A Course in Graduate Algorithms **Lecture 7**

Online Algorithms

Designed by Prof. Bo Waggoner for the University of Colorado-Boulder *Updated: 2023*

In an *online* setting, the algorithm operates interactively over time. It takes a sequence of inputs or pieces of information and, at each round, must make a decision or produce some output. Previous decisions typically cannot be revoked.

The first question that arises is how to evaluate such an algorithm. Since it cannot know the future, we don't expect it to perform perfectly; but we would like to understand and prove when an algorithm is "good". The idea is to show its performance is comparable to an optimal algorithm that knew the future in advance. We will begin with an example.

Objectives:

- Learn intuitively what an online algorithm is and know some examples.
- Know the definition of *competitive ratio* for maximization and minimization versions of online problems.
- Know how to apply greedy algorithms to some online problems and prove competitive ratios.

1 The Ski Rental Problem

A skier goes on a trip of unknown duration with a group. The first day, the skier must choose whether to rent skis (cost 1) or purchase skis (cost 10). After the first day, the skier learns whether the group decides the trip is over, or whether it will continue another day.

Each successive day, if the skier has already purchased skis, she can re-use them; if not, she has the option to either rent again or purchase. Then when that day is over, she learns whether the trip has ended or it will continue.

The skier doesn't know when it will end, but would like to spend as little money as possible. We can compare the amount she spends to the amount spent by an optimal algorithm that knows the entire trip in advance and knows how many days it will last.

Optimal offline algorithm. The first step for online algorithms is often to figure out what the optimal offline algorithm is. In this case, if we know the trip will last 10 days or fewer, it is optimal to rent skis every day. If we know it will last 10 days or longer, it is optimal to immediately buy skis on the first day. (If it lasts exactly 10 days, both approaches are optimal.)

A 2-competitive algorithm. Once the skis are purchased, there are no more decisions to make. So this problem boils down to renting the skis for some number of days, then (if the trip has not yet ended), purchasing.

Suppose the skier rents until the 10th day, then buys. If the trip lasts up to 10 days, this is optimal. Now suppose it lasts any amount longer than 10 days. Then the skier spends 20, while OPT spends 10. So the skier never spends more than twice the optimal amount. We call this a competitive ratio of 2.

General purchase cost. In general, if renting costs 1 and purchasing costs $K > 1$, we can give a 2-approximation or 2-*competitive* algorithm: rent for $\lfloor K \rfloor$ days, then purchase the next day. If the trip lasts $\lfloor K \rfloor$ days or fewer, this is optimal. Otherwise, the algorithm spends $\lfloor K \rfloor + K$, while the offline optimal is to spend just K (by purchasing immediately), for a ratio of

$$\frac{\lfloor K \rfloor + K}{K} \leq 2.$$

Exercise 1. Suppose I try to do better by purchasing earlier, say, after $\lfloor 0.5K \rfloor$ days. What is my worst-case ratio approximately? *Hint: On what day should the trip end to make me maximally regret choosing to purchase early?*

Remark (Optional)**.** In general, one can show that 2 is the best achievable factor. More precisely, there is no $C < 2$ such that we can always guarantee to spend less than $C \cdot \text{OPT}$, where OPT is the offline optimal. To see this, we let the purchase price K be an integer with $K \to \infty$ and consider an instance where, the day the algorithm decides to purchase, the trip ends. (If the algorithm never purchases, its competitive ratio is already unbounded.) In this case, the algorithm spends $K + d - 1$ if it rents for $d - 1$ days and then purchases, while $\text{OPT} = \min\{d, K\}$. So the competitive ratio is at least $\frac{K+d-1}{\min\{d,K\}}$. By cases, if $d \leq K - 1$ then this is at least $\frac{K-1+K-1}{K-1} = 2$, and similarly it also exceeds 2 if $d \geq K + 1$. If $d = K$, then this is $2 - \frac{1}{K} \to 2$ as $K \to \infty$, so no ratio better than 2 is possible.

However, it turns out that randomized algorithms can do a bit better than 2, on average. The idea is that if we use some randomness to decide exactly when to buy, then the worst-case scenario (see Exercise 1) is unlikely to happen exactly.

Exercise 2. Suppose renting costs R while purchasing continues to cost K, with $K > R > 0$. Now what policy is 2-competitive? *Hint: If you spend more than K total on renting and then spend K more to buy, this won't be 2-competitive (why?).*

2 Competitive Analysis

To evaluate online algorithms, we use the idea of *competitive analysis*: compare the algorithm's outputs to what would have been the optimal choices, had we known the entire future sequence of inputs in advance. As with approximation algorithms more generally, we will often take the ratio of the performance of the online algorithm's solution, compared to the optimal performance on these inputs.

In a minimization problem such as ski rental, we say the algorithm is C**-competitive** if it guarantees a C-approximation to the offline optimum, i.e. $\text{ALG} \leq C \cdot \text{OPT}$ for all instances of the problem. For maximization problems, we may say the algorithm is α**-competitive** if it guarantees an α approximation ratio, i.e. $\text{ALG} \geq \alpha \cdot \text{OPT}$ for all instances.

(For this class, we will not be too picky about terminology, but it is useful to know that "competitive analysis" typically refers specifically to online problems, while approximation ratios are more general.)

3 Online Bipartite Matching

In this classic problem, the input is a bipartite graph $G = (U_1, U_2, E)$ that arrives over time. The graph is unweighted and undirected.

Initially, the algorithm is given U_2, the set of "offline" vertices. It does not know any edges. It begins with an empty matching M.

Exercise 3. Recall: what is the definition of a matching, precisely?

Then, vertices $u \in U_1$ arrive one by one. When each arrives, all of its incident edges (u, v) are revealed. The algorithm may add up to one of these edges to M, provided that it remains a matching. In particular, if it adds (u, v) to M, then the offline vertex v cannot already be in the matching.

Then the next vertex u' arrives, and so on.

The goal is to maximize the size of the final matching M. Notice that the algorithm cannot remove edges from M once they are added, nor can it go back and add edges of vertices that previously arrived.

This problem models many real-world scenarios, at an abstract level. Suppose we are allocating items (or tasks) to people who arrive one at a time. Each person has a set of items they are compatible with

(or tasks they are qualified for); these represent edges in the graph. When each person u arrives, we try to assign them to an available item v that has not yet been matched. We hope to maximize the total number of items assigned over the course of the day.

In online advertising, companies such as Google and Microsoft have had active research in this problem as a model for assigning advertisers to slots. Each time a person loads a page, this is modeled as a vertex arrival. There are edges to all the advertisers who are possible matches for that page (e.g. athletic companies on a sports page); one of them must be selected to place and advertisement on this page. In this simple model, each advertiser is matched to one page per day; more sophisticated models take into account larger budgets.

3.1 Positive results

Maybe not surprisingly, we will study the **Greedy** algorithm:

- When vertex u arrives, take an arbitrary available edge (u, v), if any exists.

By "available" we mean that v is not already in the matching M.

If we look carefully, we can see that this is actually an implementation of the greedy algorithm for *offline* bipartite matching! Why? Because as the vertices of the graph arrive, the algorithm ends up iterating through all the edges of the graph, adding each edge it can to its matching. The order of the edges just happens to be chosen externally. Therefore, **we have already proven its competitive ratio is** 0.5**.** We will next give a different version of the proof for students who are interested.

Theorem 1. *Greedy for online bipartite matching has a competitive ratio of* 0.5*, i.e. for every input instance,* $|M| \geq 0.5|M^*|$ *where* M^* *is a maximum matching.*

Proof. Let us build a function f from M to subsets of M^*. For each edge $e \in M$, define $f(e)$ to be the set of all edges in M^* that share a vertex with e.

First, we claim that for each $e' \in M^*$, there is some $e \in M$ such that $e' \in f(e)$. That is, each edge in the optimal matching overlaps with at least one of Greedy's edges. Letting $e' = (u, v)$ be in the optimal matching, this follows because when u arrived, either v was already matched by Greedy, or else v is available and in this case u is definitely matched (possibly to v, but at least to some vertex).

Second, we claim $|f(e)| \leq 2$, i.e. e overlaps with at most two edges in M^*. This follows because e has two endpoints, and since M^* is a matching, each appears in at most one edge of M^*.

These claims imply implies $|M^*| \leq 2|M|$. To be very formal, the first claim implies $M^* = \bigcup_{e \in M} f(e)$. This implies

$$\begin{aligned} |M^*| &\leq \sum_{e \in M} |f(e)| \\ &\leq \sum_{e \in M} 2 && \text{(second claim)} \\ &= 2|M|. \end{aligned}$$

□

3.2 Negative (impossibility) results

Now, we will show impossibility results for online bipartite matching. For normal, non-online algorithms, one generally shows negative results by showing e.g. that the problem is NP-hard, or giving other evidence that a very slow or sophisticated algorithm is required to solve it.

But for online algorithms, we can show impossibilities for *all* algorithms, unconditionally. In other words, we will show that no matter how much computation time one has (indeed, even if one can solve the halting problem), one cannot beat a competitive ratio of $\frac{1}{2}$ with any deterministic algorithm.

The reason is that when the online algorithm is making its choices, it simply does not have enough information about the future to choose optimally, no matter how much computation it does.

Theorem 2. *No deterministic*[1] *algorithm has a competitive ratio of better than* $\frac{1}{2}$.

Proof. Let $U_1 = \{u_1, u_2\}$ and let $U_2 = \{v_1, v2\}$. Consider any algorithm. We consider several instances and show the algorithm has ratio to OPT of at most half on one of the instances.

Instance A has edge (u_2, v_1) and (u_1, v_1) and that is all. The algorithm has three choices when v_1 arrives: match the first edge, match the second edge, or make no match. If it makes no matches, its ratio to OPT on this instance is zero.

So suppose it makes a match on instance A. Without loss of generality, it matches edge (u_2, v_1). (If it chooses the other edge, everything that follows can be modified symmetrically.)

So consider instance B with edges $(u_2, v_1), (u_1, v_1), (u_2, v_2)$. The algorithm matches (u_2, v_1) at round one when v_1 arrives. Then, when v_2 arrives, it has no legal options. So its matching has size one, but the offline optimal has size two. So its competitive ratio is at most 0.5. □

Now let us consider online *weighted* bipartite matching. This is the same problem, but each time an edge arrives, the algorithm also learns the weight on that edge. The goal is to maximize the total weight of the matching, and the offline benchmark is the maximum weighted bipartite matching.

Theorem 3. *For this online weighted bipartite matching problem, no deterministic algorithm can guarantee a competitive ratio of* ϵ*, for any* $\epsilon > 0$.

Proof. Let $w_{uv} = \epsilon$ and $w_{uv'} = 1$. We have one instance with just edge w_{uv}, and one instance with both edges arriving in this order. If the algorithm does not take the first arriving edge, then its competitive ratio on the first instance is zero. But if it does, its ratio on the second instance is ϵ. □

Here we see a large contrast in difficulty of the online problems, even though offline, both the unweighted and weighted algorithms had greedy $\frac{1}{2}$ approximation ratios. However, one can formulate versions of the weighted problem where constant competitive ratios are possible. For example, one allows "free disposal" (deleting edges from M later on) or "budgets" on the vertices (allowing them to match multiple times up to some total weight).

Exercise 4. Consider the weighted online matching problem, but suppose that all edge weights are between $\frac{1}{2}$ and 1. Argue that the Greedy algorithm (which completely ignores edge weights and matches any available edge) gets a $\frac{1}{4}$ competitive ratio.

Hint: What can you guarantee about the number of edges in the algorithm's matching compared to OPT*? Now what is the worst case for the edge weights in the algorithm versus* OPT*?*

4 Online Bin Packing

In the *online bin packing* problem, items of various weights $w_j \in [0, 1]$ arrive and must be packed into bins, each of which can hold total weight 1. The goal is to use the fewest possible bins.

Specifically, in each round $j = 1, \ldots, T$, the algorithm is given w_j. The algorithm then selects a bin $i \geq 0$ to place item j into.

Let B be the number of bins that the algorithm used, and B^* the optimal (minimum) number of bins used by an optimal algorithm that knows all arrivals in advance.

Consider this simple **Greedy** algorithm: as each item arrives, place it in any currently-used bin where it fits. If none, place it in a new bin. We will prove this algorithm guarantees a 2-approximation to the minimum possible number of bins.

Some notation for analysis: Let $W = \sum_{j=1}^{T} w_j$, the total weight of all items. Let $S[i]$ be the set of items assigned to bin i. The total weight in bin i is $L[i] = \sum_{j \in S[i]} w_j$.

Lemma 1. $B^* \geq W$.

[1] An algorithm is *deterministic* if it always behaves the same on the same input.

Proof. Assume for contradiction that $B^* < W$. Then the average load of the bins in the optimal solution is $\frac{1}{B^*}W > 1$. If the average load is strictly larger than 1, then at least one bin has load larger than 1, which is an invalid solution – a contradiction. □

Lemma 2. *After running Greedy, all of the nonempty bins are at least half full, except at most one.*

Proof. Suppose we have one current bin that's less than half full, and an item arrives, but we don't put it in the bin. Then the item must have weight at least 0.5. So whichever bin it is placed in becomes at least half full. So we can never have multiple bins that are less than half full. □

Lemma 3. *Greedy satisfies $B - 1 < 2W$.*

Proof. There are at least $B-1$ bins that are at least half full, by Lemma 2. So the total weight in those $B-1$ bins alone is $\frac{1}{2}(B-1)$. Furthermore, the remaining bin has some nonzero weight in it (otherwise we would not have used it). So W is strictly larger than $\frac{1}{2}(B-1)$. □

Theorem 4. *Greedy is 2-competitive.*

Proof. By Lemma 1 and Lemma 3, $B - 1 < 2W \leq 2B^*$. Note that B and B^* are integers. If we have $a - 1 < b$ for integers a, b, then we have $a \leq b$. So $B - 1 < 2B^*$ implies $B \leq 2B^*$, as claimed. □

Remark. Even the offline version of this problem, where all item weights are given, is NP-hard to solve optimally. So achieving a factor of 2 in the online setting is quite nice. And the factor can be improved, to 1.7 and below, by more sophisticated algorithms. This problem has seen extensive study for 50 years.

Exercise 5. Suppose every item has weight 0.51. How do Greedy and OPT compare?

Exercise 6. Sketch an argument that Theorem 4 is almost tight, i.e. Greedy can do roughly as bad as twice OPT. *Hint: you will only need items of size* 0.5 *and, say,* 0.01.