New slides posted ~15 minutes ago
(a few extra announcements / fixed typos)

Online Algorithms

CSE 417 Winter 21
Lecture 27

Announcements

Added some hints and corrected some typos on the homework yesterday afternoon – grab a fresh version from the webpage.

The solutions for the practice exam don't quite match up with the exam itself – we'll add the missing solutions this afternoon.

Announcements

Want a group for talking about the final during the final? “Megathread” on Ed.

If you need a different slot for the final, email Robbie ASAP.

Remember you can’t use late days on the final.
If something unexpected happens email me as soon as you’re concerned you won’t be able to finish on time.

U.S. “springs forward” very early Sunday morning, we’ll be GMT-7 (not GMT-8) when the final starts and ends.

Announcements

You’re “responsible for” the content through Monday on the final.

Today’s content won’t be on the final.

Friday is going to be a problem session (so no new content).

One more logistical note: once final is released, we won’t answer anything except “clarifying” questions on Ed (and only via private posts).

Like we would in an in-person exam.

Announcements

We'll have 29 polleverywheres; 25 will be enough for full credit (extras don't help). If you have extra late days those count one-for-one for missed pollevs.

Extra polleverywhere opens this afternoon (version on canvas will open Friday morning).

Asks for what problem (or what topic) you want to go through in the problem session.

And we'll have number 29 on Friday.

Online Algorithms

All of our algorithms this year have gotten an input and produced an output.

But sometimes you don't get the full input. You only get part of it, and you have to make decisions **while** you're getting the rest.

Examples:

Should I get a postmates subscription or just keep paying the delivery fee (continued input: how many more times do I order)?

Should lyft match you to a driver far away who is ready now, or hope someone closer to them will log on and request a ride?

Should you try to hire someone? If you wait you can do more interviews and maybe find someone better…or they might accept another.

Measuring Quality

How do we know if our online algorithm is good?

For optimization problems (find the min), let ALG be what your algorithm finds, let OPT be the true best answer **knowing the input that's coming.**

We achieve a "competitive ratio" of α if:

$$ALG \leq \alpha \cdot OPT$$

So it's the same idea as we had for approximation algorithms!

(for find the max, switch the inequality direction)

Ski Rental

You and a group of friends have decided to take up skiing.

Every time you go skiing, you can either:

Rent skis for the day (say it costs $1, for simplicity of calculation)

Buy skis, cost $B and can use them forever (don't need to rent again).

Every time you go skiing, you have the option of rent or buy.

It's unclear how long you and your friends will keep up this hobby. How many times should you rent before you buy?

What Information Do We Get?

Every time your friends call and ask “wanna go skiing this weekend?” you decide whether to run out and buy your own skis or to rent another time.

You aren’t going to get a good sense of how often this will happen. At any time your friends might give up the hobby. You can’t estimate the chances of there being another trip.

Your goal is to decide on a plan in advance, how many times should you rent before deciding to buy. Your goal is to minimize the competitive ratio, i.e. minimize the factor of money you overpay.

Deterministic Algorithm

$B, $1 rent

Suppose you decide you'll rent for the first k days, and buy if you are invited for trip $k + 1$.

And let $\underline{D}$ be the true number of days you're invited.

$$\rightarrow = \begin{cases} D & \text{if } D \le B \\ B & \text{if } D \ge B \end{cases}$$

What is $\underline{OPT}$?

What is ALG?

$$\rightarrow = \begin{cases} D & \text{if } D \le k \\ k+B & D \ge k+1 \end{cases}$$

What's the worst case ratio? i.e. for any particular k what's the worst-case ratio? What value of k should you choose?

Deterministic Algorithm

Suppose you decide you'll rent for the first k days, and buy if you are invited for trip $k + 1$.

And let D be the true number of days you're invited.

What is OPT?

$$OPT = \begin{cases} D & \text{if } D < B \\ B & \text{otherwise} \end{cases}$$

If $D < B$ then only rent, if $D \geq B$ buy right away.
OPT has foresight (it's psychic) it knows $D \geq B$ and it's better to buy right away.

What is ALG?

What's the worst case ratio? i.e. for any particular k what's the worst-case ratio? What value of k should you choose?

Deterministic Algorithm

Suppose you decide you'll rent for the first k days, and buy if you are invited for trip $k+1$.

And let D be the true number of days you're invited.

If $D \leq k$ we rent all D days.
Otherwise we rent the first k days then buy

What is OPT?

What is ALG?

$$ALG = \begin{cases} D & \text{if } D \leq k \\ B + k & \text{otherwise} \end{cases}$$

What's the worst case ratio? i.e. for any particular k what's the worst-case ratio? What value of k should you choose?

Finding the competitive ratio

$$OPT = \begin{cases} D & \text{if } D < B \\ B & \text{otherwise} \end{cases} \quad ALG = \begin{cases} D & \text{if } D \leq k \\ B + k & \text{otherwise} \end{cases}$$

We get to choose k

If we choose $k = B - 1$, what is the worst D?

If we choose $k = B$ what is the worst D?

If we choose $k \leq B - 2$ what is the worst D?

Finding the competitive ratio

$OPT = \begin{cases} D & \text{if } D < B \\ B & \text{otherwise} \end{cases}$ $ALG = \begin{cases} D & \text{if } D \leq k \\ B + k & \text{otherwise} \end{cases}$

We get to choose k

If we choose $k = B - 1$, what is the worst D?

Worst D is B, ratio is $\frac{B-1+B}{B} = 2 - \frac{1}{B}$

If we choose $k \geq B$ what is the worst D?

Worst D is $\mathrm{k} + 1$, ratio is $\frac{k+B}{B} \geq 2$ as k gets bigger, we add a small amount to the ratio each time.

If we choose $k \leq B - 2$ what is the worst D?

Worst $D = k + 1$ ratio is $\frac{k+B}{k+1} = 1 + \frac{B-1}{k+1} \geq 2$

A slightly better algorithm

We might be worrying a bit too much about the absolute worst case.
Let's try to "spread out" the badness a little.

For example, let's say that B is 10.

The last algorithm has a competitive ratio of 1.9.

Here's a slightly better idea: flip a coin: if it's heads, choose $k = 7$, otherwise choose $k = 9$.

Analysis

How much do we pay for D days of skiing?

If $D \leq 7$,

We always pay D; exactly what OPT pays

If $8 \leq D \leq 9$

Half the time, pay $7 + 10$, half the time pay D

On average pay $\frac{17}{2} + \frac{D}{2}$, OPT pays D, ratio $\frac{1}{2} + \frac{9}{D}$ worst D is 9, gives $\frac{3}{2}$

If $D \geq 10$

Half the time, pay $7 + 10$, half the time pay $9 + 10$; OPT pays 10. ratio is $\frac{8+10}{10} = 1.8$ worst ratio is 1.8. That's better than 1.9 from before!

Huh?

How did randomness help?

Randomness makes it harder to find a "worst-case"

When $k = 8$ or $k = 10$ individually, the worst D is fixed. When we're choosing randomly, no value of D is always as bad as could be! There's at least a 50% chance the D that happens isn't the worst one!

So we "spread out" the badness and make our worst-case result a little better.

The Best Strategy

You should choose the day k to buy with probability:

$$p_k = \begin{cases} \boxed{\left(\frac{B-1}{B}\right)^{B-k}} \frac{1}{B\left(1-\left(1-\left(\frac{1}{B}\right)\right)^{B}\right)} & \text{if } k \leq B \\ 0 & \text{otherwise} \end{cases}$$

Don’t get scared – the second term is just normalizing (the probabilities have to sum to 1, ignore that part). As k increases, the probability we buy increases. Once we hit B, we’re guaranteed to buy.

Why?

A lot of math…

What's the ratio? As B gets very big, it approaches $\underbrace{\frac{e}{e-1}} \approx \underbrace{1.58}$.

Quite a bit better than $\underbrace{2 - \frac{1}{B}}$.

Please Make this app (for me)

Lets you enter the subscription vs. individual cost of services (lyft vs. buying a car, individual grocery delivery vs. subscription, etc.)

And have the app remember and flip the coin for you every time.

A Second Example

Secretary Problem

You're interviewing people for a job. But the job market is extremely competitive – By the time you interview person $i + 1$, person i will have taken another job.

There are n candidates in the pool. You'll interview them in a random order. You don't know much about the market (you can't tell who is "mediocre" vs. "amazing") but you can compare really well after interviews (after interviewing i candidates, you can rank those i relative to each other).

Your goal is to maximize your probability of hiring the **best** candidate.

A Good Algorithm

We're only going to extend an offer if the candidate is the best one we've seen so far (any other candidate is definitely not the best).

But we probably don't want to just take the first candidate, we want to interview for a bit…see some decent candidates (but hopefully not the best one too early…)

The longer we wait, the less chance that we'll pick someone bad (we've seen a good fraction of applicants)…but also the greater chance that we interview the best applicant too early.

A Good Algorithm

Interview first X applicants.

Starting with X+1, give an offer if and only if it's the best applicant you've seen.

What should X be?

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

What’s the best r?

Suppose we interview r people without considering offers – what’s the probability we select the best?

Sum over the possible locations i where the best person could be located.

$\frac{1}{n}$ chance best person is at location i. Given they’re at location i, when are you selected? If and only if best among the $i-1$ before you was among the first r interviewed. Probability $\frac{r}{i-1}$.

$$\left\{ \sum_{i=r+1}^{n} \frac{1}{n} \cdot \frac{r}{i-1} \right\} \qquad \sum_{t=1}^{n} \frac{1}{t}$$

So…what value of r maximizes that probability?

Best r

The best r is n/e (ask Robbie afterward about the math)

I.e. if you have n applicants, you should interview a $1/e$ fraction of them, then start considering offers.

Tl;dr: Interview about 37% of the pool that you're likely to see, then start considering offers.

One More Example

Online Matchings – Inspired by lyft

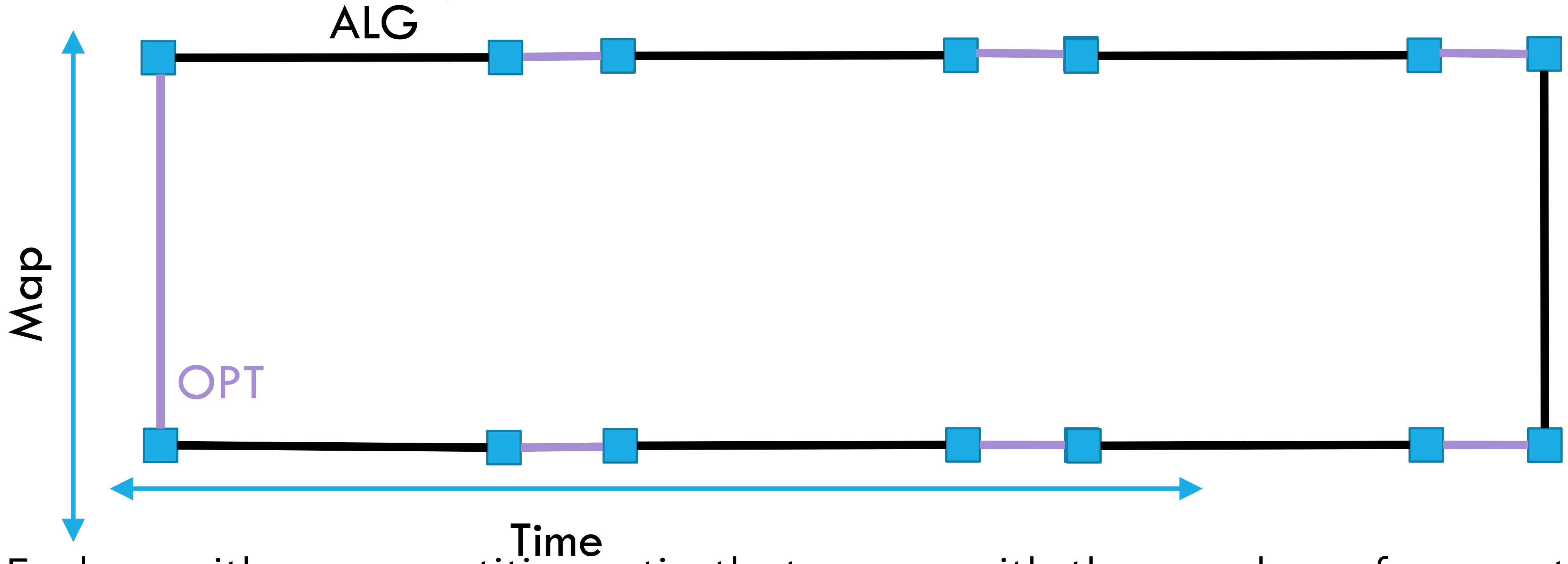
Requests (for a ride or to drive) will appear on a map over time. The cost to pair two requests is:

Distance between two requests + time each request waited between telling us where they are and us telling them “here’s your match”.

Our job is to decide who matches to who.

Worst-Case

Worst Case isn't very realistic for this problem.



End up with a competitive ratio that grows with the number of requests.
But…that input isn't realistic at all!

If you’re not worried about worst-case

If you’re worried about worst-case, there are fancy ideas (ask Robbie after class).

But you probably aren’t – there’s no enemy of your app placing requests on the map. Really they’re pretty close to random.

Under reasonable randomness assumptions, you can do very well. Here’s an algorithm:

When a request appears, it will only match to something at the same spot. After t seconds of waiting, it will match to anything within t distance of it.

If you’re not worried about worst case

i.e. grow a circle around each point. Its radius grows continuously as it waits.

Big Idea: Balance the “time” and “distance” penalty.

That idea works even if you have a different kind of penalty for waiting (e.g. worsening penalties)