1. Introduction

1.1. Organization of the Report.

1.2. 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.

1.2.1. *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.

1.2.2. *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 $R_t = \sum_{1 \leq s < t} x_s$. If u and R_t were independent, then the probability that $u \in R_t$ would simply be $\frac{|R_t|}{n} = \frac{1}{n} \sum_{1 \leq s < t} x_s$, which would complete our lemma with $1 - x_t \leq \frac{1}{n} \sum_{1 \leq s < t} x_s \leq \frac{1}{n} \sum_{1 \leq s \leq t} x_s$.

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 $\sum_{1 \leq s \leq t} x_s$. Since we are assuming that there exists a perfect matching, that means the expected competitiveness is $\frac{1}{n} \sum_{1 \leq s \leq n} x_s$, which we must lower bound using Lemma 5. If we define $S_t = \sum_{1 \leq s \leq t} x_s$, we can rewrite Lemma 5 to $1 - (S_t - S_{t-1}) \leq \frac{1}{n} S_t$, which can be simplified to $1 + S_{t-1} \leq \left(\frac{n+1}{n}\right) S_t$. It follows that S_t, and by extension also $\frac{1}{n} \sum_{1 \leq s \leq t} x_s$, is smallest when all of the inequalites are equalities, which gives us $S_t = \sum_{1 \leq s \leq t} \left(\frac{n}{n+1}\right)^s$. As a result, our expected competitiveness $\frac{1}{n} \sum_{1 \leq s \leq n} x_s = \frac{S_n}{n}$ is lower bounded by $\frac{1}{n} \sum_{1 \leq s \leq n} \left(\frac{n}{n+1}\right)^s$ 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.

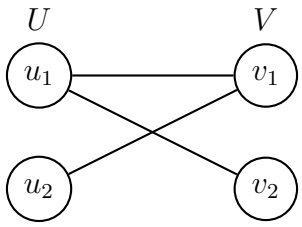
3.1. **Greedy Algorithm.** The obvious greedy solution to this problem is to match a vertex $u \in U$ to its neighbor of greatest weight, or leave it unmatched if it has no remaining neighbors. Call this algorithm Greedy.

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. □

Another interesting variant of the online bipartite matching problem is online bipartite matching with stochastic rewards. The setup is similar to the classic problem introduced by Karp et al. where the goal is to maximize the number of matchings between a stream of vertices V and with known vertices U. The twist is that each matching has some probability of success p_{uv}. The goal is now to maximize the expected number of successful matchings given that some may fail. The variant of this problem was motivated by its practical applications to internet advertising and crowdsourcing. Internet advertising involves matching advertisers' ads to be shown to users of a service where advertisers pay each time a user clicks on an ad (pay-per-click advertising). The likelihood that a user u clicks on an ad v can be modeled by a probability p_{uv}. Thus, this matching problem can be modeled as an online stochastic reward matching problem where the service wants to maximize its revenue given a stream of ads to be matched with users. This setup can also model crowdsourcing where each worker u has a probability p_{uv} of completing task v so the online assignment hopes to maximize the expected number of completed tasks.

Solving this problem for arbitrary $p_u v$ is an open problem, however, Mehta and Panigrahi present online algorithms and analysis for the problem when $\forall u, v : p_{uv} = p$. These algorithms and their bounds on competitive ratios can still be practically usable for the above applications if the variance in p_{uv} is small. For example in online advertising, the probability of a user clicking on an ad is small for almost all users so using one estimate for p should still give informative, approximate bounds. Additionally, the case where $p \to 0$ (called a vanishing probability) is interesting theoretically on its own.

As with any online algorithm problem, algorithms are compared to an optimal algorithm that knows the input stream (the full bipartite graph). A complication here is the non-determinism of the matching success. To mend this, the optimal algorithm (OPT) is defined non-stochastically as being able to fractionally match vertices to maximize $\sum_u \sum_v p x_{uv}$ such that $\sum_u x_{uv} \leq 1$. By giving OPT more power, it upper bounds the best matching of any algorithm. Note that for most graphs, the expected number of matchings is $p|U| = pn$ which is used to simply the analysis.

Algorithms for this problem are separated into two categories: adaptive and non-adaptive. After each pair is matched, the match's success is immediately computed according to p. If successful, u and v are both removed from the set of unmatched vertices and the match is scored. Otherwise, u is removed from the set of unmatched

vertices, and v remains free to be matched later. An adaptive algorithm can learn the result of a matching immediately and attempt to rematch v with a later u if prior matches failed. However, a non-adaptive algorithm does not learn this information until after the algorithm completes and may try to rematch a v that was already successfully matched. In this case, only 1 successful matching is scored for each v. Intuitively, non-adaptive algorithm are less powerful than adaptive algorithms because they learn less information which results in a lower expected score. In fact, it can be proven that the competitive ratio of non-adaptive algorithms for this problem is at most 1/2 when $p \leftarrow 0$ by applying Yao's minimax principle on a particularly bad distribution of inputs for a deterministic algorithm. This then bounds the best performance of any randomized algorithm. The proof isn't fully presented in the paper and isn't that interesting.

As expected, adaptive algorithms have more power than non-adaptive algorithms and are able to achieve better competitive ratios. An extension of the Ranking algorithm seen earlier performs well when $p \leftarrow 1$ giving the same competitive ratio as the non-stochastic classic problem. But interestingly enough, a deterministic algorithm called Stochastic Balance performs better when $p \leftarrow 0$. And this algorithm's competitive ratio is 1/2 when $p = 1$ showing that it is consistent with the theorem that any deterministic algorithm has competitive ratio of at most 1/2 in the classic problem.

Before diving into these algorithms, we note a generalization of the stochastic bipartite matching problem that makes analysis of these algorithms possible. Stochastic Matching can be viewed as a packing problem where each v has (prior to running any algorithm) a randomly choosing bin size which represents the number of matches that must be made before that vertex will have a successful match. More specifically, this threshold is a geometric random variable given by the following.

Definition: $Pr[\theta_u = pt] = p(1-p)^{t-1}$

In this generalization, the goal is to maximize the expected number of filled bins such that these bins have had enough potential matches that one of them now succeeded. Note however that adaptive algorithms working under this view don't know the size of each bin until it is filled which is consistent with learning the success of a matching. More generally this equating of load on a bin with probability of a successful match can be proven by a simply argument from linearity of expectation.

The competitive ratio of both of these algorithms is proved using a technique called factor-revealing linear programming. Drawing from ideas involving the duality between the algorithm and an adversary, it formulates the adversary's "choice" of some outcomes of the algorithm to maximize the number of vertices that are not matched. The adversary's moves however are constrained to be consistent with the applied algorithm. After finding this primal linear program, taking the dual and applying weak duality to some feasible solution of the dual gives an upper bound on the maximum primal objective value. Thus, the result is an upper bound of the worst possible

performance of the algorithm giving the bound on the competitive ratio. The goal of the algorithmist is to find a set of constraints that follow from running the algorithm that limit the ability of the adversary to maximize the worst case performance. This technique is particularly effective because of the algorithms' simplicity allows them to be encoded into these constraints.

The ranking algorithm itself is the same as previously presented. The algorithm randomly permutes the set of U vertices. Upon an incoming vertex to be matched, the algorithm matches it with the highest ranked neighbor that has not been successfully matched. The analysis introduces the notion of a good matching and a bad matching. A bad matching occurs when matched edge (u, v) has $rank(u) < rank(opt(v))$. The intuition is that the algorithm matched a vertex too early (earlier than the optimal). Otherwise, the matching is a good match. $good(s)$ and $bad(s)$ are respectfully defined to be the expected load of the vertex u with $s = rank(u)$ due to good matches and bad matches. The linear program takes the following form where the adversary chooses $good(s)$ and $bad(s)$ to minimize the expected number of matches under some algorithmic constraints.

minimize $\sum_s match(s)$
subject to
1)$\forall s : match(s) = good(s) + bad(s)$
2)$\forall t : \frac{1-(1-p)^{\frac{1}{p}}}{p} - match(t) \leq \sum_{s \leq t} \frac{bad(s)}{n-s}$
3)$\forall t : \sum_{s \leq t} \frac{s * bad(s)}{n-s} \leq \sum_{s \leq t+1} good(s)$
4)$\sum_t match(t) \geq \frac{1}{2} + \frac{\sum_t good(t)}{2}$

Constraint 1) trivially comes from the definitions of $good(s)$ and $bad(s)$. The other constraints are more involved. We prove only constraint 2) to give the flavor for how these are discovered.

Proof : Recall that using our bin load approach, we defined a vertex u being successfully matched if its load equaled $\theta_u = match(rank(u))$ here. So if the vertex at rank t is matched on expectation $E[\theta_u] = \frac{1-(1-p)^{1/p}}{p} = match(t)$ and $\frac{1-(1-p)^{1/p}}{p} - match(t) = 0$ satisfying the constraint. Otherwise if rank t is not matched on expectation, we know that incoming v vertices that would have been matched to u by OPT must be matched to some u' with a smaller rank. It follows that this set of v vertices contains at least the missing load from the vertex at rank t giving the set load at least $\frac{1-(1-p)^{1/p}}{p} - match(t)$. Now we look for an upper bound on the load of this set. We notice that the mismatch of these v vertices constitutes a bad matching for each position s they are matched because for some permutations of the ranking order they matched the vertex differently than OPT. Furthermore, it would have been a bad matching for any of the $n - s$ permutations if u had been later in ranking than

s. So the permutation with u at t is expected to give $\frac{1}{n-s}$ of the bad match load of s. Thus the upperbound on this load is $\sum_{s \leq t} \frac{bad(s)}{n-s}$ giving the constraint.

Upon taking the dual of this linear program and finding a feasible solution to the dual, the authors indicated that their solution had objective function lower bounded by $((1-\frac{1}{e}) - (1-\frac{1}{e})(1-p)^{1/p})(\frac{n}{p})$ which in turn bounds the worst performance of the adversary giving a competitive ratio of $((1-\frac{1}{e}) - (1-\frac{1}{e})(1-p)^{1/p})$

The Stochastic Balance algorithm doesn't involve any randomization. Instead, it records the load of each U (which as mentioned is the number of failed matches $*p$) and an incoming vertex to the neighbor with the least load that is also not successfully matched. The analysis for this algorithm uses the same factor-revealing linear programming approach but with a different language for constraining the adversary. It defines $f_u(x)$ to be the variables of the linear program corresponding to the probability that vertex u has failed to be matched at the end of the algorithm with load x. Note that the number of load levels is linear with $|V|$ because there are at most $|V|$ matching edges to any u which contribute p to the load. Thus the expected number of failures for the algorithm is $\sum_u \sum_x f_u(x)$ which the adversary attempts to maximize under constraints of the stochastic balance algorithm. Finally, L_u^* is an additional variable encoding the load assigned to u by the optimal algorithm.

maximize $\sum_u \sum_x f_u(x)$
subject to
1)$\forall x : \sum_{y \leq x} \sum_u (1 + L_u^*) f_u(y) + (1-p)^{x/p} \sum_{y>x} (1-p)^{-y/p} (\sum_u f_y(y)) \leq n$
2)$\forall x : \sum_u ((1-p)^{-L_u^*/p} + \sum_{y<L_u^*} f_u(y)(1 + L_u^* - x)(1-p)^{-y/p}) \leq n$
3)$\sum_x \sum_u f_u(x)(1-p)^{-x/p} = n$

Like in the Ranking Algorithm analysis, we will prove the first constraint to show the flavor for how these are constructed and along the way prove the third constraint. The original paper itself only sketches out the proof of the second constraint.

Proof : Let $L_u^\infty(\theta)$ be the load on vertex u given that θ is modified such that $\theta_u = \infty$ or in words the load on u when the algorithm finishes under the other vertices thresholds.

Let $q_u(x) = \sum_{\theta : L_u^\infty(\theta) = x} Pr[\theta = \theta_{-u}]$ This just encodes the probability that x is an resulting load for u as sum of the individual likelihoods of drawings of θ.

$f_u(x) = Pr[(\theta_u > x) \wedge (L_u^\infty(\theta_{-u}) = x)]$
$f_u(x) = Pr[\theta_u > x] Pr[L_u^\infty(\theta_{-u}) = x]$
$f_u(x) = (1-p)^{(x/p)} q_u(x)$

This follows from representing the probability of a resulting load being unsuccessful as the intersection of when the load is less than its threshold and the result of the vector constitutes an unsuccessful load vector. By independence, this separates giving the resulting expression where $(1-p)^{(x/p)}$ indicates that the matching fails x/p times,

and $q_u(x)$ is an unsuccessful resulting load. We can represent the probability that a load x succeeds for vertex u as $g_u(x)$ and find a similar looking expression.

$g_u(x) = Pr[(\theta_u = x) \wedge (L_u^{\infty}(\theta_{-u}) \geq x)]$
$g_u(x) = Pr[\theta_u = x]Pr[L_u^{\infty}(\theta_{-u}) \geq x]$
$g_u(x) = p(1-p)^{(x/p-1)} \sum_{y \geq x} q_u(y)$

This follows from noting that a load is successful if it equals its threshold. But also, the resulting load of the algorithm if there is no limit on the load must give some load greater than x or x is not possible. This probability of the first part is $p(1-p)^{(x/p-1)}$ while the probability of the second is the sum of each of these disjoint $q_u(y)$ evens. Because $f_u(x)$ and $g_u(x)$ are defined complementarily, we note their sum over the possible loads and vertices must equal the total number of vertices, $\sum_u \sum_x (f_u(x) + g_u(x)) = n$. Substituting $g_u(x)$ for $f_u(x)$ gives the third constraint. The first constraint can be derived in just a few more steps

$\sum_{y \leq x} \sum_u L_u^* f_u(y) \leq \sum_{y \leq x} y \sum_u (f_u(y) + g_u(y)) + x \sum_{y > x} \sum_u (f_u(y) + g_u(y))$

This is because for a vertex u that ends up with load x, all u' on the OPT solution that took a possible matching from u must have had a lower load at the time to be picked before u by the stochastic balance algorithm. So the expected sum of optimal loads assigned when the algorithm fails an assignment are less than the sum of expected load of the failed vertices produced by the stochastic balance algorithm. Substituting $g_u(x)$ for $f_u(x)$, simplifying, and apply the equality learned in the third constraint gives the first constraint.

$\sum_{y \leq x} \sum_u (1 + L_u^*) f_u(y) + (1-p)^{x/p} \sum_{y > x} (1-p)^{-y/p} (\sum_u f_y(y))$
$\leq \sum_x \sum_u f_u(x)(1-p)^{-x/p} = n$

Using the weak duality again, the authors mention that a bound on the linear program gives competitive ratio of $(1 + (1-p)^{2/p})/2$. The intersection of these two functions for the competitive ratios of Ranking and Stochastic Balance shows that the latter performs better when $p < 0.26$. Thus unlike the classic and weighted case, a deterministic algorithm is so far been proven to perform better than the randomized algorithm as $p \leftarrow 0$.