Notes on Online Algorithms (COL 758)

1 Introduction

In typical algorithmic problems we have seen so far, the algorithm is given the entire input and is expected to produce the desired output. The principal non-triviality lies in designing algorithms that are fast. In the setting of online algorithms, the entire input is not given to us in advance. Instead, input arrives over time, and the algorithm is expected to maintain, at each time, a solution to the part of the input that has arrived till this time. More formally, let $\sigma_1, \sigma_2, \ldots, \sigma_t, \ldots$ be the input that arrives over time, and let σ_t denote the input that arrives at time t. At time t, the algorithm maintains a solution to the input $(\sigma_1, \ldots, \sigma_t)$ – denote this by π_t. The next question is how to analyze such algorithms. We use the notion of "competitive ratio". Assume that we are solving a minimization problem (for example, minimizing cost of a solution or some similar objective). Let $\mathcal{A}$ be an online algorithm, and let $\texttt{cost}(\mathcal{A}(\pi_t))$ denote the cost of the solution maintained by $\mathcal{A}$ at time t. We compare $\mathcal{A}$ with an algorithm which knows the entire input in advance. In other words, let $\texttt{opt}(\pi_t)$ denote the cost of the optimal solution for the input π_t (note that this optimal solution *knows* the entire input π_t in advance. For example, the solutions corresponding to $\texttt{opt}(\pi_t)$ and $\texttt{opt}(\pi_{t'})$ for two different time t and t' can look very different from each other, whereas for an on-line algorithm, the solution for the latter time can only be an *extension* of the solution at the former time). Define the competitive ratio of $\mathcal{A}$ on this input $\sigma_1, \sigma_2, \ldots$ as

$$\max_{t \geq 1} \frac{\texttt{cost}(\mathcal{A}(\pi_t))}{\texttt{opt}(\pi_t)}.$$

In other words, we say that $\mathcal{A}$ is c-competitive for the input $\sigma_1, \sigma_2, \ldots$, if $\texttt{cost}(\mathcal{A}(\pi_t)) \leq c \cdot \texttt{opt}(\pi_t)$, for all time t. Finally, we say that $\mathcal{A}$ is c-competitive if it is c-competitive for all inputs. Sometimes the online algorithm will be randomized. In such cases, we say that $\mathcal{A}$ is c-competitive for the input $\sigma_1, \sigma_2, \ldots$, if the expected value of $\texttt{cost}(\mathcal{A}(\pi_t))$ is at most $c \cdot \texttt{opt}(\pi_t)$ for all time t. We now give several examples of online algorithms.

2 Ski Rental Problem

This problem explores the trade-off between renting and buying an item. Suppose there is an item (say a ski) which costs B units. You will need the item for T consecutive days, starting from day 1, where T is a non-negative integer. The online aspect arises from the fact that we do not know the value T in advance. On each day t, we get to know whether we need the item on this day, i.e., $T \geq t$, or not (i.e., $T = t - 1$). If we need the item on day t, we have two options – either we can rent the item for 1 unit of money just for day t, or we can buy it for a one time cost of B units (and then, do not have to rent it subsequently).

If the value of T is given to us, the optimal solution is easy – buy the ski if $T \geq B$, else rent it for the first T days. Clearly, the cost of this solution is $\min(B, T)$. Consider the following online algorithm – rent till the first $B - 1$ days. If we still need the item on day B, then buy it on day B. What is the cost of this solution? If $T \geq B$, the algorithm pays $B - 1 + B = 2B - 1$. If $T < B$, then the algorithm pays T, which is also the optimal cost. Therefore, its competitive ratio is equal to $\frac{2B-1}{B} = 2 - 1/B < 2$. It is also easy to show that no deterministic algorithm can have competitive ratio better than $2 - 1/B$. Indeed, any deterministic strategy just keeps a threshold D – it rents for the first D days, and then buys the item on

day $D+1$ (convince yourself that any deterministic strategy can only look like this). Now, consider an input for which $T = D$. The algorithm pays $D+B$, whereas the optimal solution is $\min(D, B)$. Now check that for any $D \geq 0$, $\frac{D+B}{\min(D,B)} \geq 2 - 1/B$.

Even though there cannot be a deterministic strategy which beats the bound $2 - 1/B$ (which is nearly 2 for large B), it turns out that there is a randomized algorithm with competitive ratio close to $e/(e-1)$, which is nearly 1.58. The intuition why a randomized algorithm may have a smaller competitive ratio is as follows. Recall that each deterministic strategy can be thought of as picking a threshold D, and renting till day D. We also saw that one can come up with an input on which the ratio of the cost of this strategy versus the optimal strategy will be at least $2-1/B$. However, notice that this "bad" input depends on the value of D, and so, it is possible that there is no single input which is simultaneously bad for all choices of D.

We now give details of this algorithm. Recall that there is no advantage in choosing a threshold $D \geq B$ – any such deterministic strategy will always be worse (on every possible input) than the strategy which buys the item on day B. Therefore, a randomized strategy needs to figure out D probabilities $p_0, \ldots, p_{B-1}$, where p_i should be thought of as the probability with which this algorithm will pick i as the threshold (i.e., it will rent for the first i days, and buy on day $i+1$). In other words, the randomized algorithm is as follows – pick a value $D \in \{0, \ldots, B-1\}$ such that the probability that D is i is exactly p_i (how will you implement this rule?). Now, the strategy is to rent for the first $D-1$ days, and if needed, buy on day D. Now, we compute the expected cost of this algorithm for various possible inputs. Let c denote the competitive ratio of this algorithm.

First consider an input where $T \leq B$. If the algorithm chooses D to be less than T, then its cost is $D+B$. If it chooses $D \geq T$, then its cost is T. Therefore its expected cost on this input is

$$\sum_{i<T} p_i \cdot (i+B) + \sum_{i \geq T} p_i \cdot T.$$

Since the optimal cost is T, we want

$$\sum_{i<T} p_i \cdot (i+B) + \sum_{i \geq T} p_i \cdot T \leq c \cdot B.$$

Note that we get one constraint for each value of T between 1 and B. What happens if we consider $T > B$? In this case, the expected cost of the online algorithm and the cost of the optimal algorithm remain same as those when $T = B$. Therefore, we do not get any new constraint when considering $T > B$.

Exercise 1 *Show that if we replace all the constraints above by equality, and require $p_0 + \ldots + p_{B-1} = 1$, we get $p_i = \frac{c}{B} \cdot (1-1/B)^{B-1-i}$ and $c = \frac{1}{1-(1-1/B)^B}$.*

The above exercise shows that by choosing p_i as mentioned above, we can get expected competitive ratio $\frac{1}{1-(1-1/B)^B} \sim \frac{e}{e-1}$.