Harvard CS 1360, Spring 2025
Instructor: Ariel Procaccia
Scribe: Jessica Chen

Matching 1: Online Matching Algorithms

Lecture 13

Example 1 (Online Algorithms)**.** Consider you're on a ski vacation, and you can buy skis for $\$B$ or rent for \$1 per day. Each day, the weather is either sunny or rainy, and because you are spoiled, you will go home when it's rainy. You consider the question: should you rent or buy when $B = 5$?

If you already know the weather in each of the coming days, then the decision is very easy: If there are $> B$ days of sun, then we buy; if there are $< B$ days of sun, then we rent.

However, what if we don't know in advance how many days of sun there will be? Every day, you will need to decide whether to rent or buy, because you don't know if the next day will be sunny or rainy. This scenario is in the **online setting**: in each time period, you find out more about a part of the input.

Consider this online algorithm: Rent skis for B days, then buy. This algorithm performs very suboptimally where there are 6 days of sun: after spending \$5 for 5 days of renting, you buy skis for another \$5 for a total cost of \$10, only to be struck by rain the next day. In contrast, this algorithm performs optimally when there are 4 days of sun: you rent the skis for \$4 for 4 days.

Now, we've begun to think about the question: How suboptimal can an online algorithm be, compared to the optimal solution? This relation sounds familiar to our previous approximation ratios. In the online setting, we refer to this relation as the **competitive ratio**.

Definition 1 (Competitive Ratio)**.** *The competitive ratio of an online algorithm is the worst-case ratio between the algorithm and the offline optimum.*

Note that, in contrast to our approximation ratios, we are comparing against the offline optimum here. The difficulty stems from the lack of information in the online setting. If we are working with a maximization problem, this ratio is typically less than 1. In minimization problems, the competitive ratio is typically more than 1.

Example 2. *What is the competitive ratio of the "rent for B days, then buy" algorithm?*

As in the prior example, the worst-case occurs when there are $B + 1$ days of sunshine, as we must pay $2B$ for renting on the first B days and buying on the last day. In the optimal case, you would have simply bought the skis for \$B on the first day. Therefore,

$$\text{Competitive ratio} = \frac{ALG}{OPT} = \frac{2B}{B} = 2$$

Note: You can actually construct a slightly "better" algorithm where you rent for $B - 1$ days and then buy. This algorithm is $\frac{2B-1}{B}$-competitive. Still, the claim holds that it is impossible for a deterministic algorithm to arrive at a lower competitive ratio than this.

Theorem 1. *No online ski rental algorithm has a lower competitive ratio than $\frac{2B-1}{B}$.*

Proof. The only decision you can make is when to buy, which we denote with $K + 1$. In other words, you rent for K days and buy on day $K + 1$.

To find the competitive ratio, we always think of the worst-case instance; here (and in literature), we refer to this worst-case construction as the **adversary**. Imagine an adversary always constructs a scenario as bad as possible for the algorithm. Here, the worst adversary will make it rain as soon as you buy, or on day $K + 2$.

Now, let us analyze how bad the performance of the algorithm can be, depending on the value of K. If $K \geq B$, then the optimal solution would just be to buy on day 1, so $OPT(1) = B$, while the algorithm would rent for K days before buying, so $ALG(1) = K + B \geq 2B$. The ratio is then ≥ 2. Intuitively, this makes sense, as when there are more days of sun, then you should have bought on the first day; as there are more days of sun, the worse you are off compared to the optimal.

If $K \leq B - 2$, then the optimal is to have rented the whole time, so $OPT(1) = K + 1$. In contrast, the algorithm would've still rented for the K days before buying, so $ALG(1) = K + B \geq 2K + 2$. In this case, the ratio is $\geq \frac{2K+2}{K+1} \geq 2$. □

Example 3 (Display Advertising)**.** Display advertising is the largest online matching problem in the world. For example, imagine the ads on the banner above the Wall Street Journal. We're matching advertisers to impressions: each person that shows up to the website is an impression, and the advertisers have preferences for a particular target audience to view the ad. We can think of this problem as a bipartite graph between advertisers and impressions. In this simplified case, we have an between an advertiser and an impression, where the impression is of a type that the advertiser specified as acceptable.

Notably, these impressions arrive online. We don't know in advance which impressions will show up, and when they arrive, we must choose an ad to display to them.

We start with this simple model, and will later discuss generalizations.

Definition 2 (The Simplest Model)**.** Consider bipartite graph $G = (U, V, E)$, where U, V are the two sides and E is the edges. U is offline, while V is online. In other words, the $|U| = n$ offline vertices are known, while the V vertices will eventually arrive, along with there incident edges. Online vertices can only be matched upon arrival.

Objective: Maximize the size of matching.

We say ALG has competitive ratio $\alpha \leq 1$ if for every graph G and every input order π of V,

$$\frac{ALG(G, \pi)}{OPT(G)} \geq \alpha$$

Example 4 (The Simplest Model)**.** Below, we illustrate how the simplest model works with online vertices a, b, c arriving in some order. Each vertex has edges associated with valid potential matches. The blue edges denote the matching that the algorithm makes; here, we do not specify an algorithm, and are only illustrating the decision mechanism. See Figures 1 and 2 for two step-by-step examples.

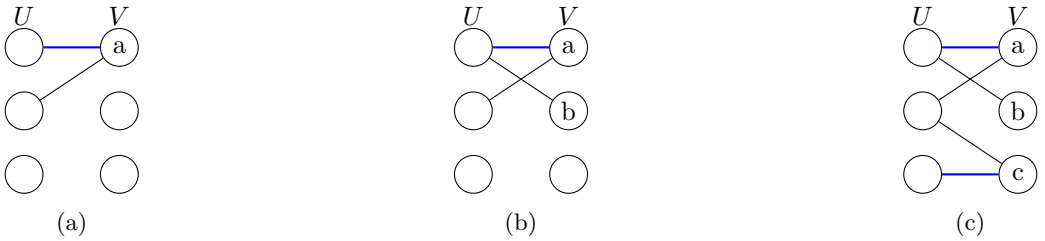

Figure 1: The simplest model. When vertex a arrives, it can match with u_1, u_2; the arbitrary algorithm matches a with u_1. When vertex b arrives, no match can be made, as the only valid match u_1 has already been matched with a. When vertex c arrives, the arbitrary algorithm matches it with u_3.

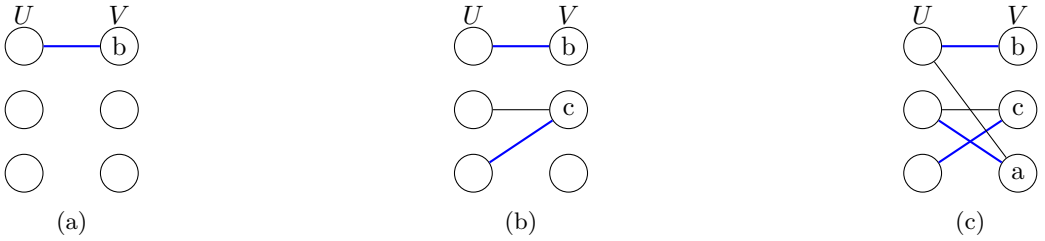

Figure 2: The simplest model. Arrival order: b, c, a. The algorithm greedily matches b to u_1, c to u_3, and then finds an augmenting path to match a to u_2 by rematching c.

Definition 3 (Greedy Algorithm)**.** When a vertex arrives, match it to an arbitrary unmatched neighbor, if one exists.

Now, we will analyze the lower and upper bounds of this Greedy algorithm.

Lower Bound. The optimal matching consists of disjoint vertices. Consider some edge in this matching that wasn't selected by the greedy algorithm. There are two possible reasons as to why the edge was not selected:

1. The online vertex was matched with some other offline vertex.

2. We could not match the online vertex because all its offline neighbors were already matched.

In other words, for every edge in the optimal solution, at least one of the two matched vertices in that edge is matched by the GREEDY algorithm. Therefore, GREEDY has a competitive ratio (lower bound) of $\alpha \geq \frac{1}{2}$.

We call this type of result, where you cannot add any additional edges to the matching without taking out some edges, *inclusion maximal matching*. For any inclusion maximal matching, you can apply the same argument and say it gives a lower bound of $\frac{1}{2}$.

Upper Bound. *We observe that the competitive ratio of any deterministic algorithm is at most* $\frac{1}{2}$. Consider an adversarial online arrival model: no matter what your deterministic algorithm tries to do, the adversary will generate and send in new vertices that are the worst-case for your algorithm. See Figure 3 for examples of the adversary.

(a) (b)

Figure 3: In the left-hand graph, the adversary generates an arriving v_2 that can only match with the already-matched u_1. In the right-hand graph, the adversary generates an arriving v_2 that can only match with the already-matched u_2.

No matter what your deterministic algorithm does in the first step, the adversary ensures that it can only match 1 edge when the optimal solution matches 2 edges. Therefore, $\alpha \leq 2$.

Note: Given some changes in assumptions, you can potentially arrive at a deterministic algorithm that looks at historical data and does better.

Definition 4 (RANDOM Algorithm)**.** Match to an unmatched neighbor uniformly at random.

Here, the adversary has full knowledge of the algorithm. It can know how the algorithm will randomize, and generate a worst-case accordingly. However, it does not see the realization of the randomness beforehand. In other words, it can know that the algorithm is planning to coin flip, but it doesn't see the coin flip results beforehand.

On Figure 3, RANDOM achieves an upper bound of $\frac{3}{4}$. The adversary must decide beforehand whether it plans to introduce a vertex v_2 as on the left-hand graph or the right-hand graph (with edge to u_1 or u_2, respectively). Therefore, RANDOM can get 1 match half the time, and 2 matches the other half, and the ratio is then $\frac{(1+2)/2}{2} = \frac{3}{4}$.

Example 5 (Competitive ratio of RANDOM on Figure 4)**.** Consider the graph in Figure 4. Let's call $\frac{n}{2} = k$. We refer to the vertex directly across the bipartite graph as the "desired" vertex, since such a match would not block matches for the bottom k vertices. For the top k online vertices (right), they have a $\frac{1}{k+1}$ chance of being matched to the "desired" vertex (based on the edges), and a $\frac{k}{k+1}$ chance of being matched to the bottom k vertices on the left. For a large k, this converges to these top k online vertices matching to a bottom k offline vertex with probability close to 1.

As such, we can say that approximately only the top k online vertices will have a successful match. Some vertices may luckily match with their "desired" neighbor, but the probability is much smaller. Therefore, the competitive ratio is about $\frac{1}{2}$.

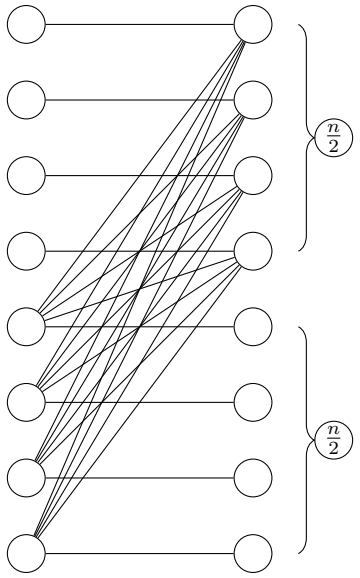

Figure 4: The top half of online vertices (right) can be matched with either a "desired" offline vertex directly to its left, or any of the bottom half of the left vertices. The bottom half of the online vertices can only be matched directly across.

Definition 5 (RANKING Algorithm)**.**

1. Choose a random permutation $\pi : U \to [n]$. In other words, we assign a random priority over the offline vertices.
2. Match each online vertex as it arrives to its unmatched neighbor u with the lowest $\pi(u)$, or its highest-priority offline neighbor.

Example 6 (Competitive ratio of RANKING on Figure 5)**.**

Intuitively, we can see RANKING would do better than RANDOM as it is not just independently randomizing on every vertex upon arrival, but instead correlating the randomness across all offline vertices. For each "undesired" matching the algorithm makes, there is a higher probability that the remaining available offline vertices have a higher priority.

Theorem 2. *The competitive ratio of* RANKING *is* $1 - \frac{1}{e} \approx 0.63$*, and this is the best possible. No randomized algorithm can do better on this online matching problem.*

We will skip the proof for this theorem.

Definition 6 (Weighted Matching)**.** Let us augment the problem with the following features:

1. Each offline vertex u has a budget B_u.
2. Each edge has a weight ("bid"). The goal is to maximize the weight of the matching.

In the advertising setting, the offline vertices are advertisers with budgets, and they may pay differently for different profiles of impressions.

A GREEDY algorithm matches the arriving vertex to the highest weight edge, subject to the budget.

Theorem 3. GREEDY *algorithm for weighted matching has a competitive ratio of* $\frac{1}{2}$*.*

In Figure 6, we sketch out that, even if the weights are much smaller than B, the competitive ratio still doesn't get better than $\frac{1}{2}$. In other words, the upper bound is $\frac{1}{2}$ for the case of small bids.

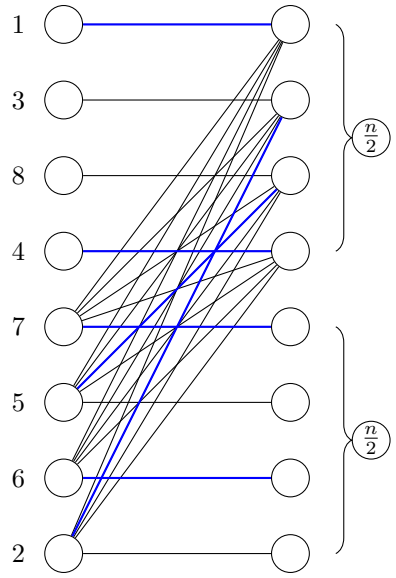

Figure 5: The left-hand offline vertices have been randomly assigned a rank. As the right-hand online vertices arrive one by one, the RANKING algorithm chooses matchings according to the highest-priority (lowest) rank (blue edges).

Clearly, we should take into account the remaining budget of the advertisers. However, solely allocating based on the remaining budget is also a bad idea. Consider the example in Figure 7.

As such, we should consider an algorithm that takes into account both the edge weights and the remaining budgets.

Definition 7 (MSVV Algorithm)**.** Denote by x_u the fraction of u's budget that has been spent. We also define a function $f(x) = 1 - e^{x-1}$, and the weight of an edge w_{uv}. Each vertex v is matched with a vertex u that maximizes $w_{uv} \cdot f(x_u)$.

In other words, we take into account both the remaining budget of u as well as the weight of the edge.

Theorem 4 (Competitive ratio of MSVV)**.** MSVV *has a competitive ratio that approaches* $1 - \frac{1}{\epsilon}$ *as the budget grows relative to the size of the bids, and this is the best possible algorithm, even among randomized algorithms.*

Considering we have just shown a deterministic algorithm to be the best even among randomized algorithms, this is a very powerful result.

The MSVV algorithm has very practical applications, and has had a large impact on how companies like Google design their advertising algorithms. It extends to many things that may happen in practice, such as advertisers arriving at different times, bidders paying only for clicks, the winning bidders paying the second-highest bid, etc.

> "The core problem of budget management remains important, and the core idea [of spending budget smoothly] remains impactful." (Aranyak Mehta, Google Research, July 2024)

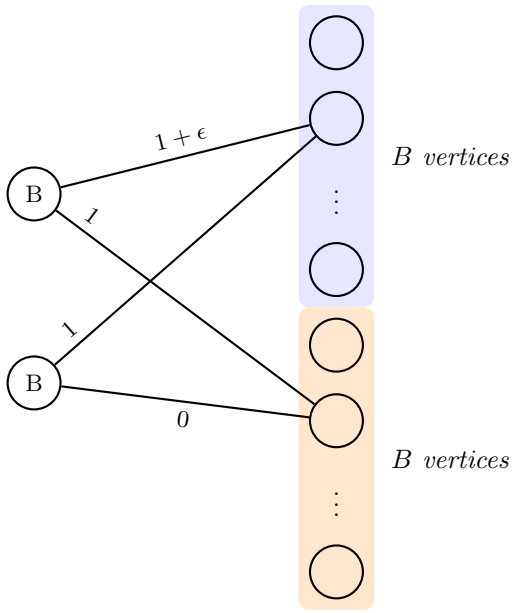

Figure 6: The optimal is to have the blue vertices match with the bottom-left vertex, and the orange vertices to match with the top-left vertex. However, based on the GREEDY algorithm, the blue vertices will prioritize the $1+\epsilon$ edge over the 1-weighted edge, thus leaving the orange vertices to match with the 0-weighted edges. As such, the competitive ratio is approximately $\frac{1}{2}$.

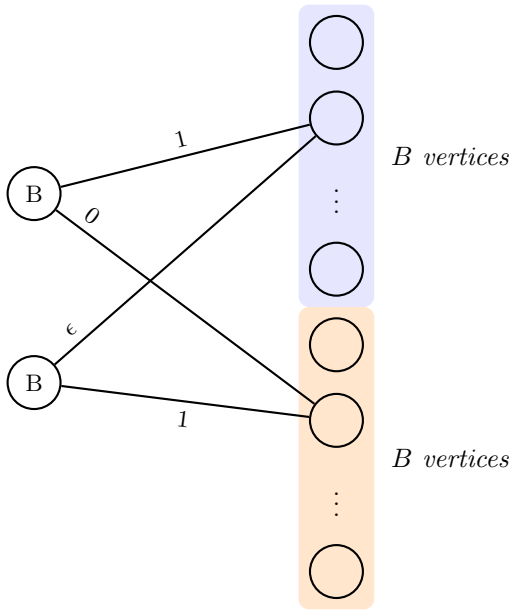

Figure 7: Assume $\epsilon \ll 1$. After the first blue vertex is matched with the top advertiser, the remaining blue vertices will choose the ϵ edge since the bottom advertiser has greater remaining budget. The orange vertices can all match with their desired bottom advertiser, but notice that we have essentially "wasted" most of the blue vertices.