Online Algorithms

Scribe: Muhan Li

Benefitial readings:

Online Algorithms by Borodin et al.

WISC CS787 note

Introduction

Online algorithms are designed to deal with input piece by piece in the order they are fed, which means it may make decisions resulting in sub-optimal final outcomes. In contrast, an offline algorithm is given the whole problem data from the beginning, which means it is possible to compute a globally optimal solution.

Therefore, we use competitive analysis to formally compare the relative performance of an online and (optimal) offline algorithm for the same problem instance. More formally:

For a minimization problem:

$$ALG(I) \leq C * OPT(I)$$

Where:

1. C - competitive factor, $C \geq 1$.
2. $ALG(I)$ - cost of some problem instance returned by our algorithm
3. $OPT(I)$ - cost returned by the optimal solution.

For a maximization problem:

$$ALG(I) \geq C * OPT(I)$$

Where $C \leq 1$.

We are usually using the biggest C for all instances I in a minimization problem and the largest C for a maximization problem. We usually use "n-competitive" as a shorter form of saying "competitive ratio is n".

Ski rental problem

We would like to move from position X to Y for some n days, we can either:

1. Rent a ski for A dollars / day.
2. Buy a ski permanently for B dollars,

Usually, we normalize both costs and divide them by A, get $A' = 1$ and $B' = B/A$ to simplify our analysis.

Deterministic solution

Algorithm:

> Day 1~$\lfloor B \rfloor$, rent ski with cost 1.
>
> Day $\lceil B \rceil$, buy ski with cost B.

Theorem:

This algorithm is 2-competitive.

Proof:

Let n be the number of days we ski, then $OPT = min(n, B)$.

For our algorithm there are two cases:

1. $n < B$, cost is n.
2. $n \geq \lceil B \rceil$, cost is $\lfloor B \rfloor + B$.

When $n < B$, competitive ratio is 1.

When $n \geq \lceil B \rceil$, the competitive ratio is:

$$\frac{ALG(I)}{OPT(I)} = \frac{\lfloor B \rfloor + B}{B} \leq \frac{2B}{B} = 2$$

Randomized solution

For a randomized algorithm, we analyze its performance using the expected competitive ratio:

$$\text{competitive ratio} \leq \max_I \frac{\mathbb{E}[ALG(I)]}{OPT(I)}$$

And we say that the algorithm is n-competitive if $\forall I, \mathbb{E}[ALG(I)] \leq n * OPT(I)$

Algorithm:

Define probability distribution with density function:

$$f(x) = \frac{e^{x/B}}{B(e-1)}, x > 0$$

We sample t according to this distribution and buys the ski on day $\lceil t \rceil$.

Theorem:

This algorithm is $\frac{e}{e-1}$-competitive (approximately 1.58).

Proof:

The cumulative distribution function is:

$$\begin{aligned} F(x) &= P\{T \leq x\} \\ &= \int_0^x f(t)\, dt \\ &= \int_0^x \frac{e^{t/B}}{B(e-1)}\, dt \\ &= \frac{1}{e-1}(e^{t/B}\Big|_0^x) \\ &= \frac{e^{x/B} - 1}{e-1} \end{aligned}$$

1. When $n \leq B$:

   Cost of the optimal solution is $OPT = n$.

   Expectation cost of our algorithm is:

$$
\begin{aligned}
\mathbb{E}[ALG] &= \mathbb{E}[\text{rental cost}] + \mathbb{E}[\text{buy cost}] \\
&= \mathbb{E}[\int_0^n \mathbb{I}\{\text{we rent at time } t\}\, dt] + \underbrace{P\{T \le n\}}_{\text{we buy } \le n} * B \\
&\le \int_0^n P\{\text{we rent at time } t\}\, dt + P\{T \le n\} * B
\end{aligned}
$$

Note we use inequality $\le$ instead of equality in the last equation, because we are converting discrete renting events to contiguous events (the last day we rent was $\lfloor t_0 \rfloor$ originally, but now it is t_0).

Therefore:

$$
\begin{aligned}
&\int_0^n P\{\text{we rent at time } t\}\, dt + P\{T \le n\} * B \\
&= \int_0^n P\{T \ge t\}\, dt + P\{T \le n\} * B \\
&= \int_0^n (1 - \frac{e^{t/B} - 1}{e - 1})\, dt + P\{T \le n\} * B \\
&= n - \frac{B(e^{n/B} - 1)}{e - 1} + \frac{n}{e - 1} - \frac{B(e^{n/B} - 1)}{e - 1} \\
&= (1 + \frac{1}{e - 1})n \\
&= \frac{e}{e - 1} n
\end{aligned}
$$

And competitive ratio of the randomized algorithm is $\frac{e}{e-1}$.

2. When $n > B$:

   Since event $T \in [0, B]$ happens with probability 1 according to the cumulative distribution function (let $x = B$ in $F(x) = \frac{e^{x/B} - 1}{e - 1}$), which means we will always buy the ski $\le B$, and these cases are equivalent to the $n = B$ case, and the competitive ratio is still $\frac{e}{e-1}$.

Why randomization helps (Yao's Lemma)

According to Yao's minimax principle:

> The worst-case expected cost of the randomized algorithm is at least the cost of the best deterministic algorithm against input distribution q.

A more straight forward explanation is that randomly guessing (Eg: evict a page that probably will not be used in the future) has some probability of being correct. While the performance of a deterministic algorithm depends on the occurring probability of the worst sequences.

Caching problem

The LRU algorithm

Algorithm:

Keeps a increasing rank number.

When any page is missed, fetch it from storage, removed the page with the smallest rank, then assign the latest rank to the new fetched page and store it in cache.

When any page is hit, assign it the latest rank.

Theorem:

LRU is k-competitive. (k is cache size in page number)

Proof:

(Note: this is the proof talked in class, however there are also many other similar proofs talked about in courses offered by other universities such as this one)

We first prove that LRU has competitive ratio $\leq k$, then show that it has competitive ratio $\geq k$, then conclude that its competitive ratio is k.

Theorem 1

LRU has $C \leq k$.

Proof:

We partition the sequence into phases, a phase immediately starts when OPT has made one page fault, and ends before the next OPT page fault.

Each phase has at most k **distinct** pages, otherwise two page faults will occur and this contradicts our definition.

Let p be some page LRU faults on, we can prove by contradiction that LRU will not fault on p in k distinct pages, because otherwise before faulting on p the second time, LRU should have met with k distinct pages, resulting in $k+1$ distinct pages in total including p, which contradicts the condition.

Therefore LRU has $C \leq k$.

Theorem 2

LRU has $C \geq k$

Proof:

It is easy to construct the following sequence which repeats every $k+1$ pages:

$$1, 2, \ldots, k, k+1, 1, 2, \ldots, k, k+1, \ldots$$

LRU will fault $k+1$ times in every block of length $k+1$.

OPT will fault 1 time in every block (after first $1, 2, \ldots, k$), first fault on $k+1$, then k, $k-1$,..... It is straight forward to simulate OPT by removing the latest accessed page in future window of size k and replace it with the missing page.

Therefore LRU has $C \geq k$

With Theorem 1 and 2 we can conclude that LRU is k-competitive.

Beyond worst-case analysis (BWCA)

There are many ways to perform BWCA, we will use Resource augmentation here.

We are using this price (augmented resource) to analyze how well our algorithm can perform compared to the optimum situation which we will not be able to reach.

LRU vs OPT

suppose LRU is working with a cache size slightly bigger than the OPT algorithm, Eg: $2k$ pages, because in most cases LRU is dealing with more complex scenarios than an off-line scenario.

LRU - $2k$, OPT - k (2 could be some number = 1 + ϵ)

Theorem:

LRU is 2-competitive in resource augmentation

Proof:

Also split runtime into blocks, where each block have $2k$ distinct page requests.

LRU:

If some page is evicted, the page must be some page requested before current time block. (since cache size is $2k$), in the worst case evict at most $2k$ pages.

OPT: can only keep k pages in cache, worst case $\geq k$ misses.

therefore it is 2-competitive.