4810-1183 Approximation and Online Algorithms with Applications
Lecture Note 8: Online Algorithm for Secretary Problem

Problem Definition

Let us consider the situation when we have a number of persons applying for a secretary position at our company. We will interview each applicants one by one, and, after the interview, we will give each applicant his/her score. We want to select an applicant with the largest score. However, we have to tell all applicants if we will hire them immediately after the interview. We cannot continue the interviews after we accept to hire an applicant, and we cannot hire someone whom we have rejected before.

We will call this problem as the "secretary problem". Its optimization model is as follows:

Input:	Score of each applicant $s_1, \dots, s_n$
Output:	Decision if we select each applicant $d_1, \dots, d_n \in \{\text{Yes}, \text{No}\}$
Constraint:	1) only one d_i is Yes. 2) d_i is decided based on only $s_1, \dots, s_i$
Objective Function:	Maximize the score of the selected participant

On last week, we have introduce an online strategy for the ski rental problem. The strategy is 2-competitive, that is $SOL_I \leq 2 \cdot OPT_I$ for all possible input I. As the secretary problem is a maximization problem, if we work on the competitive ratio, the competitive ratio α must be smaller than 1. A strategy is α-competitive if $SOL_I \geq \alpha \cdot OPT_I$ for all possible input I.

However, as discussed in the previous lecture note, the competitive ratio is not the only way to justify an online algorithm. We will introduce another justification in this lecture note. That is "the probability that we will have an optimal solution".

Let us now assume that the interview order of the applicants is random. When we have 3 applicants, there are 6 possible orders:

1. $s_1 > s_2 > s_3$,
2. $s_1 > s_3 > s_2$,
3. $s_2 > s_1 > s_3$,
4. $s_2 > s_3 > s_1$,
5. $s_3 > s_1 > s_2$,
6. $s_3 > s_2 > s_1$.

We assume that the probability of having each of the above order is $1/6$. When we have n applicants, we will have $n!$ orders and the probability of having each order is $1/n!$.

When the goal of online algorithms is "the probability of having an optimal solution", the assumption in the previous paragraph is usually essential. We usually can find some probabilistic distribution on our input that makes the probability very small.

Strategy

By the assumption in the previous section, if we always choose the first applicants. The score of our selected participant is s_1. When the number of applicants is 3, s_1 is the largest value with probability $\frac{1}{6}(\text{order } 1) + \frac{1}{6}(\text{order } 2) = \frac{1}{3}$. When the number of applicants is n, we will have the optimal value with probability $1/n$, which might be too less.

Let us consider the following strategy:

```
1: maxValue = 0
2: For i = 1 to n:
3:    if i <= n/e:
4:        d[i] = false
5:        maxValue = max(s[i], maxValue)
6:    else if i = n:
7:        d[i] = true
8:    else if s[i] > maxValue:
9:        d[i] = true
10:       d[j] = false for all j > i
11:       break
```

We will not select the first n/e applicants regardless of their scores. They are chosen to be a benchmark of our selection. We will find the maximum among their scores, and, if a score of a later applicants is larger than the maximum, we will select his/her. If none of the later applicants has a larger score, we will choose the last applicant.

Let now work on the case of 3 applicants. There, $\frac{n}{e} = \frac{3}{e} \approx 1$. We will never select the first applicant, and will use his/her score s_1 as a benchmark. Recall the 6 cases in the previous page. The score of the second applicants, s_2, is larger than the benchmark s_1, in case 3, 4, and 6. We choose the second applicant on that case, and we choose the third applicant on the others. We will have the largest score in case 3, 4, and 5, which make the probability of having an optimal solution be $\frac{1}{2}$. This increases from that of the trivial strategy, $\frac{1}{3}$.

Analysis

We have shown that the success probability is $1/2$, when $n = 3$. In this section, we will show that, for any n, the success probability is no less than $\frac{1}{e} \approx 0.36$.

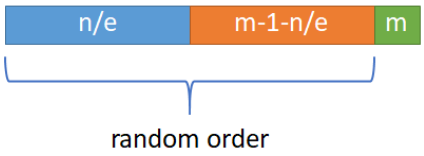
With probability $1/e$, the best applicant is among the first n/e applicants. Because the selected participants will not be in the first n/e applicants, the best participant will not be selected. Our strategy will fail to find the best applicant in this case.

Now, let us suppose that the best participant is the m^{th} participant such that $m > n/e$. There are $m - 1$ participants before the participant. Consider the figure in the following page. We will not select the m^{th} participant when there is some applications in the orange zone have a higher score than all the first n/e aplicants.

Let us consider the first two zones (blue and orange) as a subarray of the array of all scores. If the largest score in the subarray is in the first zone, we will never select an applicant from the orange zone. We will interview the m^{th} participant, and, as he/she has a higher score than everyone in the first zone, he/she will be selected. Our algorithm can find an optimal solution, which is the m^{th} participant.

On the other hand, if the largest score in the subarray is in the second zone, an applicant with the largest score or someone in the orange zone will be selected. Our algorithm will fail to find an optimal solution.

When we have a random order, a subarray of the order will be also random. Because of that, the order of the $m - 1$ predecessors is also random. The probability that we have the largest in the subarray in the first n/e, which is also the success probability, is $(\frac{n}{e})/(m - 1)$.



We know that the success probability equal $P_1 + \cdots + P_n$ when P_i is the probability that the best applicant is the i^{th} applicant and he/she is selected. It is clear from the definition that $P_i = 0$ for $i \leq n/e$. For $i > n/e$, the probability of having m equal a particular number i is $1/n$, and, by the previous paragraph, the success probability when $m = i$ is $\left(\frac{n}{e}\right)/(i-1)$. Thus, we know that the success probability is

$$\sum_{i=\frac{n}{e}+1}^{n} \frac{1}{n} \cdot \frac{\frac{n}{e}}{i-1} = \frac{1}{e} \sum_{i=\frac{n}{e}+1}^{n} \frac{1}{i-1} = \frac{1}{e} \sum_{i=\frac{n}{e}}^{n-1} \frac{1}{i} \approx \frac{1}{e} \int_{\frac{n}{e}}^{n} \frac{1}{i} di = \frac{1}{e} \left[\ln n - \ln \frac{n}{e}\right] = \frac{1}{e} [\ln n - (\ln n - \ln e)]$$
$$= \frac{1}{e}.$$

Online Navigation Summaries

Let us suppose that we have a drone that will take a video which consist of a million of images, and we want to find the meaning of the video by a machine learning algorithm. It is usually very hard to make sense of a large number of images, so we want to select the most meaningful images out of them. We call the problem "navigation summaries". In the following footage, we may select the image highlighted in blue. They tell us that the video were taken at a place with a bicycle and a gate.

When a video is as long as hours and it is broadcasted at a social network, users might want to have a part of summary before the end of the broadcasting. Also, a drone might want to place a sensor at important places to watch the change at there. It would be more efficient if the drone can place the sensor just after a photo is taken.

The authors of [2] propose a way to measure an "importance" of each picture. They assume that we will select at most k images from the video. By that, the optimization model for the online navigation summaries is as follows:

Input:	Importance of each image $s_1, \ldots, s_n$ Number of selected images k
Output:	Decision if we select each image $d_1, \ldots, d_n \in \{Yes, No\}$
Constraint:	1) The number of selected images is k 2) For all selected images i, s_i is one of the k largest score. 3) d_i is decided based only on $s_1, \ldots, s_i$
Objective Function:	None

We want to satisfy the second constraint with as high probability as possible. The authors devise an algorithm based on the strategy in the following page.

```
1: threshold = 0
2: For i = 1 to n:
3:    if i <= r:
4:        d[i] = false
5:        threshold = max(s[i], maxValue)
6:    else if i = n - k + (number of selected sensors) + 1:
7:        d[i] = true
8:    else if s[i] > maxValue:
9:        d[i] = true
10:       if we have already selected k images:
11:          d[j] = false for all j > i
12:          break
```

The only difference between the first and the second algorithm is: we do not stop after an image is selected, but wait until we have selected k images. We previously set r to n/e, and, actually, the value maximize the success probability. We are going to calculate at the following paragraph the best value for r in this application.

Let us consider the figure at the previous page again. For each $r < m \leq n$, we will calculate the probability that:

1. s_m is one of the top k score;
2. before m, $k-1$ correct images are selected;
3. m is selected in the above algorithm.

By the random order, the probability of having 1) is k/n. Then, for 2), by our algorithm, we must have k – 1 images of the top k in the second zone. The probability of having that is $\binom{m-1-r}{k-1}/\binom{n}{k-1}$. For 3), we must be sure that other than the $k-1$ images in 2), the next largest score in the first and the second zone must be in the first zone. Otherwise, we will should that next best image. That is, the k^{th} largest score in the first $m-1$ images is in the orange zone. With the assumption that the score order is random, we know that the probability is $r/(m-1)$.

To conclude, the probability that we have m satisfying the three conditions is $\frac{k}{n} \cdot \frac{\binom{m-1-r}{k-1}}{\binom{n}{k-1}} \cdot \frac{r}{m-1}$.

The probability that having some m satisfying the conditions is then equal to

$$\sum_{m=r+1}^{n} \left[\frac{k}{n} \cdot \frac{\binom{m-1-r}{k-1}}{\binom{n}{k-1}} \cdot \frac{r}{m-1} \right].$$

By experiment, the authors show that the value of r that maximizes the probability is $\frac{n}{ke^{\frac{1}{k}}}$.

In practice, if we just aim to choose the maximum scores, we might have a lot of images with the same object. The authors of the paper have a heuristic to avoid that. They do not around 2 selected images to be closer than a specific time frame.

References

[1] T. S. Ferguson, "*Who solved the secretary problem*?", Statistical Science, Vol. 4, No. 3, pp. 282-289, 1989.

[2] Y. Girdhar, G. Dudek, "*Online navigation summaries*", Proceeding of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010), pp. 2035-2040, 2010.