

No final.

Mention online as candidate for implementation project.

Last time, discussed online algorithms. Saw FIFO/LRU  $k$ -competitive, optimal.

## Randomized Online Algorithms

As discussed last time, an online algorithm is a two-player zero sum game between algorithm and adversary. Well known that optimal strategies require randomization.

A *randomized online algorithm* is a probability distribution over deterministic online algorithms.

- idea: if adversary doesn't know what you are doing, can't mess you up.
- idea: can't see adversary's "traps", but have certain probability of wiggling out of them.
- in practice, don't randomly pick 1 det algorithm at start. Instead, make random choices on the way. But retrospectively, gives 1 deterministic algorithm.

Algorithm is  $k$ -competitive if for any  $\sigma$ ,  $E[C_A(\sigma)] \leq k \cdot OPT + O(1)$ .

Adversaries:

- **oblivious:** knows probability distribution but not coin tosses. Might as well pick input in advance.
- **fully adaptive:** knows all coin tosses. So algorithm is deterministic for it.
- **adaptive:** knows coin tosses up to present—picks sequence based on what did.
- clearly adaptive stronger than oblivious.
- oblivious adversary plausible in many cases (eg paging)
- problematic if online behavior affects nature (eg, paging an alg that changes behavior if it sees itself thrashing)
- for now, oblivious

Marking algorithm:

- initially, all pages marked (technicality)
- on fault, if all marked, unmark all
- evict random unmarked page
- mark new page.

Fiat proved: Marking is  $O(\log k)$  competitive for  $k$  pages.

Phases:

- first starts on first fault
- ends when get  $k + 1^{st}$  distinct page request.
- so a phase has  $k$  distinct pages
- cost of  $M$  is cost of phases
- note: defined by input, independent of coin tosses by  $M$
- but, marking tracks:
  - by induction, unmark iff at end of phase
  - by induction, all pages requested in phase stay marked till end of phase
  - so, pay for page (if at all) only on first request in phase.
  - by induction, at end of phase memory contains the  $k$  pages requested during the phase.

Analysis:

- ignore all but first request to a page (doesn't affect  $M$ , helps offline)
- compare phase-by-phase cost
- phase  $i$  starts with  $S_i$  (ends with  $S_{i+1}$ )
- request *clean* if no in  $S_i$ .  $M$  must fault, but show offline pays too
- request *stale* if in  $S_i$ .  $M$  faults if evicted during phase. Show unlikely.

Online cost:

- Expected cost of stale request:
  - suppose had  $s$  stale and  $c$  clean requests so far.
  - so  $s$  pages of  $S_i$  known to be currently in memory
  - remaining  $k - s$  may or may not be.
  - in particular,  $c$  of them got evicted for clean requests
  - what prob current request was evicted?  $c/(k - s)$
  - this is expected cost of stale request.
- Cost of phase.
  - Suppose has  $c_i$  clean requests,  $k - c_i$  stale.
  - Pay  $c_i$  for clean.

– for stale requests, pay at most

$$c_i \left( \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{c_i+1} \right) = c_i (H_k - H_{c_i})$$

– so total at most  $c_i \log k$

Offline cost:

- potential function  $\Phi_i$  = difference between  $M$  on  $O$  (offline) at start of phase  $i$ .
- got  $c_i$  clean requests, not in  $M$ 's memory. So at least  $c_i - \Phi_i$  not in  $O$ 's memory.
- at end of round,  $M$  has all  $k$  most recent requests. So  $O$  is missing  $\Phi_{i+1}$  of  $k$  this round's requests. Must have evicted (thus paid for) them.
- so,  $C_i(O) \geq \max(c_i - \Phi_i, \phi_{i+2}) \geq \frac{1}{2}(c_i + \Phi_i - \Phi_{i+1})$ .
- sum over all phases; telescopes. Deduce  $C_i \geq \frac{1}{2} \sum c_i$ .

Summary: If online pays  $x \log k$ , offline pays  $x/2$ . So,  $(\log k)$ -competitive.

## Lower bounds

Turns out that  $O(\log k)$  is tight for randomized algorithms (Fiat). How prove? Recall that situation is a game:

- in general, optimal strategy of both sides is randomized
- online chooses random alg, adversary chooses random input
- leads to payoff matrix—expected value of game
- number in matrix is cost for alg on that input
- Von Neumann proved equality of minimax and maximins
- notice: player who picks strategy second can use deterministic choice
- note when one player's strategy known, other player can play deterministically to meet optimum.
- above, assumed adversary knew online's strategy, so he played deterministically
- for lower bound, we let adversary have randomized strategy, look for best deterministic counter!
- If give random input for which no deterministic alg does well, we get a lower bound.

Formalize:

- say online  $A$  is  $k$ -competitive against an input distribution  $p_\sigma$  if  $E_\sigma(C_A(\sigma)) \leq kE_\sigma(C_{OPT}(\sigma))$  (note: OPT gets to see sequence before going)
- Theorem: if for some distribution no deterministic alg is  $k$ -competitive, then no randomized algorithm is  $k$ -competitive.
- to prove, suppose have  $k$ -competitive randomized alg, show det  $k$ -competitive against any  $\sigma$ .
- consider payoff  $E_A[C_A(\sigma) - kC_{OPT}(\sigma)]$
- by assumption, some dist on  $A$  achieves nonpositive payoff.
- remains true even if choose best possible randomized strategy on  $\sigma$
- once do so, have deterministic counter  $A$
- so for any  $p_\sigma$  on  $\sigma$ , some  $A$  such  $E_\sigma[C_A(\sigma) - kC_{OPT}(\sigma)] \leq 0$
- in other words,  $A$  is  $k$ -competitive against  $p_\sigma$ .

For paging:

- set of  $k + 1$  pages
- uniform random sequence of requests
- *any* deterministic (or randomized!) algorithm has an expected  $1/k$  fault per request. So cost  $n/k$  if seq length  $n$
- what is offline cost? on fault, look ahead to page that is farthest in future.
- *phase* ends when all  $k + 1$  pages requested
- offline faults once per phase
- how long is a phase? coupon collection.  $\Omega(k \log k)$ .
- intuitively, number of faults is  $n/k \log k$
- formally, use “renewal theory” that works because phase lengths are i.i.d.
- deduce expected faults  $n/k \log k$ , while online is  $n/k$
- $\log k$  gap, so online not  $\log k$ -competitive.

## ***k*-server**

Definition:

- metric space with  $k$  servers on points
- request is point in space
- must move a server, cost is distance.
- eg taxi company
- paging is special case: all distances 1, servers on “memory pages”
- also multihead disks
- compute offline by dynamic program or reduction to min cost flow

Greedy doesn't work:

- 2 servers, 1 far away, other flips between 2 points.
- need an algorithm that moves a far away server sometimes in case a certain region is popular

Fancy algorithmics:

- HARMONIC: randomized, move with probability inversely proportional to distance from goal
- WORK FUNCTION: track what offline algorithms would have done (computationally very expensive) and then do your best to move into a similar configuration.
- last year, work-function was proven  $2k$ -competitive using a black magic potential function
- conjectured  $k$ -competitive.
- questions remain on finding an algorithm that does little work per input.

## **Graham's Rule**

Define  $P || \max C_j$  to minimize load.

NP-complete to solve exactly!

Always assign to least loaded machine:

- any alg has 2 lower bounds: average load and maximum job size.
- Suppose  $M_1$  has max load  $L$ , let  $p_j$  be biggest job.
- claim every machine has  $L - p_j$  (else wouldn't have assigned last job to  $M_1$ )

- thus total load at least  $\sum p_i = m(L - p_j) + p_j$
- thus  $\text{OPT} \geq L - p_j + p_j/m$
- but  $\text{OPT} \geq p_j$ , so  $(2 - 1/m)\text{OPT} \geq L$

More recent algs do somewhat better:

- keep some machines small
- algorithms not too bad, proofs awful!

Section\*Approximation Algorithms

What do you do when a problem is NP-complete?

- run a nonpolynomial (hopefully on slightly) algorithms such as branch and bound. Usually no proven bound on runtime.
- assume input is random, do “expected performance.” Eg, Hamiltonian path in all graphs. Problem: agreeing on good distribution.
- settle for a *heuristic*, but prove it does *well enough*

Definitions:

- optimization problem, instances  $I$ , solutions  $S(I)$  with values  $f : S(I) \rightarrow R$
- maximization/minimization: find solution in  $S(I)$  maximizing/minimizing  $f$
- called  $\text{OPT}(I)$
- eg bin-packing. instance is set of  $s_i \in 0, 1$ , partition so no subset exceeds 1

Technical assumptions we’ll often make:

- assumption: all inputs and range of  $f$  are integers/rationals (can’t represent reals, and allows, eg, LP).
- assumption  $f(\sigma)$  is a polynomial size (num bits) number
- look for polytime in bit complexity

NP-hardness

- optimization NP-hard if can reduce an NP-hard decision problem to it
- (eg, problem of “is opt solution to this instances  $\leq k$ ?”
- but use more general notion of turing-reducibility (GJ).

Approximation algorithm:

- any algorithm that gives a feasible answer
- eg, each item in own bin.
- of course, want *good* algorithm. How measure?

## Absolute Approximations

Definition:  $k$ -abs approx if on any  $I$ , have  $|A(I) - OPT(I)| \leq k$

Example: planar graph coloring.

- NP-complete to decide if 3 colorable
- know 4-colorable
- easy to 5-color

Known only for trivial cases, where opt is bounded by a constant.

Often, can show impossible.

EG knapsack.

- define profits  $p_i$ , sizes  $s_i$ , sack  $B$
- wlog, integers.
- suppose have  $k$ -absolute
- multiply all  $p_i$  by  $k + 1$ , solve, scale down.

EG independent set (clique)

- $k + 1$  copies of  $G$

## Relative Approximation

Define for instance, then alg.

EG, Graham's rule.

Better, LPT has ratio  $4/3$ .

Bin packing. First fit has ratio 2 since at most one  $1/2$  empty bin.

Wait, what about that bin

Solution: asymptotic relative performance ratio.