Advanced Algorithms

Lecture 11: Introduction to Online Algorithms

Lecturer: Nathan Klein

1 Introduction

The focus of this lecture is answering the question: how do we maintain a good solution to a problem when the *input is not given to us ahead of time?*

1.1 The Secretary Problem

Suppose there are two players. One player puts n pieces of paper in a box with a different number written on each piece. The second, only knowing n, randomly pulls out pieces of paper from the box and looks at them. After seeing a number, the second player can either pick this number and stop, or discard it permanently and keep trying to find a higher number. Now, the question is:

Question: can the second player always pick the largest number in the box with constant probability? (Say at least 1%?)

While at first this may look impossible, it turns out the answer is yes!

Lemma 1.1. *There is a strategy which picks the highest number with probability at least* $\frac{1}{4}$.

Proof. The strategy is as follows. Take out $\frac{n}{2}$ pieces of paper at random (suppose n is even for simplicity), and record the highest number v^* seen in the first half. Then, pick the first number you see with value greater than v^*.

This strategy will pick the highest number if the second higher number is in the first half and the highest number is in the second half. The probability the second highest candidate is in the first half, and the highest number is in the second half, is the probability that in a random permutation of $[n]$, 2 appears in the first half and 1 appears in the second half. This probability of this is a little more than $\frac{1}{4}$ (in particular, it's $\frac{1}{2} \cdot \frac{n/2}{n-1} > \frac{1}{4}$). □

This is called the secretary problem, because usually in the problem setup you are hiring a secretary, and you have n candidates you will interview in a random order. When you interview a candidate, you see their score and have to hire them on the spot or reject them permanently. You then want to hire the candidate with the highest score with some probability.[1]

It turns out you can do better than $\frac{1}{4}$: you can achieve a probability of $\frac{1}{e} \approx 0.367$.

Lemma 1.2. *There is a strategy which picks the highest number with probability* $\frac{1}{e}$.

Proof. Same idea here, but let's try to pick the best possible time to start looking for the highest number. Knowing n, say you don't allow yourself to pick anything earlier than the kth one you see.

[1] I prefer the slips of paper setup. For some reason, it feels more surprising that there is a strategy that succeeds with constant probability here, even though it's exactly the same problem.

Then,

$$\begin{aligned}\mathbb{P}\left[\text{success}\right] &= \sum_{i=k}^{n} \mathbb{P}\left[i \text{ picked}, i \text{ highest}\right] \\ &= \sum_{i=k}^{n} \mathbb{P}\left[i \text{ picked} \mid i \text{ highest}\right] \mathbb{P}\left[i \text{ highest}\right] \\ &= \frac{1}{n} \sum_{i=k}^{n} \mathbb{P}\left[\text{highest of first } i-1 \text{ is in first } k-1 \mid i \text{ highest}\right]\end{aligned}$$

But this probability is just $\frac{k-1}{i-1}$, so we get

$$\frac{1}{n} \sum_{i=k}^{n} \frac{k-1}{i-1} = \frac{k-1}{n} \sum_{i=k}^{n} \frac{1}{i-1}$$

We can choose k to maximize this. Let's say $k = \delta n$ for some $\delta \in [0,1]$. Then, this quantity is essentially

$$\delta \sum_{i=\delta n}^{n} \frac{1}{i} = \delta(H_n - H_{\delta n}) \approx \delta(\ln n - \ln(\delta n)) = \delta \ln \frac{1}{\delta} = -\delta \ln \delta$$

To maximize this expression, take its derivative with respect to δ, which is $-\ln \delta - 1$. Setting this to 0, we get $\delta = \frac{1}{e}$. This gives a success probability of $\frac{1}{e} \ln(e) = \frac{1}{e}$.

So, our algorithm is: wait until you have seen $\frac{n}{e}$ of the numbers (rounding up or down), and then pick the first number that's higher than the best you've seen so far. Some slightly more careful calculations reveal that the success probability is always better than $\frac{1}{e}$, but tends to $\frac{1}{e}$ as $n \to \infty$. □

It turns out that this is also the best you can do. You can formally show that really, the only algorithm is: "if the kth number is the largest I have seen so so far, and I have seen sufficiently many candidates, then I will select this number." But our analysis is exactly picking the best stopping time, so $\frac{1}{e}$ is the best probability you can get.

1.2 Bipartite Matching

Let's dive into a slightly more intricate real-world example of online algorithms: a ride-hailing app. Here, suppose that the set of drivers is known to us and given a ride request, we need to quickly decide with which driver to serve that request.

To formalize this model, suppose we have a bipartite graph consisting of drivers on one side and clients on the other. Now, the clients will arrive one by one in some order. When a client arrives, we see where they are located, and generate a list of drivers that are sufficiently close to them. In other words, a node v on the client side of the graph is revealed together with the edges in $\delta(v)$. We then need to irrevocably choose a driver to match them with.

Our goal will be to make sure that we serve as many clients as possible. To simplify the model, we assume that the driver list is fixed and once a driver serves a client, they cannot serve another client. (This is a reasonable model for finding a good assignment in a fixed length time window.) We call the drivers the **offline** nodes and the clients the **online** nodes.

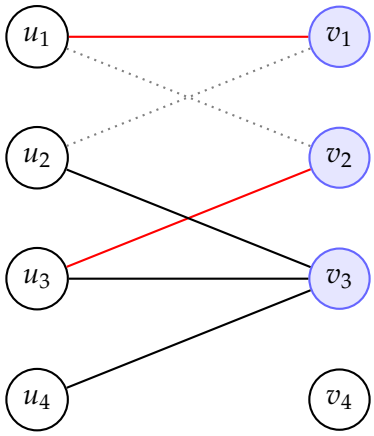

Figure 1: On the left are the offline nodes, and on the right the online nodes. At timestep 3, the edges adjacent to v_3 are revealed. We have already matched v_1 and v_2 (in solid red), and need to decide whether to match v_3 to u_2 or u_4 (we cannot match it to u_3, as this has already been taken). The edges adjacent to v_4 are still unknown.

Definition 1.3 (Online Bipartite Matching)**.** *An instance $\mathcal{I}$ of online bipartite matching consists of a bipartite graph $G = (\textsc{Off} \uplus \textsc{On}, E)$ and an ordering $v_1, \ldots, v_{|\textsc{On}|}$ of the online nodes so that node v_i arrives at timestep i. At each timestep i, the algorithm must decide what node in $\textsc{Off}$ to match $v_i \in \textsc{On}$ to, if any, using only the knowledge of the set of nodes $\textsc{Off}$ and the edges $\bigcup_{j=1}^{i} \delta(v_i)$.*

To understand the performance of an algorithm for this problem (we call this the "competitive ratio"), we will compare it to a powerful adversary: an algorithm that knows the entire sequence of clients and their edges up front. In other words, we compare to the optimal solution given the whole input.

Definition 1.4 (Competitive Ratio of an Online Algorithm)**.** *Given a (possibly randomized) algorithm $\mathcal{A}$ for an online problem, the competitive ratio $\alpha \leq 1$ of $\mathcal{A}$ is defined as the **worst-case ratio**, over all instances $\mathcal{I}$, of the expected value of $\mathcal{A}(\mathcal{I})$ divided by the value of the optimum solution of the offline problem.*

So, if $\mathcal{A}(\mathcal{I})$ is the set of edges in the matching returned by $\mathcal{A}$ on instance $\mathcal{I}$ and $M(\mathcal{I})$ is the edges in the maximum matching in $\mathcal{I}$, then the competitive ratio of $\mathcal{A}$ is:

$$\alpha = \inf_{\mathcal{I}} \frac{\mathbb{E}\left[|\mathcal{A}(\mathcal{I})|\right]}{|M(\mathcal{I})|}$$

where the expectation is over the randomness in the algorithm $\mathcal{A}$.

2 The Greedy Algorithm

First, we will analyze the most natural algorithm for this problem. When an online node arrives, **select an arbitrary available offline node to match it to** (if one exists).

The result is a *maximal* matching. A matching is maximal if no edge can be added to it. It is well known that any maximal matching is always at least *half* the size of the maximum matching.

Lemma 2.1. *Every maximal matching is at least half the size of a maximum matching.*

Proof. Let M be the edges in a maximal matching and O the edges of a maximum matching. Since M is maximal, no edge of O can be added to M so that M remains a matching. It follows that every edge $(u, v) \in O$ has the property that either u is matched in M or v is matched in M. Since O is a matching, this implies that M has at least $|O|$ matched vertices. The number of matched vertices in M is exactly $2|M|$. Therefore, $2|M| \geq |O|$, or $|M| \geq \frac{1}{2}|O|$, as desired. □

This is tight, as there are graphs where a maximal matching has size half that of a maximum matching, e.g. in Fig. 2.

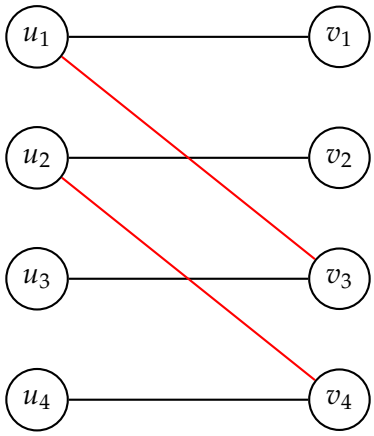

Figure 2: In red is a maximum matching of size half that of the maximum matching (in black).

Homework Preview: A natural way to try to improve this algorithm is to choose a **random** neighbor to match an incoming online vertex to. You will show in your homework that this does not improve the performance to more than $\frac{1}{2} + o(1)$.

3 Primal Dual Analysis

A popular framework for analyzing online algorithms is called *primal dual*. Here, we construct a linear programming relaxation of the problem (the **primal**) and simultaneously construct a feasible solution to its **dual**.

Our solution to the primal is going to be integral: we'll just give an actual matching. The dual solution will serve as a lower bound on the optimal solution.

3.1 Primal and Dual Formulations

We can write the following linear program for the maximum matching problem.

$$\begin{aligned} \max \sum_{e \in E} x_e & \\ \sum_{e \in \delta(v)} x_e \leq 1 & \quad \forall v \in V \\ x_e \geq 0 & \quad \forall e \in E \end{aligned} \tag{1}$$

From the second homework, we know that when the underlying graph is bipartite, this has an integrality gap of 1.[2] Now, let's write the dual formulation. In the dual, we want to find a *combination of the constraints*, with the constraint for $v \in V$ having multiplier y_v, so that the LHS is greater than the objective function $\sum_{e \in E} x_e$ of the primal, and the RHS is as small as possible.

$$\begin{aligned} \min \sum_{v \in V} y_v & \\ y_u + y_v \geq 1 \quad & \forall \{u, v\} \in E \\ y_v \geq 0 \quad & \forall v \in V \end{aligned} \tag{2}$$

To gain some intuition about the dual, let's construct one for the following graph. Since this is a

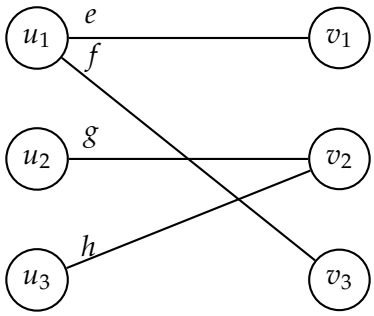

Figure 3: An example graph.

small example, it should be easy to see the maximum matching has size 2. But how would we prove it using the dual?

We can set $y_{u_1} = 1$, $y_{v_2} = 1$. This is feasible for the dual since every edge is adjacent to either u_1 or v_2, and it has objective value 2. So, this is a lower bound. But what are we really doing? We're just saying: take one copy of the constraint $x_e + x_f \leq 1$, and one copy of the constraint $x_g + x_h \leq 1$. This reads:

$$x_e + x_f + x_g + x_h \leq 2$$

But this is exactly the objective function, so it proves that the objective is at most 2. In general:

Fact 3.1. *Let y be a feasible solution to (2). Then, the optimal matching has size at most $\sum_{y \in V} y_v$.*

Proof. Let $y \in \mathbb{R}^V$ be a feasible dual solution. Consider summing the constraints in the primal to form an equation, where for each v we take y_v of the constraint $\sum_{e \in \delta(v)} x_e \leq 1$. This equation will read:

$$\sum_{e=\{u,v\} \in E} (y_u + y_v) x_e \leq \sum_{v \in V} y_v$$

But since $0 \leq y_u + y_v \leq 1$ for all $\{u, v\} \in E$ and $x_e \geq 0$ for all $e \in E$, we have:

$$OPT \leq \sum_{e \in E} x_e \leq \sum_{e=\{u,v\} \in E} (y_u + y_v) x_e \leq \sum_{v \in V} y_v$$

where OPT is the size of the optimal matching, and we have used in the first inequality that (1) is a relaxation of the problem. □

[2] When the graph is not bipartite, it has an integrality gap of $\frac{3}{2}$.

3.2 Analyzing the Greedy Algorithm

Let's prove again that the greedy algorithm is 2-competitive using the primal-dual analysis framework.

Whenever an online node v arrives, if it can be matched, pick an arbitrary edge $\{u, v\}$, add it to the matching, and set $y_u = y_v = \frac{1}{2}$.

Let M be the matching produced by the greedy algorithm. Clearly, at the end of the algorithm $|M| = \sum_{v \in V} y_v$. We will now show the following, which implies that greedy is a 2-approximation.

Lemma 3.2. *$2y$ is dual feasible.*

Proof. Suppose by way of contradiction that $2y_u + 2y_v < 1$ for some edge $e = \{u, v\}$, where v is online and u is offline. It follows that $y_u = y_v = 0$, because $y_v \in \{0, \frac{1}{2}\}$ by construction. But this means that when v arrived, it was not matched. Contradiction, since it would have used edge e as u was available. □

Therefore, we have a 2-approximation, since our matching M has $|M| = \sum_{v \in V} y_v \geq \frac{1}{2} OPT$.

3.3 Next Time: Improving upon Greedy

Karp, Vazirani, and Vazirani [KVV90] showed that you can do better than $\frac{1}{2}$ using what they call the *ranking* algorithm: they showed a $1 - \frac{1}{e}$ competitive algorithm.[3]

Their algorithm is simple. Instead of randomizing over the ranking of the *edges*, randomize over the *offline vertices*. Give each vertex v a random label ℓ_v, drawing each label independently and uniformly at random from $[0, 1]$. We may assume there are no ties, i.e. $\ell_u \neq \ell_v$ whenever $u \neq v$.

Now, when an online node arrives, match it to an available offline node (if one exists) with the *smallest label possible.* And that's it!

References

[GM08] Gagan Goel and Aranyak Mehta. "Online budgeted matching in random input models with applications to Adwords". In: *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '08. San Francisco, California: Society for Industrial and Applied Mathematics, 2008, 982–991 (cit. on p. 6).

[KVV90] R.M. Karp, U.V. Vazirani, and V.V. Vazirani. "An optimal algorithm for online bipartite matching". In: *STOC*. 1990 (cit. on p. 6).

[3] Their original paper had an error which was fixed later by [GM08].