1 Online algorithms

Motivation:

- till now, our algorithms start with input, work with it
- (exception: data structures—come back later)
- now, suppose input arrives a little at a time, need instant response
- eg stock market, paging
- question: what is a “good” algorithm.
- depends on what we measure.
- if knew whole input $\sigma$ in advance, easy to optimize $C_{MIN}(\sigma)$
- ski rental problem: rent 1, buy $T$. don’t know how often.
- notice that on some inputs, can’t do well! (stock market that only goes down, thrashing in paging)
- problem isn’t to decide fast, rather what to decide.

Definition: competitive ratio

- compare to full knowledge optimum
- $k$-competitive if for all sequences etc. $C_A(\sigma) \leq kC_{MIN}(\sigma)$
- sometimes, to ignore edge effects, $C_A(\sigma) \leq kC_{MIN}(\sigma) + O(1)$.
- idea: “regret ratio”
- analyze ski rental
- we think of competitive analysis as a (zero sum) game between algorithm and adversary. want to find best strategy for algorithm.
- supposed to be competitive against all sequences. So, can imagine that adversary is adapting to algorithm’s choices (to get worst sequence)

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

More recent algs do somewhat better:
• keep some machines small
• algorithms not too bad, proofs awful!

1.1 Move to front

Allowed to move up accessed item; other transposes cost 1.
Potential function: number of inversions.

• amortized cost
• suppose search for item $x_j$ at $j$ in opt, at $k$ in MTF
• suppose $v$ items precede $x_k$ but not $x_j$
• then $k - v - 1$ precede in BOTH
• so $k - v - 1 \leq j - 1$ so $k - v \leq j$
• MTF creates $k - v - 1$ new inversions and kills $v$ old ones,
• so amortized cost is $k + (k - v - 1) - v \leq 2(k - v) \leq 2j$
• now do opt’s move.
• moving $x_j$ towards front only decreases inversions (already at front in MTF)
• other transposes increase potential but are paid for.

2 Paging problem

• define
• LRU, FIFO, LIFO, Flush when full, Least freq use
• LIFO, LFU not competitive
• LRU, FIFO $k$-competitive.
• will see this is best possible (det)
LRU is $k$-competitive

- Note we prove this without knowing opt!
- Assume start with same pages in memory (adds const)
- Phase: $k$ page faults, ending with last fault (start counting after first fault)
- Show 1 fault to MIN in each phase
- Case 1: Two faults on $p$ in 1 phase
  - Then had accesses to $k$ other pages between faults to $p$
  - So $k + 1$ pages accessed in phase—MIN must fault once.
- Case 2: $k$ distinct faults
  - Let $p$ be last fault of previous phase
  - Case 2a: Fault to $p$ in phase. Then argue as before, $k$ pages between $p$ faults
  - Case 2b: No fault to $p$, immediately after first $p$-fault, MIN has $p$ in memory, other $k - 1$ pages. $k$ new pages accessed in phase. Deduce one faults MIN.

- Notice: In case 2, fault we charge to phase might happen before phase.
  - But, happens after last fault-for-LRU in previous phase
  - So is different fault than the one deduced for previous phase.

Observations:

- Proved without knowing optimum
- Instead, derived lower bound on cost of any algorithm
- Same argument applies to FIFO.

Lower bound: No online algorithm beats $k$-competitive.

- Set of $k + 1$ pages
- Always ask for the one $A$ doesn’t have
- Faults every time.
- So, just need to show can get away with 1 fault every $k$ steps
- Have $k$ pages in memory. When fault, look ahead, one of $k + 1$ isn’t used in next $k$, so evict it.
- One fault every $k$ steps
so $A$ is only $k$-competitive.

Observations:
- $lb$ can be proven without knowing OPT, often is.
- competitive analysis doesn’t distinguish LRU and FIFO, even though know different in practice.
- still trying to refine competitive analysis to measure better: new SODA paper: “LRU is better than FIFO”
- applies even if just have $k + 1$ pages!

Optimal offline algorithm: Longest Forward Distance
- evict page that will be asked for farthest in future.
- suppose MIN is better than LFD. Will make NEW, as good, agrees more with LFD.
- Let $\sigma_i$ be first divergence of MIN and LFD (at page fault)
- LFD discards $q$, MIN discards $p$ (so $p$ will be accessed before $q$ after time $i$)
- Let $t$ be time MIN discards $q$
- revise schedule so MIN and LFD agree up to $t$, yielding NEW
- NEW discards $q$ at $i$, like LFD
- so MIN and NEW share $k - 1$ pages. will preserve till merge
- in fact, $q$ is unique page that MIN has that new doesn’t
- case 1: $\sigma_i, \ldots, \sigma_t, \ldots, p, \ldots, q$
  - until reach $q$
  - let $e$ be unique page NEW has that MIN doesn’t (init $e = p$)
  - when get $\sigma_l \neq e$, evict same page from both
  - note $\sigma_l \neq q$, so MIN does fault when NEW does
  - both fault, and preserves invariant
  - when $\sigma_l = e$, only MIN faults
  - when get to $q$, both fault, but NEW evicts $e$ and converges to MIN.
  - clearly, NEW no worse than MIN
- case 2: $t$ after $q$
  - follow same approach as above till hit $q$
- since MIN didn’t discard q yet, it doesn’t fault at q, but
- since p requested before q, had $\sigma_I = e$ at least once, so MIN did worse
  than NEW. (MIN doesn’t have p till faults)
- so, fault for NEW already paid for
  - still same.

• prove that can get to LFD without getting worse.
• so LFD is optimal.

**Randomized Online Algorithms**

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 $\sigma$, $E[CA(\sigma)] \leq k \cdot OPT + O(1)$.

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

Idea: evict random page?

• $k$-competitive against *adaptive* adversary
• but uses no memory
• trading space for randomness

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

  - so total at most $c_i \log k$

Offline cost:

- potential function $\Phi_i = \text{difference between } M \text{ and } O \text{ (offline) at start of phase } i$.
- 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, $C_i(O) \geq \max(c_i - \Phi_i, \Phi_i + 1) = \frac{1}{2}(c_i + \Phi_i - \Phi_{i+1})$.
- 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 $c$-competitive against an input distribution $p_{\sigma}$ if $E_{\sigma}(C_{A}(\sigma)) \leq cE_{\sigma}(C_{OPT}(\sigma))$ (note: OPT gets to see sequence before going)
• Theorem: if for some distribution no deterministic alg is $c$-competitive, than no randomized algorithm is $c$-competitive.
• to prove, suppose have $c$-competitive randomized alg, show det $c$-competitive against any $\sigma$.
• consider payoff $E_{A}[C_{A}(\sigma) - cC_{OPT}(\sigma)]$
• 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 $E_{\sigma}[C_{A}(\sigma) - cC_{OPT}(\sigma)] \leq 0$
• in other words, $A$ is $c$-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.
- in 2001, 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.

2.1 **On a Line**

Greedy algorithm bad if requests alternate $a$ near $b$ but server on distant $c$.

Double coverage algorithm (DC):
- If request outside conv hull, move nearest point to it.
- else, move nearest point on each side towards it equal distance till one hits.

$k$-competitive.
- let $M$ be min-cost matching of opt points to DC points
• $\Phi = kM + \sum_{i<j} d(s_i, s_j)$

• show:
  - adversary moves $d$: increases $\Phi$ by $\leq kd$
  - DC moves moves $d$: decrease $\Phi$ by $d$

• deduce: DC is $k$-competitive because it moves only $k$ times opt.

Analysis:

• adv moves $d$ just increases $M$ by $d$, so $\Delta \Phi \leq kd$

• DC moves $d$.

• If to outside hull, note adversary already has a point at dest; moving point must match to it (else matches something else; uncross).

• so $\Delta M = -d$ while $\delta \Sigma = (k-1)d$. claim follows: $\Delta \phi = -kd + (k-1)d = -d$

• if inside hull, one of moving points is matched to request. So that move decreases $M$. Other move may increase $M$ same amount, so no change to $M$.

• Now consider $\Sigma$. Moves of two points cancel out with respect to other points, but they get $2d$ units closer.

Generalizes to trees: all servers neighboring a request move toward it. (server stops if other moving server “blocks” it.

• as before, if opt moves $d$, change $kd$ in matching contrib to $\Phi$

• for DC, suppose $m$ servers move

• as before, one moving neighbor is matched, decreases $M$. $m-1$ others increase. total $(m-2)kd$

• consider any nonmoving server: 1 moving away from it, $m$ moving towards. total $-(k-m)(m-2)d$

• moving pairs approaching each other: total $-m(m-1)(2d)/2$

• add up, get $dm$

Application: weighted paging

• cost $w(p)$ to load $p$ (equiv, $w(p)/2$ to load and same to evict)

• treat as star, with edge lengths $w(p)$
3 Finance

Known or unknown duration. But assume know which offer is last. Need fluctuation ratio $\phi$ between largest $M$ and smallest $m$ price.

Selling peanuts:
- Break into $\log \phi$ groups of equal amounts
- Sell group $i$ for value $m \cdot 2^i$
- One group sold for at least half of max price
- So achieve $\log \phi$ competitive

Selling (one) car: Best deterministic algorithm: agree to first price exceeding $\sqrt{Mm}$
- $\sqrt{\phi}$ competitive
- note have to know when last offer

Can achieve $\log \phi$ randomized
- Consider powers of 2 between $m$ and $M$
- Choose one at random
- sell all at first bid exceeding
- with prob $1/\log \phi$, pick the power of 2 that is within factor 2 of highest offered price.
- even if know $\phi$ but don’t know $m$, can just run above alg after seeing first price