26
Online Algorithms

In this chapter we introduce *online algorithms* and study two classic online problems: the rent-or-buy problem, and the paging problem. While the models we consider are reminiscent of those in *regret minimization* in online learning and online convex optimization, there are some important differences, which lend a different flavor to the results that are obtained.

26.1 *The Competitive Analysis Framework*

In the standard setting of online algorithms there is a sequence of requests $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_t, \ldots)$ received online. An online algorithm does not know the input sequence up-front, but sees these requests one by one. It must serve request σ_i before it is allowed to see σ_{i+1}. Serving this request σ_i involves some choice of actions, and incurs some cost. We will measure the performance of an algorithm by considering the ratio of the total cost incurred on σ to the optimal cost of serving σ in hindsight. To make all this formal, let us xsee an example of an online problem.

26.1.1 *Example: The Paging Problem*

The paging problem arises in computer memory systems. Often, a memory system consists of a large but slow *main memory*, as well as a small but fast memory called a *cache*. The CPU typically communicates directly with the cache, so in order to access an item that is not contained in the cache, the memory system has to load the item from the slow memory into the cache. Moreover, if the cache is full, then some item contained in the cache has to be evicted to create space for the requested item.

We say that a *cache miss* occurs whenever there is a request to an item that is not currently in the cache. The goal is to come up with an eviction strategy that minimizes the number of cache misses.

Typically we do not know the future requests that the CPU will make so it is sensible to model this as an online problem.

If the entire sequence of requests is known, show that *Belády's rule* is optimal: evict the page in cache that is next requested furthest in the future.

We let U be a universe of n items or pages. The cache is a memory containing at most k pages. The requests are pages $\sigma_i \in U$ and the online algorithm is an eviction policy. Now we return back to defining the performance of an online algorithm.

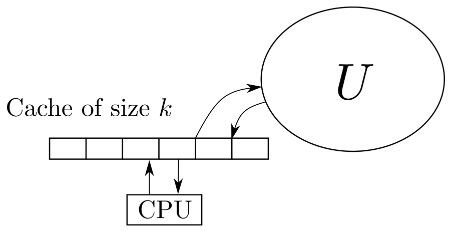

Figure 26.1: Illustration of the Paging Problem

26.1.2 The Competitive Ratio

As we said before, the online algorithm incurs some cost as it serves each request. If the complete request sequence is σ, then we let $\mathrm{Alg}(\sigma)$ be the total cost incurred by the online algorithm in serving σ. Similarly, we let $\mathrm{Opt}(\sigma)$ be the optimal cost in hindsight of serving σ. Note that $\mathrm{Opt}(\sigma)$ represents the cost of an optimal *offline* algorithm that knows the full sequence of requests. Now we define the *competitive ratio* of an algorithm to be:

$$\max_{\sigma} \frac{\mathrm{Alg}(\sigma)}{\mathrm{Opt}(\sigma)}$$

In some sense this is an "apples to oranges" comparison, since the online algorithm does not know the full sequence of requests, whereas the optimal cost is aware of the full sequence and hence is an "offline" quantity.

Note two differences from regret minimization: there we made a prediction x_t before (or concurrently with) seeing the function f_t, whereas we now see the request σ_t before we produce our response at time t. In this sense, our problem is easier. However, the benchmark is different—we now have to compare with the best *dynamic* sequence of actions for the input sequence σ, whereas regret is typically measured with respect to a *static* response, i.e., to the cost of playing the same fixed action for each of the t steps. In this sense, we are now solving a harder problem. There are is a smaller, syntactic difference as well: regret is an additive guarantee whereas the competitive ratio is a multiplicative guarantee—but this is more a reflection on the kind of results that are possible, rather than fundamental difference between the two models.

26.1.3 What About Randomized Algorithms?

The above definitions generally hold for deterministic algorithms, so how should we characterize randomized algorithms. For the deterministic case we generally think about some adversary choosing the worst possible request sequence for our algorithm. For randomized algorithms we could consider either oblivious or adaptive adversaries. Oblivious adversaries fix the input sequence up front and then let the randomized algorithm process it. An adaptive adversary is allowed to see the results of the coin flips the online algorithm makes and thus adapt its request sequence. We focus on oblivious adversaries in these notes.

To define the performance of a randomized online algorithm we just consider the expected cost of the algorithm. Against an oblivious adversary, we say that a randomized online algorithm is α-competitive if for all request sequences σ,

$$\mathbb{E}[\mathrm{Alg}(\sigma)] \leq \alpha \cdot \mathrm{Opt}(\sigma).$$

26.2 *The Ski Rental Problem: Rent or Buy?*

Now that we have a concrete analytical framework, let's apply it to a simple problem. Suppose you are on a ski trip with your friends. On each day you can choose to either rent or buy skis. Renting skis costs \$1, whereas buying skis costs \$$B$ for $B > 1$. However, the benefit of buying skis is that on subsequent days you do not need to rent or buy again, just use the skis you already bought. The problem that arises is that for some mysterious reason we do not know how long the ski trip will last. On each morning we are simply told whether or not the trip will continue that day. The goal of the problem is to find a rent/buy strategy that is competitive with regards to minimizing the cost of the trip.

In the notation that we developed above, the request for the i'th day, σ_i, is either "Y" or "N" indicating whether or not the ski trip continues that day. We also now that once we see a "N" request that the request sequence has ended. For example a possible sequence might be $\sigma = (Y, Y, Y, N)$. This allows us to characterize all instances of the problem as follows. Let I_j be the sequence where the ski trip ends on day j. Suppose we knew ahead of time what instance we received, then we have that $\mathrm{Opt}(I_j) = \min\{j, B\}$ since we can choose to either buy skis on day 1 or rent skis every day depending on which is better.

26.2.1 *Deterministic Rent or Buy*

We can classify and analyze all possible deterministic algorithms since an algorithm for this problem is simply a rule deciding when to buy skis. Let Alg_i be the algorithm that rents skis until day i, then buys on day i if the trip lasts that long. The cost on instance I_j is then $\mathrm{Alg}_i(I_j) = (i - 1 + B) \cdot \mathbf{1}_{\{i \leq j\}} + j \cdot \mathbf{1}_{\{i > j\}}$. What is the best deterministic algorithm from the point of view of competitive analysis? The following claims answer this question.

Lemma 26.1. *The competitive ratio of algorithm* Alg_B *is* $2 - 1/B$ *and this is the best possible ratio for any deterministic algorithm.*

Proof. There are two cases to consider $j < B$ and $j \geq B$. For the first case, $\text{Alg}_B(I_j) = j$ and $\text{Opt}(I_j) = j$, so $\text{Alg}_B(I_j)/\text{Opt}(I_j) = 1$. In the second case, $\text{Alg}_B(I_j) = 2B - 1$ and $\text{Opt}(I_j) = B$, so $\text{Alg}_B(I_j)/\text{Opt}(I_j) = 2 - 1/B$. Thus the competitive ratio of Alg_B is

$$\max_{I_j} \frac{\text{Alg}_B(I_j)}{\text{Opt}(I_j)} = 2 - 1/B$$

Now to show that this is the best possible competitive ratio for any deterministic algorithm. Consider algorithm Alg_i. We find an instance I_j such that $\text{Alg}_i(I_j)/\text{Opt}(I_j) \geq 2 - 1/B$. If $i \geq B$ then we take $j = B$ so that $\text{Alg}_i(I_j) = (i - 1 + B)$ and $\text{Opt}(I_j) = B$ so that

$$\frac{\text{Alg}_i(I_j)}{\text{Opt}(I_j)} = \frac{i - 1 + B}{B} = \frac{i}{B} + 1 - \frac{1}{B} \geq 2 - \frac{1}{B}$$

Now if $i = 1$, we take $j = 1$ so that

$$\frac{\text{Alg}_i(I_j)}{\text{Opt}(I_j)} = \frac{B}{1} \geq 2$$

Since B is an integer > 1 by assumption. Now for $1 < i < B$, we take $j = \lfloor (i - 1 + B)/(2 - 1/B) \rfloor \geq 1$ so that

$$\frac{\text{Alg}_i(I_j)}{\text{Opt}(I_j)} \geq 2 - \frac{1}{B}.$$

□

26.2.2 *Randomized Rent or Buy*

Can randomization improve over deterministic algorithms in terms of expected cost? We will show that this is in fact the case. So how do we design a randomized algorithm for this problem? We use the following general insight about randomized algorithms, notably that *a randomized algorithm is a distribution over deterministic algorithms*.

To keep things simple let's consider the case when $B = 4$. We construct the following table of payoffs where the rows correspond to deterministic algorithms Alg_i and the columns correspond to instances I_j.

	I_1	I_2	I_3	I_∞
Alg_1	4/1	4/2	4/3	4/4
Alg_2	1/1	5/2	5/3	5/4
Alg_3	1/1	2/2	6/3	6/4
Alg_4	1/1	2/2	3/3	7/4

While the real game is infinite along with coordinates, we do not need to put columns for $I_4, I_5, \ldots$ because these strategies for the adversary are dominated by the column I_∞. Now given that the adversary would rather give us only these inputs, we do not need to put rows after $B = 4$ since buying after day B is worse than buying on day B for these inputs. (Please check these for yourself!) This means we can think of the above table as a 2-player zero-sum game with 4 strategies each. The row player chooses an algorithm and the column player chooses an instance, then the number in the corresponding entry indicates the loss of the row player. Thinking along the lines of the Von Neumann minimax theorem, we can consider mixed strategies for the row player to construct a randomized algorithm for the ski rental problem.

Let p_i be the probability of our randomized algorithm choosing row i. What is the expected cost of this algorithm? Suppose that the competitive ratio was at most c in expectation. The expected competitive ratio of our algorithm against each instance should be at most c, so this yields the following linear constraints.

$$\begin{aligned} 4p_1 + p_2 + p_3 + p_4 &\leq c \\ \frac{4p_2 + 5p_2 + 2p_3 + 2p_4}{2} &\leq c \\ \frac{4p_1 + 5p_3 + 6p_3 + 3p_4}{3} &\leq c \\ \frac{4p_1 + 5p_2 + 6p_3 + 7p_4}{4} &\leq c \end{aligned}$$

We would like to minimize c subject to $p_1 + p_2 + p_3 + p_4 = 1$ and $p_i \geq 0$. It turns out that one can do this by solving the following system of *equations*:

Why is it OK to set the inequalities to equalities? Simply because it works out: in essence, we are guessing that making these four constraints tight gives a basic feasible solution—and our guess turns out to right. It does not show optimality, but we can do that by giving a matching dual solution.

$$\begin{aligned} p_1 + p_2 + p_3 + p_4 &= 1 \\ 4p_1 + p_2 + p_3 + p_4 &= c \\ 4p_1 + 5p_2 + 2p_3 + 2p_4 &= 2c \\ 4p_1 + 5p_2 + 6p_3 + 3p_4 &= 3c \\ 4p_1 + 5p_2 + 6p_3 + 7p_4 &= 4c \end{aligned}$$

Subtracting each line from the previous one gives us

$$\begin{aligned} p_1 + p_2 + p_3 + p_4 &= 1 \\ 3p_1 &= c - 1 \\ 4p_2 + p_3 + p_4 &= c \\ 4p_3 + p_4 &= c \\ 4p_4 &= c. \end{aligned}$$

This gives us $p_4 = c/4$, $p_3 = (3/4)(c/4)$, etc., and indeed that

$$c = \frac{1}{1-(1-1/4)^4} \qquad \text{and} \qquad p_i = (3/4)^{i-1}(c/4)$$

for $i = 1, 2, 3, 4$. For general B, we get

$$c = c_B = \frac{1}{1-(1-{}^1\!/_B)^B} \leq \frac{e}{e-1}.$$

Moreover, this value of c_B is indeed the best possible competitive ratio for anyth randomized algorithm for the ski rental problem. How might one prove such a result? We instead consider playing a random instance against a deterministic algorithm. By Von Neumann's minimax theorem the value of this should be the same as what we considered above. We leave it as an exercise to verify this for the case when $B = 4$.

26.2.3 (Optional) A Continuous Approach

This section is quite informal right now, needs to be made formal. For simplicity, assume $B = 1$ by scaling, and that both the algorithm and the length of the season can be any real in $[0, 1]$. Now our randomized algorithm chooses a threshold $t \in [0, 1]$ from the probability distribution with density function $f(t)$. Let's say f is continuous. Then we get that for any season length ℓ,

$$\int_{t=0}^{\ell} (1+t)\, f(t)\, dt + \int_{t=\ell}^{1} \ell\, f(t)\, dt = c \cdot \ell.$$

(Again, we're setting an equality there without justification, except that it works out.) Now we can differentiate w.r.t. ℓ to get

$$(1+\ell)\, f(\ell) + \int_{t=\ell}^{1} f(t)\, dt - \ell f(\ell) = c.$$

(This differentiation is like taking differences of successive lines that we did above.) Simplifying,

$$f(\ell) + \int_{t=\ell}^{1} f(t)\, dt = c. \tag{26.1}$$

Taking derivatives again:

$$f'(\ell) - f(\ell) = 0$$

But this solves to $f(\ell) = Ce^{\ell}$ for some constant C. Since f is a probability density function, $\int_{\ell=0}^{1} f(\ell) = 1$, we get $C = \frac{1}{e-1}$. Substituting into (26.1), we get that the competitive ratio is $c = \frac{e}{e-1}$, as desired.

26.3 The Paging Problem

Now we return to the paging problem that was introduced earlier and start by presenting a disappointing fact.

Lemma 26.2. *No deterministic algorithm can be $<$ k-competitive for the paging problem.*

Proof. Consider a universe with $k+1$ pages in all. In each step the adversary requests a page not in the cache (there is always at least 1 such page). Thus the algorithm's cost over n requests is n. The optimal offline algorithm can cut losses by always evicting the item that will be next requested furthest in the future, thus it suffers a cache miss every k steps so the optimal cost will be n/k. Thus the competitive ratio of any deterministic algorithm is at least $\frac{n}{n/k} = k$. □

It is also known that many popular eviction strategies are k-competitive such as *Least Recently Used (LRU)* and *First-In First-Out (FIFO)*. We will show that a 1-bit variant of LRU is k-competitive and also show that a randomized version of it achieves an $O(\log k)$-competitive randomized algorithm for paging.

26.3.1 The 1-bit LRU/Marking Algorithm

The 1-bit LRU/Marking algorithm works in phases. The algorithm maintains a single bit for each page in the universe. We say that a page is marked/unmarked if its bit is set to 1/0. At the beginning of each phase, all pages are unmarked. When a request for a page not in the cache comes, then we evict an arbitrary unmarked page and put the requested page in the cache, then mark the requested page. If there are no unmarked pages to evict, then we unmark all pages and start a new phase.

Lemma 26.3. *The Marking algorithm is k-competitive for the paging problem.*

Proof. Consider the i'th phase of the algorithm. By definition of the algorithm, Alg incurs a cost of at most k during the phase since we can mark at most k different pages and hence we will have at most k cache misses in this time. Now consider the first request after the i'th phase ends. We claim that Opt has incurred at least 1 cache miss by the time of this request. This follows since we have now seen $k+1$ different pages. Now summing over all phases we see that $\text{Alg} \leq k\,\text{Opt}$ □

Now suppose that instead of evicting an arbitrary unmarked page, we instead evicted an unmarked page uniformly at random. For this randomized marking algorithm we can prove a much better result.

Lemma 26.4. *The Randomized Marking Algorithm is $O(\log k)$-competitive*

Proof. We break up the proof into an upper bound on Alg's cost and a lower bound on Opt's cost. Before doing this we set up some notation. For the i^{th} phase, let S_i be the set of pages in the algorithm's cache *at the beginning of the phase*. Now define

$$\Delta_i = |S_{i+1} \setminus S_i|.$$

We claim that the expected number of cache misses made by the algorithm in phase i is at most $\Delta_i(H_k+1)$, where H_k is the k^{th} harmonic number. By summing over all phases we see that $\mathbb{E}[\text{Alg}] \leq \sum_i \Delta_i(H_k+1)$.

Now let R_i be the set of distinct requests in phase i. For each request in R_i we say that it is *clean* if the requested page is not S_i, otherwise the request is called *stale*. Every cache miss in the i^{th} phase is caused by either a clean request or a stale request.

1. The number of cache misses due to clean requests is at most Δ_i since there can be at most Δ_i clean requests in phase i: each clean request brings in a page not belonging to S_i into the cache and marks it, so it will be in S_{i+1}.

2. To bound the cache misses due to stale requests, suppose there have been c clean requests and s stale requests so far, and consider the $s+1^{st}$ stale request. The probability this request causes a cache miss is at most $\frac{c}{k-s}$ since we have evicted c random pages out of $k-s$ remaining stale requests. Now since $c \leq \Delta_i$, we have that the expected cost due to stale requests is at most

This is like the Airline seat problem, where we can imagine that c confused passengers get on at the beginning.

$$\sum_{s=0}^{k-1} \frac{c}{k-s} \leq \Delta_i \sum_{s=0}^{k-1} \frac{1}{k-s} = \Delta_i H_k.$$

Now the expected total cost in phase i is at most

$$\Delta_i H_k + \Delta_i = \Delta_i(H_k+1).$$

Now we claim that $\text{Opt} \geq \frac{1}{2}\sum_i \Delta_i$. Let S_i^* be the pages in Opt's cache at the beginning of phase i. Let ϕ_i be the number of pages in S_i but not in Opt's cache at the beginning of phase i, i.e., $\phi_i = |S_i \setminus S_i^*|$. Now let Opt_i be the cost that Opt incurs in phase i. We have that $\text{Opt}_i \geq \Delta_i - \phi_i$ since this is the number of "clean" requests that Opt sees. Moreover, consider the end of phase i. Alg has the k most recent requests in cache, but Opt does not have ϕ_{i+1} of them by definition of ϕ_{i+1}. Hence $\mathrm{Opt}_i \geq \phi_{i+1}$. Now by averaging,

$$\mathrm{Opt}_i \geq \max\{\phi_{i+1}, \Delta_i - \phi_i\} \geq \frac{1}{2}(\phi_{i+1} + \Delta_i - \phi_i).$$

So summing over all phases we have

$$\mathrm{Opt} \geq \frac{1}{2}\sum_i \Delta_i + \phi_{final} - \phi_{initial} \geq \frac{1}{2}\sum_i \Delta_i,$$

since $\phi_{final} \geq 0$ and $\phi_{initial} = 0$. Combining the upper and lower bound yields

$$\mathbb{E}[\mathrm{Alg}] \leq 2(H_k + 1)\,\mathrm{Opt} = O(\log k)\,\mathrm{Opt}. \qquad \square$$

It can also be shown that no randomized algorithm can do better than $\Omega(\log k)$-competitive for the paging problem. For some intuition as to why this might be true, consider the *coupon collector problem*: if you repeatedly sample a uniformly random number from $\{1, \ldots, k+1\}$ with replacement, show that the expected number of samples to see all $k+1$ coupons is H_{k+1}.

26.4 Generalizing Paging: The k-Server Problem

Another famous problem in online algorithms is the *k-server* problem. Consider a metric space $M = (V, d)$ with point set V and distance function $d : V \times V \to \mathbb{R}_+$ satisfying the triangle inequality. In the k-server problem there are k servers that are located at various points of M. At each timestep t we receive a request $\sigma_t \in V$. If there is a server at point σ_t already, then we can server that request for free. Otherwise we move some server from point x to point σ_t and pay a cost equal to $d(x, \sigma_t)$. The goal of the problem is to serve the requests while minimizing the total cost of serving them.

The paging problem can be modeled as a k-server problem as follows. We let U be the points of the metric space and take $d(x, y) = 1$ for all pages x, y where $x \neq y$. This special case shows that every deterministic algorithm is at least k-competitive and every randomized algorithm is $\Omega(\log k)$-competitive by the discussion in the previous section. It is conjectured that there is a k-competitive deterministic algorithm: the best known result is a $(2k-1)$-competitive algorithm of Elias Koutsoupias and Christos Papadimitriou.

For randomized algorithms, a poly-logarithmic competitive algorithm was given by Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Seffi Naor. This was recently improved via an approach based on Mirror descent by Sebastien Bubeck, Michael Cohen, and James Lee, Yin Tat Lee, Aleksander Madry; see this paper of Niv Buchbinder, Marco Molinaro, Seffi Naor, and myself for a discretization.