Harvard CS 1360, Spring 2026
Instructor: Ariel Procaccia
Scribe: Jessica Chen, Jakob de Raaij

Online Matching Algorithms

—

Lecture 13

In the last lectures, we have studied the fair division of goods through the lens of proportionality and envy-freeness. We now shift to a related problem of finding *matchings*, where each 'person' is supposed to receive exactly one (or a predefined number of) 'items'. Examples of this setting include matching students to schools, doctors to hospitals, leaders to followers in ballroom dancing, or website users to ads.

1 Online Algorithms

In this lecture we'll focus on *online* matching, in which information about the problem is revealed sequentially. To do so, let's first introduce *online algorithms* and the type of guarantees we will be looking to obtain. We start by looking at a classical online problem.

Definition 1 (Ski Rental Problem)**.** You're on a ski vacation and have two options: Either, you buy skis for $\$B$ and use them for all future days, or you can rent skis, for \$1 per day. Each day the weather is either sunny or rainy and (because you are spoiled) you will go home on the first rainy day. If you don't know what the weather is going to be, should you rent or buy?

If you know a reliable weather report for all of the coming days, the decision is easy: If there are $\geq B$ days of sun, you buy skis; else, you rent. However, what if we don't know in advance how many days of sun there will be? Now, every day, you need to decide again whether to buy skis or rent for the day. This is an example of the *online setting*: In each time step, you find out about a new part of the input and then need to make an irreversible decision (here, to buy or to rent on this day), before seeing the next part of the input.

Let's now consider the following online algorithm: On the first B days of skiing, you rent. If you are still skiing on the $(B+1)$th day, you finally buy skis. This algorithm performs particularly badly when there are exactly $B+1$ days of sun: After spending $\$B$ for the B days of renting, you buy skis for another $\$B$ for a total cost of $\$2B$, only to be struck by rain the next day and go home. In contrast, this algorithm performs optimally when there are at most B days of sun since we rent skis for at most $\$B$, less than if we had bought skis.

Indeed, there is no strategy that will never make us spend more money than necessary. If we decide to buy skis, it could always rain the next day, in which case we would have been better off renting. At the same time, if we never buy skis but always rent, we will eventually pay much more than if we had bought skis initially if there is a long sequence of sunny days. However, intuitively, some purchasing strategies are more suboptimal than others. We'll now try to quantify this.

Definition 2 (Competitive Ratio)**.** The *competitive ratio* of an online algorithm is the worst-case ratio between the value achieved by the algorithm and the optimal value possible with full knowledge of the input.

The notion of a competitive ratio is very similar to the notion of an approximation ratio and the notions of Price of Anarchy and Price of Stability we saw earlier in this class. The competitive ratio tells us how close our *online* algorithm gets to the optimum that would have been achievable *offline*, i.e., if we were able to 'see the future' (e.g., knew ahead of time how many sunny days are ahead of us). The approximation ratio tells us how close our bounded-complexity (e.g., polynomial time) algorithm gets to the optimum that would have been achievable with unlimited resources (e.g., infinite computation time). The Price of Anarchy/Stability tells us how close an equilibrium outcome (restricted to players being selfish) gets to the optimum that would have been achievable with cooperation (e.g., the social optimum).

Similarly to the approximation ratio and PoA/PoS, the competitive ratio in a maximization problem is always at most 1 and in minimization problem is always at least 1 — we can never do better than the optimum. [1]

[1] As with approximation ratio and PoA/PoS, beware that some sources may refer to what we call a c-competitive algorithm here as a $1/c$-

So let's return to the ski rental problem and calculate the ratio of our 'algorithm' to rent for B days and then buy. We know that our algorithm is optimal for the first B days since we don't buy skis, so never pay more than necessary. However, our algorithm may become suboptimal once we buy skis. In particular, the worst-case occurs if it starts raining the day after we bought skis. In this case of there being exactly $B + 1$ days of sunshine, we pay a total of $2B$ for renting on the first B days and then buying on the last day. In the optimal case, you would have simply bought the skis for \$B on the first day. Therefore,

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

It turns out that we can actually construct a slightly better algorithm if we rent for $B - 1$ days (instead of B) and then buy. By the same logic as above, this algorithm is $\frac{2B-1}{B}$-competitive, which is slightly better than 2-competitive. It turns out that this is optimal.

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

Proof. There actually aren't many possible ski rental algorithms. In particular, the only decision we can make is when to buy. We denote the day on which we buy as $K + 1$. In other words, we rent for K days and then buy on day $K + 1$ (so never need to rent again).

To find the competitive ratio we need to search for worst-case instance. It is a helpful (and common) technique to think of an *adversary* who, seeing what our algorithm does at each timestep, picks the worst-possible piece of input in the next timestep. This adversary always constructs an input scenario that is as bad as possible for the given algorithm. Thus, here, the adversary will make it rain the day after we buy, so on day $K + 2$.

Now, let us analyze how bad the performance of the algorithm will be on this worst-case instance, depending on the value of K: If $K \geq B$, we know it is going to be sunny for more than B days, so the optimal solution would be to buy on the first day, giving $OPT = B$. The algorithm would rent for K days before buying, so $ALG = K + B \geq 2B$. Thus, the competitive ratio is at least 2. If $K \leq B - 2$, we know it is going to be sunny for at most B days, so the optimal is to rent the whole time, giving $OPT = K + 1$. In contrast, our algorithm would have rented for K days before buying, so $ALG = K + B \geq 2K + 2$. The competitive ratio is at least $\frac{2K+2}{K+1} \geq 2$.

Thus, if $K \neq B - 1$, the competitive ratio is always at least 2, so the competitive ratio of $\frac{2B-1}{B} < 2$ we get from $K = B - 1$ is optimal. □

2 Online Matching

Now that we have defined the online setting, we can study the matching problem in this setting. Our motivating example will be advertising on the internet:

Example 1 (Display Advertising)**.** Every time a user enters a website with a designated spot for an ad, the website can decide which advertiser gets to show their ad. This is known as display advertising and arguably is the largest online matching problem in the world. The goal is to match advertisers to impressions: Each person that visits the website is an impression and the advertisers have preferences (e.g., target audiences) over impressions. Notably, these impressions arrive online: We don't know in advance which people will show up on the homepage (though we may have estimates in practice) but upon arrival of a person, we must choose the ad to display to them (without knowledge of the future people).

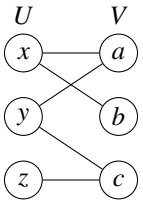
We can think of display advertising as a bipartite graph between advertisers and impressions. In the most simple case, we may assume that there is an edge between an advertiser and an impression if the impression is of a type that the advertiser specified as acceptable, e.g. , if the person is in their target audience. Furthermore, for now, we will assume that each website only wants a single user (in their target audience) to see their ad.

Definition 3 (Matching)**.** Given a bipartite graph $G = (U, V, E)$ where U, V are the two sides of vertices and E are the edges, a matching $M \subseteq E$ is a set of edges such that every vertex $u \in U$ and $v \in V$ is incident to at most one edge in M.

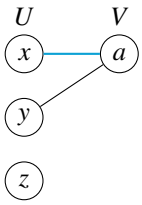
competitive algorithm. Since we know that the approximation constant c is at most 1 for a maximization problem and at least 1 for a minimization problem, we can usually infer from the context what competitive guarantee the authors refer to. This ambiguity is in fact so widespread that it, regrettably, also appeared on problem 1 of the midterm.

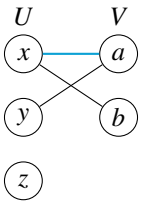
Definition 4 (Online Matching, the Simplest Model)**.** Given is a bipartite graph $G = (U, V, E)$. Initially, we are only given the vertices in U, while V and E are revealed online: The vertices in V with the edges in E incident to them are revealed to us one after another. Whenever a vertex $v \in V$ is revealed to us, we either match it with a still unmatched vertex $u \in U$ that it is connected to (we add edge (u, v) to the matching M), or we leave it forever unmatched. The goal is to match as many vertices as possible, i.e., to arrive at a matching M of size as large as possible.

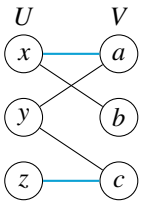
Example 2 (Online Matching, the Simplest Model)**.** Below, we illustrate online matching in this simplest model for the following graph with online vertices $V = \{a, b, c\}$ arriving in some order. The edges of each vertex in V represent the potential matches.



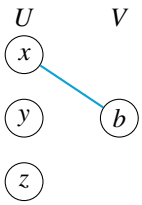
Let's first consider what an (arbitrary) online matching algorithm might do if the vertices in V arrive in the order a, b, c. When vertex a arrives, it can be matched with x, y; let's say our algorithm matches a with x, shown in blue. Thus, $M = \{(a, x)\}$. When vertex b arrives next, no match can be made, since its only valid match x has already been matched with a. Still, $M = \{(a, x)\}$. Indeed, we realize now that matching a with x was suboptimal since it blocks b from being matched, but we can no longer change it so b remains unmatched. When vertex c arrives, our algorithm may match it with z. We arrive at the matching $M = \{(a, x), (c, z)\}$ of size 2.

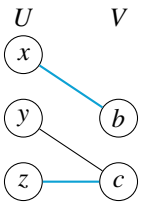


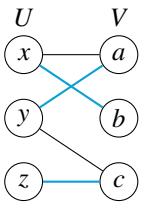




Now, let's consider the case of the vertices arriving in the order b, c, a. First, the algorithm can only match b to x since that's its only potential match. Then, c may get matched to z. Ultimately, when a arrives, the only potential match still possible is y, so we match a to y. We realize now that matching c with z indeed was the right decision: If we matched it with y, a would remain unmatched now. We arrive at the matching $M = \{(b, x), (c, z), (a, y)\}$ of size 3.







We start by analyzing how well we can do by simply matching each vertex arbitrarily as it is revealed to us.

Algorithm 1 Greedy Algorithm for Online Matching

1: When a vertex v arrives, match it to an arbitrary, unmatched $u \in U$ that v is connected to, if one exists.
2: Else, leave v unmatched.

Theorem 2. *The competitive ratio of the greedy algorithm is* $1/2$.

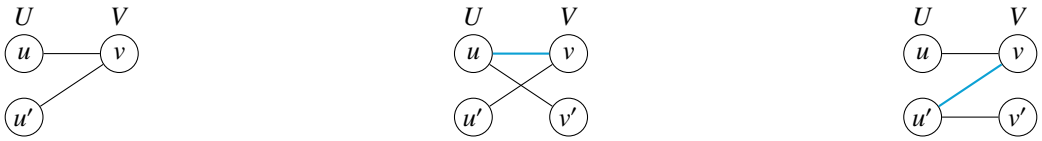
Proof. It is $1/2$*-competitive:* Consider an optimal matching. For any edge (u, v) in the optimal matching that is not in the greedy matching — that is, a pair of a vertex $u \in U$ and a vertex $v \in V$ that are matched to each other in this optimal

matching but are not matched to each other in the greedy matching — there has to be one of two possible explanations: Either, vertex $v \in V$ was matched to another vertex $u' \in U$ in the greedy matching, or when vertex v was to be matched it couldn't be matched, in which case we know that u was already matched to some other $v' \in V$.

Thus, for every pair of vertices (u, v) matched to each other in the optimal solution, at least one of the two vertices is also matched to some vertex in the greedy algorithm. Therefore, the number of matches returned by the greedy algorithm is at least $1/2$ the optimal number of matchings, so its competitive ratio is at least $1/2$. [2]

It is no better than $1/2$-competitive: We try to construct worst-case instances. In particular, consider an adversarial online arrival mode: Observing the matches done by our algorithm, the adversary decides on the edges of the next vertex in the worst possible way. In particular, let's assume that the first vertex, $v \in V$, that we see is connected to two unmatched vertices $u, u' \in U$ as shown on the left.

Now, if our algorithm matches v to u, the adversary sends a vertex v' that has only an edge to u', as shown in the middle. If our algorithm matches v to u', the adversary sends a vertex v' that has only an edge to u, as shown on the right. In either case, we are unable to match v', so our matching has size 1. However, in both cases, the optimal matching has size 2. If the adversary repeatedly sends (copies of) this instance, we will only ever get 1 pair of vertices



matched, while the optimum would be two. The greedy algorithm is no better than $1/2$-competitive. □

The argument that we described above for why the greedy algorithm is no better than $1/2$-competitive, in fact, holds for any deterministic algorithm. Faced with this adversarial instance, no matter what any deterministic algorithm does in the first step, the adversary can ensure that only one edge is in the matching while the optimal matching contains two.

Thus, the greedy algorithm is optimal within the class of all deterministic algorithms for online matching. Can we do better when we allow for randomization?

Algorithm 2 Random Online Matching Algorithm

1: When a vertex v arrives, pick one unmatched $u \in U$ that v is connected to, if one exists, uniformly at random and match it with v.
2: Else, leave v unmatched.

For randomized algorithms, we need to specify how much power we give the adversary. If the adversary is allowed to observe the random choice of our algorithm and then decide on the next vertex $v \in V$ to present to us, randomization does not help us here: In the upper-bound construction seen above, the adversary could observe how we match v and then decide on v' accordingly, giving the same upper bound of $1/2$. Such an adversary is called an *adaptive (online) adversary*.

A slightly weaker adversary model (that is arguably more realistic when interacting with a worst-case, random but not actively adversary environment) is an *oblivious adversary*. In this setting, we assume that the adversary has to decide on the graph G with knowledge of our algorithm but without knowledge of the randomness used by our algorithm. Thus, if our algorithm, in the upper-bound construction above, deterministically matches v to u, the oblivious adversary would know this and could create the graph to be the worst-case response. However, if our algorithm randomly matches v, the oblivious adversary does not know what the outcome of this random choice is when deciding on v'.

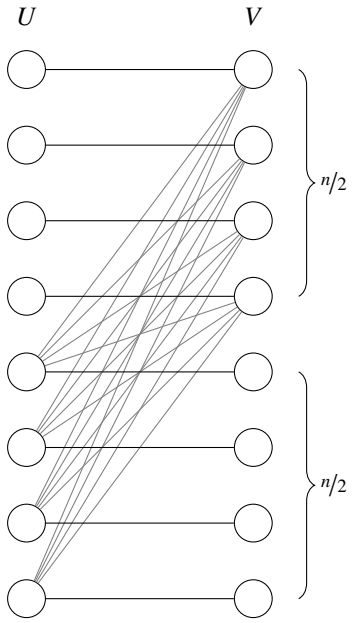
On the upper-bound construction, the random online matching algorithm achieves is $3/4$-competitive, in expectation. The adversary must decide beforehand whether it plans to introduce a vertex v' connected to just u' (as in the middle graph) or a vertex v' connected to just u (as in the right graph). Therefore, the random algorithm gets one match with probability $1/2$ and two matches with probability $1/2$, so in expectation 1.5. This is $3/4$ of the optimum.

[2] A matching to which we cannot add any additional edge (match two new vertices) without taking out some edge (undoing an existing matching) is called an *inclusion maximal matching*. The greedy matching is an inclusion maximal matching. For any inclusion maximal matching, we can apply the same argument to get a lower bound of $1/2$ on the competitive ratio.

While this looks promising, we can see that our random algorithm actually does not outperform the greedy algorithm in the worst case.

Theorem 3. *The competitive ratio of the random online matching algorithm is* $1/2$.

Proof. We can see that the random algorithm is $1/2$-competitive with the exact same argument as for the greedy algorithm. To show that it is no better than $1/2$-competitive, consider the following graph. Each vertex in V is connected to the vertex in U right to its left. Furthermore, every vertex in the top half of V is connected to every vertex in the bottom half of U. The vertices in V are revealed from top to bottom.



For any vertex $v \in V$, we refer to the vertex $u \in U$ directly to its left as its *desired* vertex. If every vertex in the top half of V is matched to its desired vertex, we get the optimal matching of size n. Every vertex in the top half of V that is not matched to its desired vertex blocks one vertex in the bottom half of V from being matched to its desired vertex, thus decreasing the size of the achievable matching by one. In particular, we know that the size of the matching returned by the random algorithm is n, minus the number of vertices in the top half of V that are matched to a vertex in the bottom half of U, i.e., aren't matched to their desired vertex.

When the jth vertex in V from the top is being matched, for $j \in \{1, ..., n/2\}$, at most $j-1$ vertices in the lower half of U are already matched. Thus, their probability of being matched to their desired vertex is at most $\frac{1}{n/2+1-(j-1)}$. By the linearity of expectation, the number of vertices in the top half of V that is matched to their desired vertex is at most

$$\sum_{j=1}^{\frac{n}{2}} \frac{1}{n/2+1-(j-1)} = \sum_{j=1}^{\frac{n}{2}} \frac{1}{j+1} = H_{(n/2)+1} - 1 = O(\log n).$$

In particular, almost all vertices in the top half of V will be matched to a vertex in the lower half of U, except for $O(\log n)$ vertices. Thus, the expected size of a matching returned by our algorithm is $\frac{n}{2} + O(\log n)$, while the optimal matching has size n. This ratio, the competitive ratio, approaches $1/2$ □

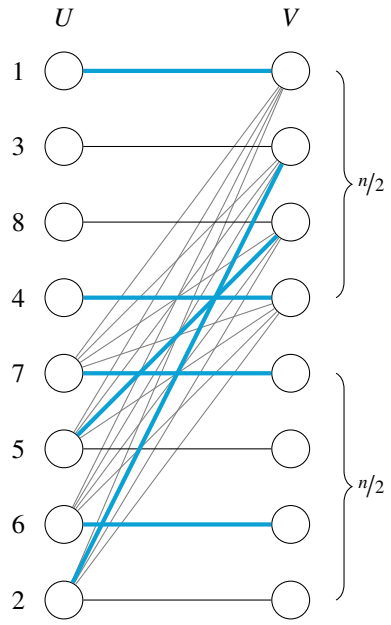
However, a different randomized algorithm can do better than $1/2$.

Algorithm 3 Ranking Algorithm for Online Matching

1: At the beginning, choose a random permutation $\pi : U \to [n]$.
2: When a vertex v arrives, pick the unmatched $u \in U$ with lowest $\pi(u)$ that v is connected to. Match v with this u.
3: If no such u exists, leave v unmatched.

Intuitively, π is a random priority ranking over the offline vertices. Whenever we can choose which unmatched vertex in U to map to a new vertex $v \in V$, we pick the one with the lowest value of π, i.e., highest priority.

As an example, let's consider what this algorithm might do on the previous upper-bound construction for $n = 8$ and the ranking shown below. As the online vertices in V arrive one by one, the ranking algorithm chooses matchings according to the highest-priority (i.e., lowest) rank and matches them, as shown by the blue edges.



While we won't formally prove it, we give some intuition for why the ranking algorithm does better than random algorithm on this instance. The ranking algorithm is not just independently randomizing over all possible $u \in U$ upon arrival of a $v \in V$, but instead correlating the randomness across all offline vertices. For each undesired matching the algorithm makes, a high-priority vertex in the lower half of U disappears. Thus, for any later vertex in the top half of V, there is a higher probability that its desired vertex has a higher priority than all vertices in the lower half of U.

For example, consider the $n/4$th vertex in V arriving and assume that (almost) all vertices in V so far have been matched to undesired vertices, i.e., in the lower half of U. Thus, the $n/4$ vertices of highest priority in U are already matched, so in expectation, the lowest priority unmatched vertex in the lower half of U will have priority $\approx n/2$. However, the expected priority of the desired vertex of the next v is also $\approx n/2$. Thus, informally speaking, this $n/4$th vertex in V will be matched with its desired vertex with probability $\approx 1/2$. Under the random algorithm, in this scenario, the same vertex v would have been matched to its desired vertex with probability $\approx \frac{1}{n/2-n/4} = \frac{4}{n}$, a much smaller probability. This, in expectation, a lot more vertices end up matched to their desired vertex.

Theorem 4. *The competitive ratio of the ranking algorithm is* $1 - 1/e \approx 0.63$. *No randomized algorithm has a large competitive ratio for this online matching problem.*

3 Online Weighted Matching

In our motivating example, matching impressions to ads on the internet, every ad will be seen by many impressions. Thus, we need to relax the assumption that each vertex in U (an ad) is only matched to one vertex in V (an impression).

Definition 5 (Weighted Matching)**.** Given is a bipartite graph $G = (U, V, E)$. Each offline vertex $u \in U$ has a *budget* B_u. Each edge $e \in E$ has a bid b_e. A *weighted matching* $M \subseteq E$ is a set of edges such that

- every vertex $v \in V$ is incident to at most one edge in M, that is, $|u \in U : (u, v) \in M| \leq 1$, and
- for each vertex $u \in U$, the sum of the bids of the edges in M that are incident to u is at most B_u. That is, $\sum_{v \in V:(u,v)\in M} b_{(u,v)} \leq B_u$.

Intuitively, every advertiser $u \in U$ now has a total budget B_u that they want to spend on ads. For each impression $v \in V$ and advertiser $u \in U$ that are connected by an edge, the bid $b_{(u,v)}$ corresponds to the value of advertiser u for impression v. Each impression can be matched to at most one advertiser, but advertisers can be matched to multiple impressions as long as the sum of all their matched bids doesn't exceed their budgets.

Definition 6 (Online Weighted Matching)**.** Given is a bipartite graph $G = (U, V, E)$ with budgets for U and bids for E. The vertices in V with the edges in E incident to them (and the corresponding bids) are revealed to us online, one after another. Whenever a vertex $v \in V$ is revealed to us, we either match it with a vertex $u \in U$ that it is connected to and that has enough budget left to afford its bid $b_{(u,v)}$ (i.e., we add this edge (u, v) to the weighted matching M), or we leave this v forever unmatched. The goal is to maximize the bids of matched edges; that is, we want to maximize $\sum_{e \in M} b_e$.

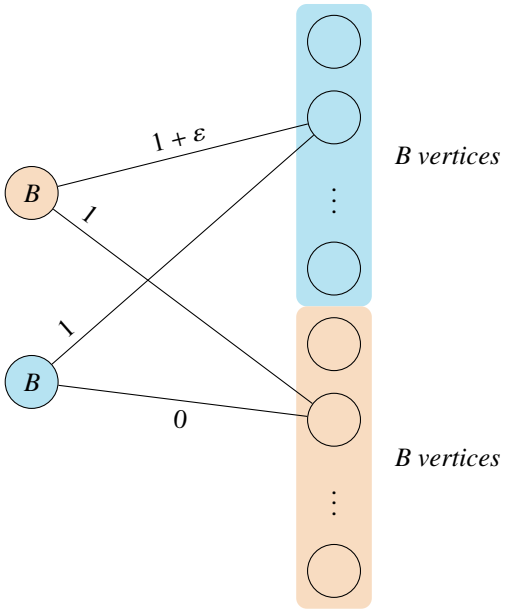
We can again define a greedy algorithm in this setting:

Algorithm 4 Greedy Algorithm for Online Weighted Matching

1: When a vertex v arrives, pick the $u \in U$ that maximizes $b_{(u,v)}$, subject to u having an edge to v and u having enough budget left to afford $b_{(u,v)}$. Match v with this u.
2: If no such u exists, leave v unmatched.

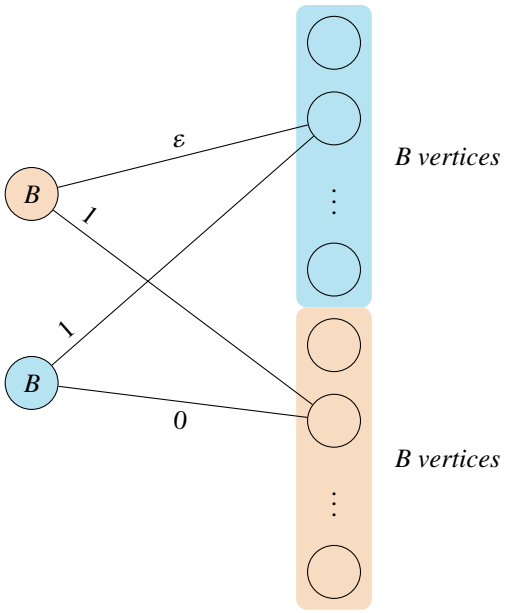
Theorem 5. *The greedy algorithm for online weighted matching has a competitive ratio of* $1/2$.

We may hope that in practice, the competitive ratio of the greedy algorithm will get better if the budget of the vertices in U grows. However, we can see that this is not the case with the following graph:



The optimal weighted matching is to have the blue vertices on the right match with the blue vertex on the left and the orange vertices on the right match with the orange vertex on the left, for a total sum of bids of $2B$ in the matching. However, under the greedy algorithm, as the blue vertices on the right arrive in order, all of them (except the last, assuming $\varepsilon \ll 1/B$) are going to be matched to the orange vertex on the left, since its bid of $1 + \varepsilon$ is larger than the bid of 1 of the blue vertex on the left. Thus, by the time the orange vertices on the right arrive, the entire budget of the orange vertex on the left is used up, so they get matched with the blue vertex on the left for a bid of 0. The total sum of bids is approximately B in this matching. As such, the competitive ratio of the greedy algorithm is at most, approximately, $1/2$, for arbitrarily large B.

One potential improvement could be to take into account the remaining budget of the advertisers. If we aim to balance the remaining budget that the advertisers have, we keep our options open for longer instead of using up one advertiser's budget completely, as happened to the blue vertex on the left above. However, solely allocating based on the remaining budget is also a bad idea. Consider the following graph:



Again, assume $\epsilon \ll 1/B$. As before, the optimal matching is to match blue with blue and orange with orange for a total bid sum of $2B$ in the matching. Let's consider what an algorithm only optimizing to balance the remaining budgets would do (assuming ties are broken in favor of the higher-bid vertex): After the first blue vertex on the right is matched with the blue vertex on the left, the remaining blue vertices will all be matched to the orange vertex on the left, since it has greater remaining budget. The orange vertices, then, will all but one match with the blue vertex on the left, since after the first orange vertex on the right was matched to the orange vertex on the left, the blue vertex on the left will have more budget remaining. This gives a matching of total bid sum around 2 — the competitive ratio of this algorithm is no better than $1/B$, a lot worse than the greedy algorithm.

It turns out that if we carefully balance between these two approaches, selecting the next matching based both on the bids and on the remaining budgets, we can outperform the greedy algorithm.

Algorithm 5 MSVV Algorithm for Online Weighted Matching

1: When a vertex v arrives, denote by x_u the fraction of u's budget that has been spent so far.
2: Pick the $u \in U$ that maximizes $b_{(u,v)} \cdot (1 - e^{x_u - 1})$, subject to u having an edge to v and u having enough budget left to afford $b_{(u,v)}$. Match v with this u.
3: If no such u exists, leave v unmatched.

Theorem 6 (Competitive Ratio of MSVV)**.** *The competitive ratio of the MSVV*[3] *algorithm approaches* $1 - 1/e$ *as the budget grows (relative to the size of the bids). No algorithm, even randomized, has a better competitive ratio.*

This is a surprising result: While in the simple setting, randomized algorithms outperformed deterministic algorithms, we can see that in online weighted matching, for large enough bids, this deterministic algorithm is optimal!

The MSVV algorithm has many applications in practice and has had a significant impact on how companies like Google design their algorithms for deciding which ads to show to which user. It can be extended to capture an even more realistic setting, for example to allow for advertisers arriving at different times, advertisers paying only if the user interacts with the ad, and the advertiser 'winning' only paying the second-highest bid. In a test-of-time talk at the Economics and Computation conference 2024, Aranyak Mehta from Google Research summarized the impact of the MSVV algorithm as:

> *The core problem of budget management remains important, and the core idea [of spending budget smoothly] remains impactful.*

[3] Named after Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani.