1. Preliminaries

We will begin by formalizing the classic online bipartite matching problem and providing definitions of some terms which we will use throughout this paper.

1.1. Classic Online Bipartite Matching Problem Definition.

In the classic version of the problem, we define the input as the bipartite graph $G = (U, V, E)$, where U and V are two independent sets of n vertices each and E is the set of edges connecting the vertices in U to the vertices in V. For this problem we assume that our algorithm has full access to all of the vertices in V from the beginning, but receives each vertex $u \in U$ one at a time in a preselected order. Each time a vertex $u \in U$ is received, the edges incident to u become known to the algorithm and the algorithm is thus tasked with permanently matching u to an adjacent, unmatched vertex $v \in V$ if possible. The goal of the problem is t o create the largest sized matching possible. Unless otherwise stated, we assume that G has a perfect matching and thus the size of the optimal matching is n.

1.2. Useful Definitions.

Definition 1. We let $N(u) \subset V$ to be the set of previously unmatched vertices in V that are adjacent to a vertex $u \in U$.

Thus, it is not until the algorithm receives a vertex $u \in U$ that the set of vertices $N(u)$ becomes known.

Definition 2. We define m as the algorithms current matching and $m(u)$ as the vertex $v \in V$ that u is matched with. Furthermore, we define m^* as an optimal matching, which we assume is a perfect matching.

2. Classic Online Bipartite Matching Algorithms

Below we present and analyze two algorithms for the classic online bipartite matching problem: the Greedy algorithm and the Ranking algorithm.

2.1. Greedy Algorithm.

The Greedy algorithm, despite its simplicity and naï¿œvetï¿œ, actually provides a reasonable competitiveness, which, as we will later prove, is optimal for deternministic algorithms.

2.1.1. *Greedy Algorithm Description.*

As one might expect, upon reception of a vertex $u \in U$ the Greedy algorithm always, simply matches u with an arbitrary vertex $v \in N(u)$ if $N(u) \neq \emptyset$.

2.1.2. *Greedy Algorithm Analysis.*

It is easy to see that the Greedy algorithm has a competitive ratio of at least $\frac{1}{2}$, because for every vertex u it is either the case that u is matched to some vertex $v \in V$ or $N(u) = \emptyset$. Remember, we assume that there exists a perfect matching m^* which matches all $2n$ of the vertices in n matchings. For every vertex $u \in U$, at least one of u or $m^*(u)$ must be in our Greedy algorithm's matching. Thus, our algorithm matches a minimum of n vertices in total in a minimum of $\frac{n}{2}$ matchings.

2.1.3. *Greedy Algorithm Optimality.* We can further demonstrate that no deterministic algorithm can have better competitiveness than $\frac{1}{2}$. Suppose, for example, an adversarial input in which each of the first $\frac{n}{2}$ vertices from U that the algorithm receives are adjacent to every vertex V. Then the remaining $\frac{n}{2}$ vertices from U are only adjacent to the vertices that the algorithm was determined to have already matched. Thus, clearly for every deterministic algorithm there is an input for which it is $\frac{1}{2}$ competitive.

2.2. **Ranking Algorithm.** Since the upperbound for the competitiveness of determinstic algorithms is $\frac{1}{2}$, we will need to add randomization in order to get an improvement. Luckily, by simply adding a single element of randomness and tweaking our Greedy algorithm so that there is no arbitrary choice, we can get an improved expected competitiveness.

2.2.1. *Ranking Algorithm Description.* The Ranking algorithm begins with an initialization phase that consists of choosing a random permutation of the vertices in V, which will serve as the basis for our ranking function σ. Then upon receiving each vertex $u \in U$, the algorithm matches u to the vertex $v \in N(u)$ which minimizes $\sigma(v)$, such that v has not already been matched. Obviously, if every vertex adjacent to u is already matched, i.e. $N(u) = \emptyset$, the algorithm does not match u.

2.2.2. *Ranking Algorithm Analysis.* Like in the Greedy algorithm, a received vertex $u \in U$ is not matched if and only if $N(u) = \emptyset$. As a result, we know that its competitiveness is always at least $\frac{1}{2}$. Since, the Ranking algorithm has a random element to it, it is not bounded in the same way that deterministic algorithms are, which leads us to the following theorem.

Theorem 3. *The Ranking algorithm has an expected competitiveness of* $1 - \frac{1}{e} \approx 0.63$.

We will "prove" this theorem with an extremely intuitive, but slightly incorrect proof. We encourage enthusiastic readers to read [[[[]]], if they would like to get the correct, but more technical version of the proof.

Let us define m_σ as the matching produced by the Ranking algorithm with ranking function σ for some given input. We can relate Ranking(σ) with the optimal matching m^* using the following lemma.

Lemma 4. *For every vertex $u \in U$, if vertex $v = m^*(u)$ is not matched in m_σ, then in m_σ u is matched to a vertex $v' = m_\sigma(u)$ such that $\sigma(v') < \sigma(v)$.*

Proof. Since vertex v is not matched in m_σ, that means that when u was recieved by the Ranking algorithm, v was still unmatched. Thus, the only reason u would not be matched to v would be if it could get matched to a higher ranking vertex v', such that $\sigma(v') < \sigma(v)$. □

We will use the following lemma to get our competitiveness.

Lemma 5. *If we define x_t as the probability over ranking function σ that the vertex $v \in V$ such that $\sigma(v) = t$ is matched in m_σ, then $1 - x_t \leq \frac{1}{n} \sum_{1 \leq s \leq t} x_s$ for all t in range $[1, n]$.*

Proof. Let us define u as the vertex in U that v is matched with in the optimal matching, i.e. such that $v = m^*(u)$. Furthermore, let us define the set of vertices $R_{t-1} \subset U$ as the set of vertices in U that are matched in m_σ to vertices in V of rank that is no more than $t-1$. In otherwords for all $u \in U$ such that u is matched in m_σ, u is in R_t if and only if $\sigma(m_\sigma(u)) < t$. By Lemma 4, if vertex v is not matched in m_σ, then $u \in R_t$. Thus, the probability that v is not matched in m_σ is bounded by the probability that $u \in R_t$. Clearly, by the defintion of x_t, the probability that v is not matched is $1 - x_t$ and the expected size of $R_t = \sum_{1 \le s < t} x_s$. If u and R_t were independent, then the probability that $u \in R_t$ would simply be $\frac{|R_t|}{n} = \frac{1}{n} \sum_{1 \le s < t} x_s$, which would complete our lemma with $1 - x_t \le \frac{1}{n} \sum_{1 \le s < t} x_s \le \frac{1}{n} \sum_{1 \le s \le t} x_s$.

However, this is where this proof fails. The fact is u and R_t are not actually independent in the way we set it up. The correct, but less intuitive, proof demonstrates how choosing vertices v and u randomly and independently of σ can result in having the relation that if vertex v is not matched in m_σ, then $u \in R_{t+1}$. In this case u and R_{t+1} are independent, which correctly proves the lemma. Again, we encourage interested readers to read the correct proof in [[[]]]. □

Now we can compute the expected competetiveness. It is easy to see that the expected size of m_σ is $\sum_{1 \le s \le t} x_s$. Since we are assuming that there exists a perfect matching, that means the expected competitiveness is $\frac{1}{n} \sum_{1 \le s \le n} x_s$, which we must lower bound using Lemma 5. If we define $S_t = \sum_{1 \le s \le t} x_s$, we can rewrite Lemma 5 to $1 - (S_t - S_{t-1}) \le \frac{1}{n} S_t$, which can be simplified to $1 + S_{t-1} \le \left(\frac{n+1}{n}\right) S_t$. It follows that S_t, and by extension also $\frac{1}{n} \sum_{1 \le s \le t} x_s$, is smallest when all of the inequalites are equalities, which gives us $S_t = \sum_{1 \le s \le t} \left(\frac{n}{n+1}\right)^s$. As a result, our expected competitiveness $\frac{1}{n} \sum_{1 \le s \le n} x_s = \frac{S_n}{n}$ is lower bounded by $\frac{1}{n} \sum_{1 \le s \le n} \left(\frac{n}{n+1}\right)^s$ which after some mathemagic becomes $1 - (\frac{n}{n+1})^n$ which approaches $1 - \frac{1}{e}$ as n goes to infinity, thus concluding our proof.

2.2.3. *Optimality of Ranking Algorithm.*

3. Online Vertex-Weighted Bipartite Matching

The online vertex-weighted bipartite matching problem is a generalization of the online bipartite matching problem. The difference is that each vertex in V now has an associated weight. Instead of maximizing the size of the matching, we now aim to maximize the total weight of the vertex of V which are included in the matching. The online bipartite matching problem described in Section ?? is the special case where all edges in V have weight 1.

Formally, the input to the problem is a bipartite graph $G = (U, V, E, \{w_v\}_{v \in V})$. The vertices in V as well as their weights are known ahead of time. The vertices in U arrive online. As before, the edges of a vertex $u \in U$ are revealed when u arrives. When a vertex arrives it can either be matched to a neighbor in V or not matched. However, once made, a decision cannot be undone. We aim to maximize

the sum of weights of all the matched vertices in V. As before, we will assume that $|U| = |V|$ and that G contains a perfect matching.

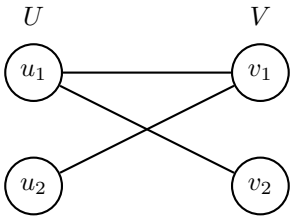
3.1. **Greedy Algorithm.** The obvious greedy solution to this problem is to match a vertex $u \in U$ to its neighbor of greatest weight, or leave it unmatched if it has no remaining neighbors. Call this algorithm Greedy.

Lemma 6. Greedy *is a (1/2)-approximation for vertex-weighted bipartite matching. Furthermore, this is the best approximation ratio the algorithm can achieve.*

Proof. First we show that Greedy is a (1/2)-approximation. Let $O \subseteq E$ be the optimal offline matching, and $A \subseteq E$ be the matching produced by Greedy. If $A \neq O$, then there is some $v_O \in V$ such that v_O is in O but not A. Let u be the vertex which was matched to v_O in O. Then there must be some other vertex v_A such that u was matched to v_A in A. This is because we know that v_O was unmatched when u was being matched, so the only way u would not be matched to v_O is if there were some other neighbor with greater weight.

But now we have that the optimal value 'lost' by u is at most equal to the weight of its match in A. Therefore the total amount which $|A|$ losses is A. Hence, $|A| \geq |O|/2$.

To see that this is the tightest approximation ratio for this algorithm, we offer an example. Consider the following graph.



Let $w_{v_1} = 1 + \epsilon$ for some $\epsilon > 0$ and $w_{v_2} = 1$. Then the greedy algorithm will match only (u_1, v_1), and the optimal solution will match $(u_1, v_2), (u_2, v_1)$. As ϵ approaches 0, the approximation ratio approaches 1/2.

□

We will not offer a proof here, but it turns out that no deterministic algorithm can do better than Greedy.

3.2. **Generalizing Ranking.** Consider the Ranking algorithm described above. For an input in which the range of weights is small, this algorithm will perform well. This is reflected in the fact that Ranking is optimal in the case where all weights are equal. However, when the range of weights is large, this algorithm can perform very poorly.

We want to generalize the ranking algorithm in order to account for the weighted case. We will present a new algorithm which we will call Perturbed-Greedy. We will show that this algorithm is equivalent to Ranking in the case where all weights are equal, and we will show that this algorithm achieves an approximation ratio of $1 - \frac{1}{e}$.

The Perturned-Greedy algorithm will use the function

$$\psi(x) = 1 - e^{x-1}.$$

(1) For each $v \in V$, choose x uniformly at random from $[0, 1]$.
(2) For arriving $u \in U$, match u to the unmatched neighbor v with highest value $b_v\psi(x)$.
(3) Break ties by vertex ID.

Consider the case where all weights are equal. Since x is chosen uniformly at random, choosing vertices according to the highest values of $b_v\psi(x)$ is equivalent to choosing a random ranking. Thus, Perturbed-Greedy is equivalent to Ranking when all weights are equal, as desired.

We will prove the following Theorem.

Theorem 7. *Perturned-Greedy achieves an approximation ratio of* $1 - \frac{1}{e}$ *for the vertex-weighted online bipartite matching problem.*

While we will not offer a proof here, it turns out that this is the optimal approximation ratio for this problem. Intuitively, this result seems reasonable because we know that this is the optimal approximation ratio for a more specific version of this problem.

Proof. Proof here. □