced Algorithms

Lecture 11

Lecture by Shay Mozes
Typeset by Steven Karas

2017-01-22
Last edited 20:58:56 2017-01-22

Note The video from last week was not uploaded because Shay was at a conference that discussed algorithms, where almost all of the topics we discussed in the course were presented or discussed.

1 Online Algorithms

Online algorithms are a class of algorithms that operate with perfect knowledge of the past, but imperfect knowledge of the future. Offline algorithms, in contrast, can operate on full knowledge of the input.

For this type of problem, we will design algorithms that are competitive with the optimal offline algorithm, which has perfect knowledge.

1.1 Example: Ski Rental Problem

Assume that you are taking ski lessons. It takes 1$ to rent ski equipment for a day, and $y$$ to purchase the equipment outright.

Offline Algorithm If you knew in advance how much you'd ever go skiing in your life, then the decision is trivial: if you will ski more than y times, you should buy. Otherwise, you should always rent. The cost of this algorithm is $min(t, y)$, where t is the number of times you go skiing.

Online An online strategy is a number k such that after renting $k - 1$ times, you will purchase skis (just before the k^{th} visit).

Setting $k = y$ guarantees that we will never pay more than twice the cost of the offline strategy.

Competition If $t < y$, then you pay only t, which is optimal. Otherwise, $t \geq y$.

Your total payment is $y-1+y = 2y-1$. The offline optimum is $\min(t, y) = y$. The ratio is: $\frac{2y-1}{y} = 2 - \frac{1}{y}$.

Lower bound on competition Let k be any strategy (buy after $k-1$ rents). Suppose you buy the skis at the k^{th} time and then break your leg and never ski again (i.e., $t = k$). Your total ski cost is $k - 1 + y$ and the optimum offline cost is $\min(k, y)$. For every k, the ratio $\frac{(k-1+y)}{\min(k,y)}$ is at least $2 - \frac{1}{y}$. Therefore, every strategy is at least $(2 - \frac{1}{y})$-competitive.

1.2 Online minimization problems

For a minimization problem, any online algorithm A is k-competitive if for any input, $C_A \leq k \cdot C_{OPT} + b$ for some constant factor b.

1.3 Example: Hole in the Fence Problem

A cow wants to escape from the farm. It knows there is a hole somewhere in the fence, but doesn't know which direction it is.

Exponential walk

```
d=1; direction=right
repeat:
  walk d steps in the current direction
  if we found the hole, exit
  else, return to the origin
  d = 2d
  flip the current direction
```

Competition We claim this algorithm is 9-competitive. The worst case is that it finds the hole a little bit beyond the distance it last searched on this side[1]. Thus, $OPT = 2^j + \varepsilon$ where j = number of iterations, and ε is some small distance.

$$OPT = 2^j + \varepsilon > 2^j$$

$$COW = 2(1 + 2 + ... + 2^{j+1}) + 2^j + \varepsilon \leq 2 \cdot 2^{j+2} + 2^j + \varepsilon = 9 \cdot 2^j + \varepsilon < 9 \cdot OPT$$

1.4 Example: Edge coloring

An edge coloring of a graph $G = (V, E)$ is an assignment c of integers to the edges such that for any edges e_1, e_2 that share an endpoint, $c(e_1) \neq c(e_2)$. Let Δ be the maximal degree of some vertex in G. In the offline case, it is possible to edge-color G using Δ or $\Delta + 1$ colors.

Online The graph is not known in advance. In each step a new edge is added and we need to color it before the next edge is known.

Let $e = (u, v)$ be a new edge. Color e with the smallest color which is not being used by any edge adjacent to u or v.

[1]Note that we would need to prove this is the worst case for this proof to be rigorous.

Competition We claim this algorithm uses at most $2\Delta - 1$ colors. Assume we need the color 2Δ. It must be that all the colors $1, 2, ..., 2\Delta - 1$ are used by edges adjacent to either u or v. Therefore, either u or v has Δ edges excluding e, which contradicts our assumption. Therefore, we use at most $2\Delta - 1$ colors.

Lower Bound We claim that any deterministic algorithm needs at least $2\Delta - 1$ colors. Assume there exists an algorithm that uses only $2\Delta - 2$ colors. Given Δ we add to the graph many $\Delta - 1$-stars.

There are a finite number of ways to edge-color a $\Delta - 1$-star with colors $1, 2, ..., 2\Delta - 1$, so at some point we must have Δ stars all colored with the same set of $\Delta - 1$ colors. Now we add a new vertex a and connect it to the center of each of the stars. These edges must all have a unique color that is not one of the $\Delta - 1$ colors used to the color the stars. Therefore, it must use $2\Delta - 1$ colors.

1.5 Example: Load Balancing

Given: we have a set of m identical machines. A sequence of jobs with processing times $p_1, p_2, ...$. Each job must be assigned to a machine. When job j is scheduled, we don't know how many additional jobs we are going to have or what their processing times will be.

Our goal is to schedule the jobs on the machines in a way that minimizes the *makespan* $= \max_i \sum_{j \text{ on } m_i} p_j$, which is the maximal load on a single machine.

Offline Optimum We compare the last completion time in the schedule of the online algorithm to the last completion time in the schedule of the optimal offline algorithm.

Note that this particular problem is one of the most interesting problems, with applications in †ask scheduling for operating systems, data storage on disks, shift scheduling, factory production, etc. In almost all these situations, online algorithms are far more realistic than an offline algo.

1.5.1 List scheduling

Graham 1966. A greedy algorithm that schedules a job on the least loaded machine[2].

Competition List-Scheduling is $2 - \frac{1}{m}$ competitive.

Let j be the last job to finish. Let t be the load of the machine to which j is assigned, just before j was assigned. The total load of all jobs prior to t is $\sum_{i<j} p_i$, so $t \leq \frac{1}{m} \sum_{i<j} p_i$ because j is assigned to the least loaded machine.

Note that $OPT \geq \frac{1}{m} \sum p_i$, and $OPT \geq p_j$.

$$t + p_j \leq \frac{1}{m} \sum_{i<j} p_i + p_j \leq \frac{1}{m} \sum_{i \leq j} p_i + \left(1 - \frac{1}{m}\right) p_j \leq \left(2 - \frac{1}{m}\right) OPT$$

[2] An example run of this is on slide 21

Lower bound for Online Scheduling For $m = 2$, there is no algorithm such that $r < 1.5$.

Consider the sequence $1, 1, 2$. If the first two jobs are scheduled on different machines, the third job completes at time 3. If the first two jobs are scheduled on the same machine, the adversary stops. In the first case, $r = 1.5$. In the second case, $r = 2$.

1.6 Example: Paging - Cache Eviction

Given two levels of memory: fast memory M_1 of k pages (cache), and slow memory M_2 of n pages ($k < n$).

Pages in M_1 are a strict subset of the pages in M_2. Pages are only accessible through M_1. Accessing a page in M_1 has cost 0. When accessing a page not in M_1, it must first be brought in from M_2 with a cost of 1 before it can be accessed. This is called a page fault. If M_1 is full when a page fault occurs, a page currently in M_1 must be evicted to make room.

1.6.1 Offline optimum: Longest-Forward-Distance

On each page fault, evict the page in M_1 that will be requested farthest in the future. An example run of LFD can be found on slide 28. LFD was proven to be offline-optimal in 1966.

LFD Optimality For any other algorithm A, the cost of A is not increased if in the 1st time that A differs from LFD, we evict in A the page that is requested furthest in the future.

1.6.2 FIFO

Evict the page that has been in the cache the longest. An example can be found on slide 30. FIFO is k-competitive: for any sequence, $FIFO \leq k \cdot LFD$. A full proof of this can be found on slide 31.

1.6.3 LIFO

Evict the last page that was promoted to the cache. An example can be found on slide 32. For any $n > k$, LIFO is not competitive: for any c, there exists a sequence of requests such that $LIFO \geq c \cdot LFD$. Proof: consider $\sigma = 1, 2, ..., k, k + 1, k, k + 1, ...$

1.6.4 LRU

Evict the least recently used page from the cache. An example can be found on slide 33. LRU is k-competitive. Proof is almost identical to FIFO, although in practice it is much better.

Lower Bound for deterministic online We want to prove that for any k and any deterministic online algorithm A, the competitive ratio of $A \geq k$. Assume that $n = k + 1$. Always request the page that is not in M_1. This causes a page fault on every access. What is the price of LFD in this sequence? At most a single page fault in any k accesses, because LFD evicts the page that will be needed in the $k+1$th request or later. The total cost of LFD is at most $\frac{|\sigma|}{k}$. Therefore, worst-case analysis is not so important in analyzing paging algorithms.

Randomization however can help. There is a randomized $2H_k$-competitive algorithm[3].

1.6.5 MARKING

The algorithm is described on slides 36-40.

We did not go over the proof in class, due to time constraints.

1.7 Example: Bin Packing

Given a sequence of items $a_1, ..., a_n$, such that for all i, $0 < a_i < 1$. The goal is to "pack" the items in bins of size 1 using as few bins as possible.

1.7.1 Next-Fit

Exactly as we described in prior lectures, it is 2-competitive.

1.7.2 First-Fit

Exactly as we described in prior lectures, it is 1.7-competitive.

1.7.3 HARMONIC-k

Classify items into k intervals according to size.

Examples can be found on slide 43. Analysis and proofs can be found on later slides.

[3] $H_k = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{k} = O(\ln k)$