Algorithms and Uncertainty, Winter 2017/18 Lecture 6 (4 pages)

The Secretary Problem and the Random-Order Model

Instructor: Thomas Kesselheim

Let us consider the following *online selection problem*, in which you have to make commitments before you know all you choices. Suppose you want to buy a house. You go see several houses and (a bit simplifying here) after each visit you have to decide immediately and irrevocably if you want to buy this particular house or if you want to keep on looking – then somebody else will buy it. Another motivation would be that you want to find the love of your life. You start dating and (even more simplifying here) after each first date you have to decide whether you you want to marry this person or if you want to keep looking.

We can model this problem as follows. There are n candidates of values $v_1, \dots v_n \in \mathbb{R}$, $v_i \geq 0$. You see the values of these candidates in order 1, ..., n. After having seen the i-th candidate you can choose to select it or to reject it. The goal is to maximize the value of the candidate that you select.

It is easy to observe that no algorithms have any reasonable performance if we apply competitive analysis as we did so far in this course.

Observation 6.1. *There is no deterministic online algorithm that is better than* 0*-competitive for any n. No randomized algorithm is c-competitive for $c > \frac{1}{n}$.*

Note that the trivial algorithm that simply chooses a uniformly drawn random candidate is $\frac{1}{n}$-competitive, although it does not even look at the values. This, of course, does not mean that this stupid algorithm is the best thing one could do. Competitive analysis as we have seen it so far just cannot express this. Therefore, we will consider a different model today.

We will consider the *random-order model*. First, an adversary defines values $v_1, \dots, v_n$ and then a randomly drawn permutation π is applied before we get to see and select the candidates. This problem is known as the *secretary problem*. To simplify the argument, we assume that all values are distinct, i.e., $v_i \neq v_j$ for $i \neq j$.

1 Threshold Algorithm

If we can take advantage of the random order then it is reasonable to first observe the sequence a bit and then later use these observations to estimate how good the newly arriving candidates are in comparison to the entire sequence.

In particular, let us consider the following *threshold algorithm*: Observe the first τ elements in the sequence, without selecting any of these. Afterwards, select an element if it is the best one so far.

Theorem 6.2. *For any τ, the threshold algorithm selects the maximum-weight element with probability*

$$\sum_{t=\tau+1}^{n} \frac{1}{n} \frac{\tau}{t-1} \ .$$

Proof. Without loss of generality, let $v_1 > v_2 > \ldots > v_n$. By this definition, the step in which the maximum-weight element arrives is given as $\pi(1)$ and so on.

Observe that the algorithm succeeds if $\pi(1) > \tau$ and no other element is picked before that round.

$$\mathbf{Pr}\left[\text{select best}\right] = \sum_{t=\tau+1}^{n} \mathbf{Pr}\left[\pi(1) = t, \text{no element is picked before round } t\right] \ .$$

Let $S_t \subseteq [n]$ be the set of elements that arrive before round t. Among these elements, $\min S_t$ is the one with highest value.[1] Observe that no element is picked before round t if and only if $\pi(\min S_t) \leq \tau$. This gives us

$$\mathbf{Pr}\left[\text{select best}\right] = \sum_{t=\tau+1}^{n} \mathbf{Pr}\left[\pi(1) = t\right] \mathbf{Pr}\left[\pi(\min S_t) \leq \tau \mid \pi(1) = t\right] \ .$$

It is clear that $\mathbf{Pr}\left[\pi(1) = t\right] = \frac{1}{n}$ but what is $\mathbf{Pr}\left[\pi(\max S_t) \leq \tau \mid \pi(1) = t\right]$? By conditioning on $\pi(1) = t$, the set S_t is a uniformly random subset of size $t-1$ drawn from $n-1$ possible elements. Each possible outcome, gives us a minimum. And this minimum is within the first τ rounds with probability $\frac{\tau}{t-1}$. Very formally, we can write this as

$$\begin{aligned}
&\mathbf{Pr}\left[\pi(\min S_t) \leq \tau \mid \pi(1) = t\right] \\
&= \sum_{M \subseteq \{2,\ldots,n\}} \mathbf{Pr}\left[S_t = M, \pi(\min M) \leq \tau \mid \pi(1) = t\right] \\
&= \sum_{M \subseteq \{2,\ldots,n\}} \mathbf{Pr}\left[S_t = M, \mid \pi(1) = t\right] \mathbf{Pr}\left[\pi(\min M) \leq \tau \mid S_t = M, \pi(1) = t\right] \\
&= \sum_{M \subseteq \{2,\ldots,n\}} \mathbf{Pr}\left[S_t = M, \mid \pi(1) = t\right] \frac{\tau}{t-1} \\
&= \frac{\tau}{t-1} \ .
\end{aligned}$$

Note that in this argument it is very crucial that if you condition on $M = S_t$ you have only fixed which elements arrive in rounds $1, \ldots, t-1$ but not their mutual order.

Overall, we now get

$$\mathbf{Pr}\left[\text{select best}\right] = \sum_{t=\tau+1}^{n} \frac{1}{n} \frac{\tau}{t-1} \ .$$

□

Observe that we can approximate the sum by an integral

$$\sum_{t=\tau+1}^{n} \frac{1}{n} \frac{\tau}{t-1} \geq \frac{\tau}{n} \int_{\tau}^{n} \frac{1}{x} \mathrm{d}x = \frac{\tau}{n} \ln\left(\frac{n}{\tau}\right) \ .$$

Now setting $\tau = \lfloor \frac{n}{\mathrm{e}} \rfloor$, gives $\frac{\tau}{n} \ln\left(\frac{n}{\tau}\right) \geq \frac{\frac{n}{\mathrm{e}}-1}{n} \ln\left(\frac{n}{\frac{n}{\mathrm{e}}}\right) = \frac{1}{\mathrm{e}} - \frac{1}{n}$.

Corollary 6.3. *There is an algorithm that selects the maximum-weight element with probability at least* $\frac{1}{\mathrm{e}} - \frac{1}{n}$.

2 The Optimal Algorithm

After this positive result, we would like to understand whether there is something better that we could do. We will only consider the case of pairwise comparisons and our goal will be to maximize the probability that we select the best candidate. We will show that indeed the algorithm with the highest probability is indeed a threshold algorithm.

[1] The use of the minimum can be slightly confusing here. It is because smaller indices mean higher values.

Theorem 6.4. *For any n, any algorithm that only uses pairwise comparisons selects the maximum-weight element with probability at most*

$$\max_{\tau\in\{0,1,\dots,n\}} \sum_{t=\tau+1}^{n} \frac{1}{n}\frac{\tau}{t-1} \ .$$

In order to capture the setting of pairwise comparisons, let $R_t \in \{1,\dots,t\}$ be the *relative rank* of the candidate arriving in step t compared to the once that arrived before. That is, $R_t = 1$ means that it is the best so far, $R_t = 2$ that it is the second best, and so forth, up to $R_t = t$, which means that all other candidates up to this point were better.

It is not difficult to see that such rank vectors are in one-to-one correspondence with permutations. Drawing the permutation uniformly, we assume that all R_t are independent, R_t is drawn uniformly from $\{1,\dots,t\}$. We select the best candidate if we stop the sequence at t such that $R_t = 1$ and $R_{t'} > 1$ for $t' > t$. The advantage of this notation is that, in step t, the algorithm knows exactly $R_1,\dots,R_t$ but not $R_{t+1},\dots,R_n$.

Let us get acquainted to this notation with a simple calculation that we need later on.

Lemma 6.5. *For all t, we have*

$$\mathbf{Pr}\left[R_{t+1} > 1, R_{t+2} > 1, \dots, R_n > 1\right] = \frac{t}{n} \ .$$

Proof. We can argue in two ways. On the one hand, the event $R_{t+1} > 1, R_{t+2} > 1, \dots, R_n > 1$ means that the best candidate arrives by round t. As we know, the probability for this is $\frac{t}{n}$.

On the other hand, we can also consider the permutation being constructed piece-wise. Some candidates have arrived by t. Now, we draw how the candidate in steps t, $t+1$, and so on compare to these existing candidates. We have

$$\begin{aligned}\mathbf{Pr}\left[R_{t+1} > 1, R_{t+2} > 1, \dots, R_n > 1\right] &= \mathbf{Pr}\left[R_{t+1} > 1\right] \cdot \mathbf{Pr}\left[R_{t+2} > 1\right] \cdot \ldots \cdot \mathbf{Pr}\left[R_n > 1\right] \\ &= \frac{t}{t+1} \cdot \frac{t+1}{t+2} \cdot \ldots \cdot \frac{n-1}{n} = \frac{t}{n} \ .\end{aligned}$$

□

Given $t \in \{1,\dots,n\}$, consider the class of online algorithms that never make any selections in the first $t-1$ steps. Let $\mathcal{A}_t$ be the algorithm with the highest probability of success among them. Let p_t be its success probability.

We would like to understand $\mathcal{A}_1$ and p_1. To this end, let us first consider $t = n$. Clearly, such an algorithm can only win if $R_n = 1$. Therefore $p_n = \frac{1}{n}$. One possible choice for $\mathcal{A}_n$ is to select in step n if and only if $R_n = 1$.

Next, we consider $t = n-1$. If $R_{n-1} > 1$, then the algorithm should better reject the candidate in step $n-1$ and move forward. It only wins if $R_n = 1$ if this case. Otherwise, if $R_{n-1} = 1$, it does have a choice. It can select the current candidate, in which case it only loses if then $R_n = 1$, or it can reject it, in which case it only wins if $R_n = 1$. Note that $\mathbf{Pr}\left[R_n = 1\right] = \frac{1}{n} \le \frac{1}{2}$, so accepting is a better choice. Therefore

$$\begin{aligned}p_{n-1} &= \mathbf{Pr}\left[R_{n-1} > 1\right]\mathbf{Pr}\left[R_n = 1\right] + \mathbf{Pr}\left[R_{n-1} = 1\right]\mathbf{Pr}\left[R_n > 1\right] \\ &= \left(1 - \frac{1}{n-1}\right)\frac{1}{n} + \frac{1}{n-1}\left(1 - \frac{1}{n}\right) \ .\end{aligned}$$

Let us now move to an arbitrary t. In the case $R_t > 1$, we have to reject because there is no chance of winning if we accept. In the case $R_t = 1$, we can accept, then we win if and only if $R_{t'} > 1$ for all $t' > t$. This happens with probability $\frac{t}{n}$ by Lemma 2. If we reject, then we have

to follow algorithm $\mathcal{A}_{t+1}$ because we cannot change previous decisions or random outcomes. So, then we win with probability p_{t+1}. We choose the better of these two options, so

$$\begin{aligned} p_t &= \mathbf{Pr}\left[R_t > 1\right] \cdot p_{t+1} + \mathbf{Pr}\left[R_{t+1} = 1\right] \cdot \max\left\{\frac{t}{n}, p_{t+1}\right\} \\ &= \left(1 - \frac{1}{t}\right) p_{t+1} + \frac{1}{t} \max\left\{\frac{t}{n}, p_{t+1}\right\} \quad . \end{aligned}$$

Observe that $p_t \geq p_{t+1}$, so $(p_t)_{t\in\{1,\ldots,n\}}$ is non-decreasing. In contrast, $\frac{t}{n} < \frac{t+1}{n}$. So, if $p_{t+1} \leq \frac{t}{n}$, then $p_{t+2} < \frac{t+1}{n}$. So, we know that there has to be a $\tau \in \{1, \ldots, n\}$ such that

$$\begin{aligned} p_t &= \begin{cases} \left(1 - \frac{1}{t}\right) p_{t+1} + \frac{1}{t} p_{t+1} & \text{if } t \leq \tau \\ \left(1 - \frac{1}{t}\right) p_{t+1} + \frac{1}{t}\frac{t}{n} & \text{otherwise} \end{cases} \\ &= \begin{cases} p_{t+1} & \text{if } t \leq \tau \\ \left(1 - \frac{1}{t}\right) p_{t+1} + \frac{1}{n} & \text{otherwise} \end{cases} \end{aligned}$$

We can solve this recursion easily

$$p_1 = p_{\tau+1} = \frac{1}{n} + \frac{\tau}{\tau+1} p_{\tau+2} = \frac{1}{n} + \frac{\tau}{\tau+1}\frac{1}{n} + \frac{\tau}{\tau+1}\frac{\tau+1}{\tau+2} p_{\tau+3} = \ldots = \sum_{t=\tau+1}^{n} \frac{1}{n}\frac{\tau}{t-1} \quad .$$