SURVEY ON ONLINE BIPARTITE MATCHING AND ITS VARIANTS

CAELAN GARRETT (CAELAN@MIT.EDU), CHRIS GRAVES (GRAVES@MIT.EDU),
AND CASEY O'BRIEN (CMOBRIEN@MIT.EDU)

1. Introduction

Although first introduced in 1990 by Karp, Vazirani, and Vazirani [[KVV]], the online bipartite matching problem has in recent years garnered a lot of new interest due to its applications in Internet advertising and crowdsourcing. Like its offline counterpart, the general goal of the online bipartite matching problem and its variants is to produce an optimal matching between two mutually exclusive sets of vertices. Unlike its offline counterpart, however, one of the sets of vertices is received online, i.e. one-by-one. For each online step, a vertex of the online set is revealed along with its set of adjacent vertices. The algorithm is tasked with irreversibly matching the incoming vertex if possible. Variants of this problem include considering weighted vertices and adding uncertainty if a matching will succeed. These variants generalize the original problem to a wide variety of applications. For example, it is not too difficult to see how this problem applies to crowdsourcing. Consider matching an online set of users to a static set of tasks.

1.1. Organization of the Report.

In Section 2, we will demonstrate and analyze two algorithms (one deterministic and one random) that solve the classic version of the online bipartite matching problem. Next, in Section 3, we will present the online vertex-weighted bipartite matching problem and the analogous algorithms that solve that problem. Finally, in Section 4 we will introduce online bipartite matching with stochastic rewards presenting again both deterministic and random solutions.

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, as first introduced in [[KVV]], we define the input as the bipartite graph $G = (U, V, E)$, where U and V are two independent sets of vertices 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 on u are revealed and the algorithm then either permanently matches u to an unmatched, adjacent vertex $v \in V$ or chooses to leave u permanently unmatched. The goal of the problem is to maximize the size of the matching. For the purposes of this paper, we assume that $|U| = |V| = n$ and that G has a perfect matching. Thus the size of the optimal matching is n.

1.2.2. *Useful Definitions.*

Definition 1. Let $N(u) \subset V$ be the set of available vertices that u can still be matched with. More formally, we define $N(u)$ as the set of vertices in V that are adjacent to a vertex $u \in U$, but still haven't been matched by the algorithm.

Note that $N(u)$ for a vertex $u \in U$ is revealed to the algorithm only once the algorithm receives vertex u.

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

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) as they were presented in [[KVV]] and [[BM]].

2.1. Greedy Algorithm.

The Greedy algorithm, which we will call Greedy, actually provides a reasonable competitiveness, which, as we will later prove, is optimal for deterministic algorithms. The algorithm works as follows. Upon reception of a vertex $u \in U$, Greedy matches u to an arbitrary vertex $v \in N(u)$ if $N(u) \neq \emptyset$, and otherwise leaves it unmatched.

Theorem 3. Greedy *is a (1/2)-competitive algorithm for the classic online bipartite matching problem.*

Proof. It is easy to see that Greedy has a competitive ratio of at least $1/2$, because for every vertex $u \in U$ either 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 Greedy's matching. Thus, our algorithm matches a minimum of n vertices in total in a minimum of $n/2$ matchings. □

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

2.2. Ranking Algorithm.

Since the upper-bound for the competitiveness of deterministic algorithms is $1/2$, we will need to add randomization in order to get an improvement. Luckily, as suggested in [[KVV]], by simply adding a single element of randomness and tweaking Greedy so that there is no arbitrary choice, we can get an improved expected competitiveness.

The Ranking algorithm, which we will call Ranking, begins with an initialization phase that consists of choosing a random permutation of the vertices in V. With this random permutation we define our ranking function $\sigma(v)$ for all vertices $v \in V$, which simply returns the index of v in the permutation. 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.

Like in Greedy, 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 $1/2$. Since Ranking 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 4. Ranking *has an expected competitiveness of* $1 - 1/e \approx 0.63$.

We will "prove" this theorem with an extremely intuitive, but slightly negligent proof that is presented in [[BM]] for the sake of intuition. We encourage enthusiastic readers to read [[BM]] in its entirety, if they would like to also get the fully correct, but more technical version of the proof. It is important to note here, that the original proof, as presented in [[KVV]] had an error that was detected by Krohn and Varadarajan in 2007. The error was first corrected by [[GM]], but then quickly simplified in [[BM]].

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

Lemma 5. *For every vertex $u \in U$, if vertex $v = m^*(u)$ is not matched in m_σ, then $m_\sigma(u) = v'$ such that $\sigma(v') < \sigma(v)$.*

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

We will use the following lemma to get our competitiveness.

Lemma 6. *If we define p_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 for all t in range $[1, n]$ we have*

$$1 - p_t \leq \frac{1}{n} \sum_{1 \leq s \leq t} p_s.$$

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 $R_t \subset U$ as the set of vertices in U that are matched in m_σ to vertices in V of a rank that is less than t. In other words for all $u \in U$ such that u is matched in m_σ, u is in R_t if and only if $\sigma(m_\sigma(u)) < t$. We can see an example of the formulation of R_t in Figure 2.1.

By Lemma 5, if vertex v is not matched in m_σ, then $\sigma(m_\sigma(u)) < t$ which means $u \in R_t$. Thus, the probability that v is not matched in m_σ is bounded by the probability that $u \in R_t$. Clearly, by the definition of p_t, the probability that v is not matched is $1 - p_t$. Futhermore, the expected size of R_t is $|R_t| = \sum_{1 \leq s < t} p_s$, that is the sum of the probabilities of each vertex v' being matched such that $\sigma(v') < t$.

Let us suppose, albeit somewhat incorrectly, that u and R_t were independent, meaning that our formulation of R_t shared no dependencies with our formulation of u. Then the probability that $u \in R_t$ would simply be the expected size of R_t over n, or $\frac{|R_t|}{n} = \frac{1}{n} \sum_{1 \leq s < t} p_s$. Then we can complete our lemma with

$$1 - p_t \leq \frac{1}{n} \sum_{1 \leq s < t} p_s \leq \frac{1}{n} \sum_{1 \leq s \leq t} p_s.$$

However, the rigorous, correct proof is more complex. The fact is that u and R_t are not actually independent in the way we set it up, because both variables are dependent on t. Vertex u is dependent on t since we have that $\sigma(m^*(u)) = t$ and the vertex set R_t is by definition dependent on t. The correct, but far 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 $u \in R_t$. In this case u is not dependent on t and therefore R_t and u are independent, which

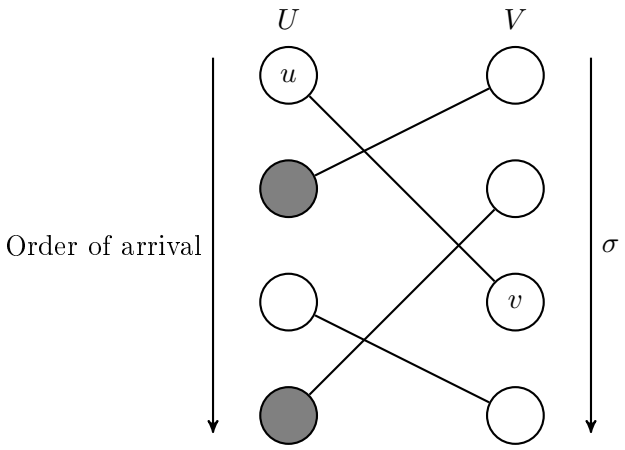

FIGURE 2.1. The matching m_σ is shown in the graph above. Let $t = \sigma(v)$. Then the highlighted nodes represent the set R_t.

correctly proves the lemma. Again, we encourage interested readers to read the fully correct proof in [[BM]]. □

Now we can compute the expected competitiveness. It is easy to see that the expected size of m_σ is $\sum_{1\leq s\leq n} p_s$. Since we are assuming that there exists a perfect matching, that means the expected competitiveness is

$$\frac{|m_\sigma|}{|m^*|} = \frac{1}{n}\sum_{1\leq s\leq n} p_s.$$

We will lower bound this competitiveness using Lemma 6. Let us define $S_t = \sum_{1\leq s\leq t} p_s$ (note that this makes our expected competitiveness equal to S_n/n). We can rewrite Lemma 6 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 turns out that S_t, and by extension also S_n/n, is smallest when all of the inequalities are equalities, which can be solved to give us

$$S_t = \sum_{1\leq s\leq t} \left(\frac{n}{n+1}\right)^s.$$

As a result, our expected competitiveness S_n/n is lower bounded by

$$\frac{1}{n}\sum_{1\leq s\leq n} \left(\frac{n}{n+1}\right)^s.$$

After some non-trivial mathematical manipulation, we can discover that this value is equivalent to

$$1 - \left(\frac{n}{n+1}\right)^n.$$

As n approaches infinity, this value converges to $1 - 1/e$, thus concluding our proof.

Now that we demonstrated that the expected competitiveness of RANKING is at least $1 - 1/e$, the question becomes, "Is it possible do better?" The following theorem proven in [[KVV]], suggest that no, it is not possible.

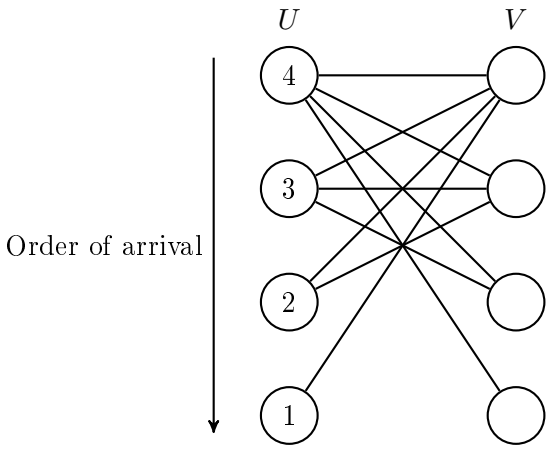

FIGURE 2.2. Consider the graph above, where the degree of nodes in U are displayed. Note that unlike in Figure 2.1, the edges in this graph represent E and not a possible matching. Given the order of arrival, it is very unlikely that a randomized algorithm returns the optimal matching.

Theorem 7. *The competitiveness of any online bipartite matching algorithm is bounded above by* $(1 - 1/e) + O(1)$.

In [[KVV]], the authors demonstrate this theorem by proving that no online algorithm can do better for a particular graph with exactly one perfect matching in which the first few incoming vertices from U are adjacent to all or most of the the vertices in V, but the last few incoming vertices from U are only adjacent to a few or just one of the vertices in V. An example of such a graph is shown in Figure 2.2. The reason it causes this bound, after some nontrivial mathematical analysis, is that the first vertices received from U have many potential vertices to be matched with from V. Thus, it is unlikely that they will be matched with their optimal matches, which uses up the few potential matches for the later incoming vertices from U before the algorithm gets to them.

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 vertices of V which are included in the matching. The classic online bipartite matching problem described in Section 1.2.1 is the special case where all vertices in V have weight 1.

Formally, the input to the problem is a bipartite graph $G = (U, V, E, \{w_v\}_{v \in V})$. Similarly to before, the vertices in V as well as their weights are known ahead of time and the vertices in U arrive online. The edges of a vertex $u \in U$ are revealed when u arrives. When a vertex arrives it can either be permanently matched to an unmatched neighbor in V or not matched. We aim to maximize the sum of weights of all the matched vertices in V. As before, we will assume that $|U| = |V| = n$ and that G contains a perfect matching.

In this section, we will present two algorithms to solve the online vertex-weighted bipartite matching problem. We will then show that these algorithms are generalizations of the algorithms from Section 2, and furthermore that they are able to achieve the same competitive ratios.

3.1. **Generalized 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 Weighted-Greedy.

Theorem 8. Weighted-Greedy *is a (1/2)-competitive algorithm for vertex-weighted bipartite matching. Furthermore, this is the best competitive ratio the algorithm can achieve.*

Proof. First we show that Weighted-Greedy has a competitive ratio of at least $1/2$. Let m be the matching produced by Weighted-Greedy. Let $S \subset V$ be the set of vertices which are not matched in m. Since m^* is a perfect matching, we have that $|m| + |S| = |m^*|$, where $|m|$ denotes the sum of the weights of all weighted vertices matched in m.

Consider a vertex $v \in S$ and $u \in U$ such that u and v are matched in m^*. By the same reasoning as in the proof of Lemma 5, u must be matched in m to a vertex v' of weight at least w_v. But then for each vertex in S, there is a vertex in m with at least equal weight. Therefore, $|S| \leq |m|$. It follows that $|m| \geq |m^*|/2$.

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

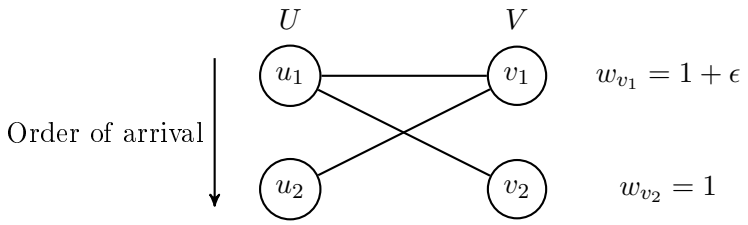

Figure 3.1. A graph demonstrating the tightness of the $1/2$ competitive ratio for Weighted-Greedy.

Say that vertices arrive in the order (v_1, v_2), and 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 competitive ratio approaches $1/2$.

□

In Section 2.1 we showed that no deterministic algorithm can have a competitive ratio of better than $1/2$ in the unweighted case. Analogously, this is the best competitive ratio that can be achieved by any deterministic algorithm in the weighted case.

3.2. **Generalized Ranking Algorithm.** Consider the algorithm Ranking, as described in Section 2.2. 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, it turns out that when the range of weights is large, this algorithm can perform very poorly.

We want to generalize Ranking in order to account for the weighted case. We will present a new algorithm which we will call Perturbed-Greedy, and then we will show that this algorithm

is equivalent to RANKING in the case where all weights are equal. This algorithm will begin by 'perturbing' each of the weights of the vertices in V. Then it will follow the GREEDY scheme as outlined above.

More formally, the PERTURBED-GREEDY algorithm will use the function

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

The algorithm begins with an initialization step in which we 'perturb' the weight of each $v \in V$. To do so, for each v we choose x uniformly at random from $[0, 1]$, and then change the weight of the vertex to $w_v\psi(x)$. Then, for arriving $u \in U$, we match u to the unmatched neighbor with greatest 'perturbed' weight, breaking 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 $w_v\psi(x)$ is equivalent to choosing a random ranking. Thus, PERTURBED-GREEDY is equivalent to RANKING when all weights are equal, as desired.

Theorem 9. PERTURBED-GREEDY *achieves an approximation ratio of* $1 - 1/e$ *for the vertex-weighted online bipartite matching problem.*

We will offer some intuition as to why the PERTURBED-GREEDY algorithm is an effective solution to the weighted vertex problem. Consider two previous algorithms which we could use to solve this problem: WEIGHTED-GREEDY and RANKING. When V has a large range of weights, WEIGHTED-GREEDY will perform quite well, because the algorithm will place higher priority on matching the larger weights, and the smaller weights matter less when the range is large. On the other hand, we have already noted that RANKING performs well on graphs where the range of weights is small.

To develop a good algorithm for the weighted case, we need something which combines the benefits of each of these algorithms. When weights are highly variable, perturbing the weights will have little effect on their relative ordering. Thus PERTURBED-GREEDY will behave very similarly to WEIGHTED-GREEDY, and therefore is able to find a good solution in that case. When weights are very similar, PERTURBED-GREEDY will behave very similarly to RANKING, and thus does well in this situation as well.

While we will not offer a full proof of the theorem here, we note that this results is consistent with the competitive ratio of RANKING. It turns out that, as is the case with RANKING, this is the optimal competitive ratio for this problem. This result seems reasonable because we know that this is the optimal competitive ratio for the more specific version of this problem.

The proof of the approximation ratio of PERTURBED-GREEDY is quite similar to the proof of the approximation ratio of RANKING in Section 2.2. We encourage readers to read this proof in [[AGKM]].

4. STOCHASTIC ONLINE BIPARTITE MATCHING

Another interesting variant of the online bipartite matching problem is online bipartite matching with stochastic rewards, as presented in [[MP]]. The setup is similar to the classic problem presented in Section 2. As before, the goal is to maximize the number of matchings between a set of U of vertices which are received online and a set V which is known from the start. The difference here is that each matching has some probability of success p_{uv}. The algorithms presented in this section are adaptive meaning they immediately learn the result of an attempted matching. The goal is now to maximize the expected number of successful matchings given that some attempts may fail.

This variant of the classic 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 v clicks on an ad u 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 v has a probability p_{uv} of completing task u so the online assignment hopes to maximize the expected number of completed tasks.

Solving this problem for arbitrary p_{uv} is an open problem; however [[MP]] presents online algorithms and analysis for the problem when all edges have the same probability of success. Formally, for all $u \in U$ and $v \in V$, we have $p_{uv} = p$. These algorithms and their bounds on competitive ratios are still practically usable for the above applications if the variance in values of 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.

4.1. **Stochastic Optimal Algorithm.** As with any online algorithm problem, algorithms are compared to an optimal offline algorithm. A complication here is the non-determinism of the matching success. To mend this, we say that the optimal algorithm is being able to fractionally match vertices. Let $0 \leq x_{uv} \leq 1$ be the fraction of u and v that are matched. Thus, it aims to maximize the expected number of matchings, given by

$$\sum_{u \in U} \sum_{v \in V} p \cdot x_{uv},$$

such that no vertex has more than a total of one full matching (i.e., $\sum_{u \in U} x_{uv} \leq 1$ for all $u \in U$ and $\sum_{v \in V} x_{uv} \leq 1$ for all $v \in V$). By giving the optimal algorithm more power, we are able to upper bound the best matching of any algorithm.

Two algorithms are known to perform well for Stochastic Online Bipartite Matching [[MP]]. An extension of Ranking, as described in Section 2.2, called Stochastic-Ranking, performs better when $p \to 1$. Interestingly enough, a deterministic algorithm, called Stochastic-Balance, performs better when $p \to 0$. These algorithms will be discussed in more detail in the following sections.

4.2. **Proof Techniques.** The competitive ratio of both of these algorithms is proved using a technique called factor-revealing linear programming [[JMMSV]]. 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, in this case the expected number of failed matchings. 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 on the outcomes that follow from running the algorithm and limit the ability of the adversary to maximize the worst case performance. These constraints are not usually nice looking or have obvious intuitive meaning, but with more discovered constraints, the bound on the competitive ratio will be tighter. This technique is particularly effective in online algorithms because the algorithms' simplicity allows them to be encoded into these constraints.

4.3. **Stochastic Balance Algorithm.** STOCHASTIC-BALANCE does not involve any randomization. On an incoming u, the algorithm attempts to balance the number of failed attempted matchings among vertices in V. To do so, STOCHASTIC-BALANCE always matches u to the unsuccessfully matched neighbor with the least number of failed matchings. The intuition for why this is effective for $p \to 0$ is that the concavity of the function for the expected number of matchings when $p \to 0$ is such that it is maximized when the number of attempts is distributed as evenly as possible.

The analysis of the competitive ratio is made possible by formulating this problem using bin packing. The idea is that each v randomly chooses a bin size b_v that is unknown to the algorithm before the algorithm starts. Each potential matching with v is an item with size p that is placed into the bin for v and adds a load of p. We say that v is successfully matched if and only if its bin is exactly full. At that point, it no longer accepts potential matches. In essence, b_v encodes the number of matching attempts needed before a success. This language allows us to easily represent the number of matching attempts as the current load divided by p. By linearity of expectation, the expected load on a vertex is equal to the probability of a successful matching.

Definition 10. Let b_v be a multiple of p of a geometric random variable with parameter p. It represents the bin size for vertex v. Formally, we have

$$Pr[b_v = pt] = p(1-p)^{t-1}.$$

The factor-revealing linear program of [[MP]] defines $f_v(x)$ to be the variables of the linear program corresponding to the probability that vertex v has failed to be matched at the end of the algorithm with resultant load x. Note that the number of load levels is linear with $|U|$ because there are at most $|U|$ matching edges to any v which contribute p to the load. The expected number of failures for the algorithm is

$$\sum_v \sum_x f_v(x),$$

which the adversary attempts to maximize under the constraints of STOCHASTIC-BALANCE. Finally, L_v^* is an additional variable encoding the load assigned to v by the optimal offline algorithm at the end of the algorithm.

Next, we display the factor-revealing linear program for STOCHASTIC-BALANCE.

Definition 11. STOCHASTIC-BALANCE Factor-Revealing Linear Program

$$\text{maximize} \quad \sum_x \sum_v f_v(x)$$

$$\text{subject to} \quad \sum_x \sum_v f_v(x)(1-p)^{-x/p} = n \tag{4.1}$$

$$\sum_{y \le x} \sum_v (1 + L_u^*) f_u(y) + (1-p)^{x/p} \sum_{y > x} (1-p)^{-y/p} \left(\sum_u f_y(y) \right) \le n \quad \forall x \tag{4.2}$$

$$\sum_v \left((1-p)^{-L_v^*/p} + \sum_{y < L_v^*} f_v(y)(1 + L_v^* - x)(1-p)^{-y/p} \right) \le n \quad \forall x \tag{4.3}$$

We will prove the correctness of constraints 4.1 and 4.2 to give a flavor for how these constraints are discovered. The full proof of the derivation of constraint 4.3 is quite involved, but interested readers can see it in [[MP]]. Before proceeding, we give the following definitions.

Definition 12. Let $L_v^\infty(b)$ be the load on vertex v given that b is modified such that $b_v = \infty$ but $b_{v'}$ for $v \neq v'$ is the same. In words, it is the load on v after the algorithm runs with an unlimited threshold for v making it not possible to be successfully matched.

Definition 13. Let $q_v(x)$ be the probability that x is the resulting load for v after any run of the algorithm. This is computed by summing the probabilities of randomly obtaining each b that gives load x for v. Formally, we have

$$q_v(x) = \sum_{b: L_v^\infty(b) = x} \left(\prod_{z \neq v} Pr[b_z] \right).$$

We are now equipped to prove the correctness of the constraints 4.1 and 4.2. We will do so through the following Lemmas.

Lemma 14. *For the variables as defined above, we have*

$$f_v(x) = (1-p)^{(x/p)} q_v(x).$$

Proof.

$$\begin{aligned} f_v(x) &= Pr[(b_v > x) \wedge (L_u^\infty(b_v) = x)] \\ &= Pr[b_v > x] Pr[L_u^\infty(b_v) = x] \\ &= (1-p)^{(x/p)} q_v(x) \end{aligned}$$

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 algorithm terminates with x. By independence, this separates giving the resulting expression where $(1-p)^{(x/p)}$ indicates that the matching fails x/p times and $q_v(x)$ is probability of the resulting load. We can represent the probability that load x succeeds for vertex v as $g_v(x)$ and find a similar expression. □

Lemma 15. *For the variables as defined above, we have*

$$g_v(x) = p(1-p)^{(x/p-1)} \sum_{y \geq x} q_v(y).$$

Proof.

$$\begin{aligned} g_v(x) &= Pr[(b_v = x) \wedge (L_v^\infty(b_v) \geq x)] \\ &= Pr[b_v = x] Pr[L_v^\infty(b_v) \geq x] \\ &= p(1-p)^{(x/p-1)} \textstyle\sum_{y \geq x} q_v(y) \end{aligned}$$

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 possibly obtained. 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 $q_v(y)$ events. □

Lemma 16. *Constraint 4.1 is correct. Formally, we have*

$$\sum_x \sum_v f_v(x)(1-p)^{-x/p} = n.$$

Proof. Because $f_v(x)$ and $g_v(x)$ are defined complementarily, the sum over the possible loads and vertices must equal the total number of vertices, $\sum_v \sum_x (f_v(x) + g_v(x)) = n$. Substituting $g_v(x)$ for $f_v(x)$ gives constraint 4.1. □

Lemma 17. *Constraint 4.2 is correct. Formally, we have*

$$\sum_{y \le x} \sum_v (1 + L_u^*) f_u(y) + (1-p)^{x/p} \sum_{y > x} (1-p)^{-y/p} (\sum_u f_y(y)) \le n.$$

Proof.

$$\sum_{y \le x} \sum_v L_v^* f_v(y) \le \sum_{y \le x} y \sum_v (f_v(y) + g_v(y)) + x \sum_{y > x} \sum_v (f_v(y) + g_v(y))$$

$$\sum_{y \le x} \sum_v (1 + L_v^*) f_v(y) + (1-p)^{x/p} \sum_{y > x} (1-p)^{-y/p} (\sum_v f_y(y)) \le \sum_x \sum_v f_v(x)(1-p)^{-x/p} = n$$

This is because for a vertex v that ends up with load x, all v' on the optimal offline solution that took a possible matching from v must have had a lower load at the time to be picked before u by Stochastic-Balance. So the expected sum of optimal loads assigned when the algorithm fails an assignment is less than the sum of expected load of the failed vertices produced by Stochastic-Balance. Substituting $g_v(x)$ for $f_v(x)$, simplifying, and appling the equality learned in constraint 4.1 gives constraint 4.2. □

Using this linear program, [[MP]] proves the following theorem.

Theorem 18. Stochastic-Balance *achieves a competitive ratio for the stochastic online bipartite matching problem of*

$$\frac{1 + (1-p)^{2/p}}{2}$$

By taking the dual of this linear program given in Definition 11, finding a feasible solution, and applying weak duality, [[MP]] was able to to obtain an upper bound on the number of failed matchings which Stochastic-Balance will encounter. The full derivation of the competitive ratio is not given here, but we encourage interested readers to consult [[MP]] for the full proof.

4.4. **Stochastic Ranking Algorithm.** Stochastic-Ranking is very similar to Ranking, as described in Section 2.2. It begins by randomly permuting the set of vertices V. Upon an incoming vertex to be matched, the algorithm matches it with the highest ranked neighbor that has not been successfully matched. Unlike our previous proofs for the competitive ratio of Ranking, the proof for Stochastic-Ranking uses a Factor-Revealing Linear Program. We won't go into the details of this linear program, but it compares the difference in performance of Stochastic-Ranking from the optimal offline algorithm by classifying matchings as good matchings or bad matchings. A bad matching occurs when matched edge (u, v) has $\sigma(v) < \sigma(m^*(u))$. Otherwise, a matching is a good matching. The linear program finds constraints relating the load caused by good and bad matchings resulting from the algorithm to prevent the adversary from maximizing the number of failed matchings.

Theorem 19. Stochastic-Ranking *achieves a competitive ratio for the stochastic online bipartite matching problem of*

$$\left(1 - \frac{1}{e}\right) - \left(1 - \frac{2}{e}\right) \cdot (1-p)^{1/p}$$

The technique for proving this competitive ratio is very similar to the technique used to prove Theorem 18. We again encourage interested readers to consult the proof in [[MP]].

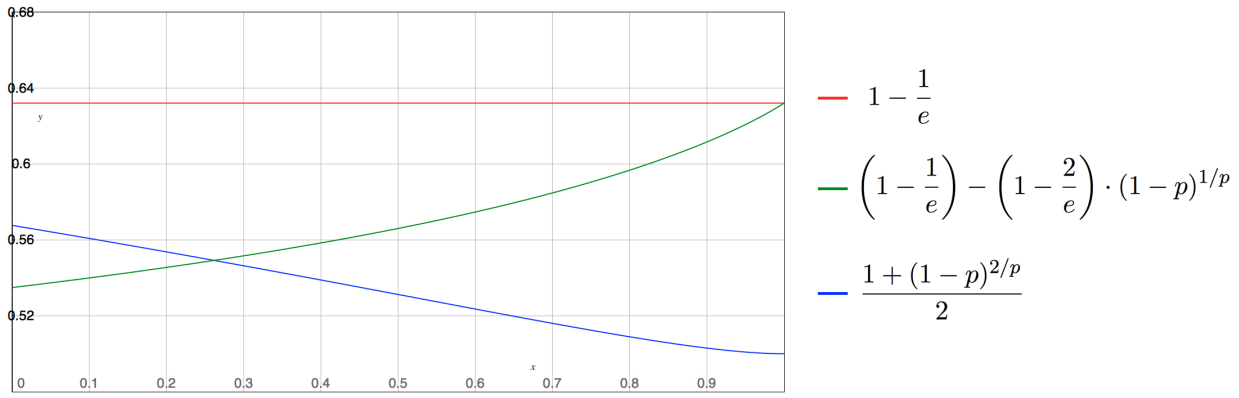

FIGURE 4.1. Shown above are the competitive ratios of RANKING (red), STOCHASTIC-BALANCE (green), and STOCHASTIC-RANKING (blue).

4.5. **Discussion.** The intersection of the two functions for the competitive ratios of STOCHASTIC-BALANCE and STOCHASTIC-RANKING shows that the latter performs better when $p < 0.26$. This is illustrated in Figure 4.1. Thus unlike the classic and weighted cases, a deterministic algorithm not only achieves a better competitive ratio than $1/2$ but also so far is known to perform better than the randomized algorithm as $p \to 0$. Also note that these competitive ratios are consistent with the reduction to the non-stochastic case where $p = 1$. In this case, STOCHASTIC-BALANCE has a competitive ratio of $1/2$ while STOCHASTIC-RANKING has a competitive ratio of $1 - 1/e$.

5. ACKNOWLEDGEMENTS

We would like to thank Professor Karger for a great semester. We would also like to thank Anvisha Pai, Fermi Ma, and Matt Jordan for their constructive feedback.

REFERENCES

[[KVV]] R. Karp, U. Vazirani, and V. Vazirani, "An optimal algorithm for online bipartite matching," in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990. (http://www.cs.berkeley.edu/~vazirani/pubs/online.pdf)

[[BM]] B. Birnbaum and C. Mathieu, "Online bipartite matching made simple," 2008. (http://cs.brown.edu/~claire/Publis/sigactnews08.pdf)

[[GM]] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on discrete algorithms, pages 982-991, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[[AGKM]] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta, "Online vertex-weighted bipartite matching and single-bid budgeted allocations," SODA, 2011. (arXiv:1007.1271 [cs.DS])

[[MP]] A. Mehta and D. Panigrahi, "Online matching with stochastic rewards," FOCS, 2012, pp. 728-737. (http://www.cs.duke.edu/~debmalya/papers/focs12-matching.pdf)

[[JMMSV]] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani, "Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp," J. ACM, vol. 50, no. 6, pp. 795-824, 2003.