Online Bipartite Matching with Decomposable Weights

Moses Charikar[1], Monika Henzinger[2], and Huy L. Nguyễn[1]

[1] Princeton University, USA, {moses, hlnguyen}@cs.princeton.edu
[2] University of Vienna, Austria, monika.henzinger@univie.ac.at

Abstract. We study a weighted online bipartite matching problem: $G(V_1, V_2, E)$ is a weighted bipartite graph where V_1 is known beforehand and the vertices of V_2 arrive online. The goal is to match vertices of V_2 as they arrive to vertices in V_1, so as to maximize the sum of weights of edges in the matching. If assignments to V_1 cannot be changed, no bounded competitive ratio is achievable. We study the weighted online matching problem with *free disposal*, where vertices in V_1 can be assigned multiple times, but only get credit for the maximum weight edge assigned to them over the course of the algorithm. For this problem, the greedy algorithm is 0.5-competitive and determining whether a better competitive ratio is achievable is a well known open problem.

We identify an interesting special case where the edge weights are decomposable as the product of two factors, one corresponding to each end point of the edge. This is analogous to the well studied related machines model in the scheduling literature, although the objective functions are different. For this case of decomposable edge weights, we design a 0.5664 competitive randomized algorithm in complete bipartite graphs. We show that such instances with decomposable weights are non-trivial by establishing upper bounds of 0.618 for deterministic and 0.8 for randomized algorithms.

A tight competitive ratio of $1 - 1/e \approx 0.632$ was known previously for both the 0-1 case as well as the case where edge weights depend on the offline vertices only, but for these cases, reassignments cannot change the quality of the solution. Beating 0.5 for weighted matching where reassignments are necessary has been a significant challenge. We thus give the first online algorithm with competitive ratio strictly better than 0.5 for a non-trivial case of weighted matching with free disposal.

1 Introduction

In recent years, online bipartite matching problems have been intensely studied. Matching itself is a fundamental optimization problem with several applications, such as matching medical students to residency programs, matching men and women, matching packets to outgoing links in a router and so on. There is a rich body of work on matching problems, yet there are basic problems we don't understand and we study one such question in this work. The study of the online

setting goes back to the seminal work of Karp, Vazirani and Vazirani [26] who gave an optimal $1 - 1/e$ competitive algorithm for the unweighted case. Here $G(V_1, V_2, E)$ is a bipartite graph where V_1 is known beforehand and the vertices of V_2 arrive online. The goal of the algorithm is to match vertices of V_2 as they arrive to vertices in V_1, so as to maximize the size of the matching.

In the weighted case, edges have weights and the goal is to maximize the sum of weights of edges in the matching. In the application of assigning ad impressions to advertisers in display advertisement, the weights could represent the (expected) value of an ad impression for an advertiser and the objective function for the maximum matching problem encodes the goal of assigning ad impressions to advertisers to as to maximize total value. If assignments to V_1 cannot be changed and if edge weights depend on the *online* node to which they are adjacent, it is easy to see that no competitive ratio bounded away from 0 is achievable.

Feldman et al [18] introduced the *free disposal* setting for weighted matching, where vertices in V_1 can be assigned multiple times, but only get credit for the maximum weight edge assigned to them over the course of the algorithm. (On the other hand, a vertex in V_2 can only be assigned at the time that it arrives with no later reassignments permitted). [18] argues that this is a realistic model for assigning ad impressions to advertisers. The greedy algorithm is 0.5 competitive for the online weighted matching problem with free disposal. They study the weighted matching problem with capacities – here each vertex $v \in V_1$ is associated with a capacity $n(v)$ and gets credit for the largest $n(v)$ edge weights from vertices in V_2 assigned to v. They designed an algorithm with competitive ratio approaching $1 - 1/e$ as the capacities approach infinity. Specifically, if all capacities are at least k, their algorithm gets competitive ratio $1 - 1/e_k$ where $e_k = (1 + 1/k)^k$. If all capacities are 1, their algorithm is 1/2-competitive.

Aggarwal et al [1] considered the online weighted bipartite matching problem where edge weights are only dependent on the end point in V_1, i.e. each vertex $v \in V_1$ has a weight $w(v)$ and the weight of all edges incident on v is $w(v)$. This is called the *vertex weighted setting*. They designed a $1 - 1/e$ competitive algorithm. Their algorithm can be viewed as a generalization of the Ranking algorithm of [26].

It is remarkable that some basic questions about a fundamental problem such as matching are still open in the online setting. Our work is motivated by the following tantalizing open problem; *Is it possible to achieve a competitive ratio better than* 0.5 *for weighted online matching ?* Currently no upper bound better than $1 - 1/e$ is known for the setting of general weights – in fact this bound holds even for the setting of 0-1 weights. On the other hand, no algorithm with competitive ratio better than 0.5 (achieved by the greedy algorithm) is known for this problem. By the results of [18], the case where the capacities are all 1 seems to be the hardest case and this is what we focus on.

1.1 Our results

We identify an interesting special case of this problem where we have a complete graph between V_1 and V_2 and the edge weights are decomposable as the product of two factors, one corresponding to each end point of the edge. This is analogous to the well studied related machines model in the scheduling literature [5, 6, 9, 17] where the load of a job of size p on a machine of speed s is p/s although the objective functions are different. Scheduling problems typically involving minimizing the maximum machine load (makespan) or minimizing the ℓ_p norm of machine loads, where the load on a machine is the sum of loads of all jobs placed on the machine. By contrast, in the problem we study, the objective (phrased in machine scheduling terminology) is to maximize the sum of machine loads where the load of a machine is the load of the largest job placed on the machine. For this case of decomposable edge weights, we design a 0.5664 competitive algorithm (Section 3). For display advertisement using a complete graph models the setting of a specific market segment (such as impressions for males between 20 and 30), where every advertiser is interested in every impression. The weight factor of the offline node u can model the value that a click has for advertiser u, the weight factor of the online node v can model the clickthrough probability of the user to which impression v is shown. Thus, the maximum weight matching in the setting we study corresponds to maximizing the sum of the expected values of all advertisers.

Our algorithm uses a now standard *randomized doubling technique* [8, 20, 12, 23]; however the analysis is novel and non-trivial. We perform a recursive analysis where each step proceeds as follows: We lower bound the profit that the algorithm derives from the fastest machine (i.e. the load of the largest job placed on it) relative to the difference between two optimum solutions - one corresponding to the original instance and the other corresponding to a modified instance obtained by removing this machine and all the jobs assigned to it. This is somewhat reminiscent of, but different from the local ratio technique used to design approximation algorithms. Finally, to exploit the randomness used by the algorithm we need to establish several structural properties of the worst case sequence of jobs – this is a departure from previous applications of this *randomized doubling technique*. While all previous online matching algorithms were analyzed using a *local*, step-by-step analysis, we use a *global* technique, i.e. we reason about the entire sequence of jobs at once. This might be useful for solving the case of online weighted matching for *general* weights. The algorithm and analysis is presented in Section 3 and an outline of the analysis is presented in Section 3.1.

A priori, it may seem that the setting of decomposable weights ought to be a much easier case of weighted online matching since it does not capture the well studied setting of 0-1 weights. We show that such instances with decomposable weights are non-trivial by establishing an upper bound of $(\sqrt{5}-1)/2 \approx 0.618$ on the competitive ratios of deterministic algorithms (Section 4) and an upper bound of 0.8 on the competitive ratio of randomized algorithms (Section 5). The deterministic upper bound constructs a sequence of jobs that is the solution to a

certain recurrence relation. Crucial to the success of this approach is a delicate choice of parameters to ensure that the solution of the recurrence is oscillatory (i.e. the roots are complex). In contrast to the setting with capacities, for which a deterministic algorithm with competitive ratio approaching $1 - 1/e \approx 0.632$ exists [18], our upper bound of $(\sqrt{5}-1)/2 < 1-1/e$ for deterministic algorithms shows that no such competitive ratio can be achieved for the decomposable case with unit capacities. Note that the upper bound of $1 - 1/e$ for the unweighted case [26] is for randomized algorithms and does not apply to the setting of decomposable weights that we study here.

In contrast to the vertex weighted setting (and the special case of 0-1 weights) where reassignments to vertices in V_1 cannot improve the quality of the solution, any algorithm for the decomposable weight setting must necessarily exploit reassignments in order to achieve a competitive ratio bounded away from 0. For this class of instances, we give an upper bound approaching 0.5 for the competitive ratio of the greedy algorithm. This shows that for decomposable weights greedy's performance cannot be better than for general weights, where it is 0.5-competitive (Section 2).

1.2 Related work

Goel and Mehta [21] and Birnbaum and Mathieu [10] simplified the analysis of the Ranking algorithm considerably. Devanur et al [15] recently gave an elegant randomized primal-dual interpretation of [26]; their framework also applies to the generalization to the vertex weighted setting by [1]. Haeupler et al [24] studied online weighted matching in the stochastic setting where vertices from V_2 are drawn from a known distribution. The stochastic setting had been previously studied in the context of unweighted bipartite matching in a sequence of papers [19, 28]. Recent work has also studied the random arrival model (for unweighted matching) where the order of arrival of vertices in V_2 is assumed to be a random permutation: In this setting, Karande, at al [25] and Mahdian and Yan [27] showed that the Ranking algorithm of [26] achieves a competitive ratio better than $1-1/e$. A couple of recent papers analyze the performance of a randomized greedy algorithm and an analog of the Ranking algorithm for matching in general graphs [32, 22]. Another recent paper introduces a stochastic model for online matching where the goal is to maximize the number of successful assignments (where success is governed by a stochastic process) [30].

A related model allowing cancellation of previously accepted online nodes was studied in [13, 7, 4] and optimal deterministic and randomized algorithms were given. In their setting the weight of an edge depends *only* on the online node. Additionally in their model they decide in an online fashion only which online nodes to accept, *not* how to match these nodes to offline nodes. If a previously accepted node is later rejected, a non-negative cost is incurred. Since the actual matching is only determined after all online nodes have been seen, their model is very different from ours: Even if the cost of rejection of a previously accepted node is set to 0, the key difference is that they do not commit to a matching

at every step and the intended matching can change dramatically from step to step. Thus, it does *not* solve the problem that we are studying.

A related problem that has been studied is online matching with preemption [29, 3, 16]. Here, the edges of a graph arrive online and the algorithm is required to maintain a subset of edges that form a matching. Previously selected edges can be rejected (preempted) in favor of newly arrived edges. This problem differs from the problem we study in two ways: (1) the graph is not necessarily bipartite, and (2) edges arrive one by one. In our (classic) case, vertices arrives online and all incident edges to a newly arrived vertex v are revealed when v arrives.

Another generalization of online bipartite matching is the Adwords problem [31, 14]. In addition, several online packing problems have been studied with applications to the Adwords and Display Advertisement problem [11, 21, 2].

1.3 Notation and preliminaries

We consider the following variant of the online bipartite matching problem. The input is a complete bipartite graph $G = (V_1 \cup V_2, V_1 \times V_2)$ along with two weight functions $s : V_1 \to \mathbb{R}_+$ and $w : V_2 \to \mathbb{R}_+$. The weight of each edge $e = (u, v)$ is the product $s(u) \cdot w(v)$. At the beginning, only s is given to the algorithm. Then, the vertices of V_2 arrive one by one. When a new vertex v arrives, $w(v)$ is revealed and the algorithm has to match it to a vertex in V_1. At the end, the reward of each vertex $u \in V_1$ is the maximum weight assigned to u times $s(u)$. The goal of the algorithm is to maximize the sum of the rewards. To simplify the presentation we will call vertices of V_1 *machines* and vertices of V_2 *jobs*. The s-value of a machine u will be called the *speed* of the machines and the w-value of a job v is called the *size* of the job. Thus, the goal of the online algorithm is to assign jobs to machines. However, we are not studying the "classic" variant of the problem since we are using a different optimization function, motivated by display advertisements.

2 Upper bound for the greedy algorithm

We begin by addressing an obvious question, which is how well a greedy approach would solve our problem, and using the proof to provide some intuition for our algorithm in the next section. We analyze here the following simple greedy algorithm: When a job v arrives, the algorithm computes for every machine u the difference between the weight of (u, v) and the weight (u, v'), where v' is the job currently assigned to u. If this difference is positive for at least one machine, the job is assigned to a machine with maximum difference.

Theorem 1. *The competitive ratio of the greedy algorithm is at most* $\frac{1}{2-\epsilon}$ *for any* $\epsilon > 0$.

Proof. Consider the following instance. V_1 consists of a vertex a with $s(a) = 1$ and $t = 1/\epsilon^2$ vertices $b_1, \ldots, b_t$ with $s(b_i) = \epsilon/2 \ \forall i$. V_2 consists of the following vertices arriving in the same order $d_1, \ldots, d_{1+t}$ where $w(d_i) = (1-\epsilon/2)^{-i}$. We will

prove by induction that all vertices d_i are assigned to a. When d_1 arrives, nothing is assigned so it is assigned to a. Assume that all the first t vertices are assigned to a when d_{t+1} arrives. The gain by assigning d_{i+1} to a is $(w(d_{i+1})-w(d_i))s(a) = \epsilon(1-\epsilon/2)^{-i-1}/2$. The gain by assigning d_{i+1} to some b_j is $w(d_{i+1})s(b_j) = \epsilon(1-\epsilon/2)^{-i-1}/2$. Thus, the algorithm can assign d_{i+1} to a. The total reward of the algorithm is $(1-\epsilon/2)^{-1-t}$. The optimal solution is to assign d_{1+t} to a and the rest to b_i's, getting $(1-\epsilon/2)^{-1-t} + (1-\epsilon/2)^{-t} - 1 \geq (2-\epsilon)(1-\epsilon/2)^{-1-t}$. Thus, the competitive ratio is at most $\frac{1}{2-\epsilon}$. □

The instance used in the proof above suggests some of the complications an algorithm has to deal with in the setting of decomposable weights: in order to have competitive ratio bounded away from 0.5, an online algorithm must necessarily place some jobs on the slow machines. In fact it is possible to design an algorithm with competitive ratio bounded away from 0.5 for the specific set of machines used in this proof (for any sequence of jobs). The idea is to ensure that a job is placed on the fast machine only if its size is larger than $(1+\gamma)$ times the size of the largest job currently on the fast machine (for an appropriately chosen parameter γ). Such a strategy works for any set of machines consisting of one fast machine and several slow machines of the same speed. However, we do not know how to generalize this approach to an arbitrary set of machines. Still, this strategy (i.e. ensuring that jobs placed on a machine increase in size geometrically) was one of the motivations behind the design of the randomized online algorithm to be presented next.

3 Randomized algorithm

We now describe our randomized algorithm which uses a parameter c we will specify later: The algorithm picks values $x_i \in (0,1]$ uniformly and at random, independently for each machine i. Each job of weight w considered by machine i is placed in the unique interval $w \in (c^{k+x_i}, c^{k+1+x_i}]$ where k ranges over all integers. When a new job w arrives, the algorithm checks the machines in the order of decreasing speed (with ties broken in an arbitrary but fixed way). For machine i it first determines the unique interval into which w falls, which depends on its choice of x_i. If the machine currently does not have a job in this or a bigger interval (with larger k), w is assigned to i and the algorithm stops, otherwise the algorithm checks the next machine.

The following function arises in our analysis:

Definition 1. *Define* $h(c) = 1 - \frac{1}{\beta} W\left(\frac{\beta e^{\beta}}{c}\right)$ *where* $\beta = \frac{c\ln(c)}{c-1} - 1$ *and* $W()$ *is the Lambert W function (i.e. inverse of* $f(x) = xe^x$*).*

We will prove the following theorem:

Theorem 2. *For* $c \geq e$*, the randomized algorithm has competitive ratio* $\min\left(\frac{c-1}{c\ln(c)}, h(c)\right)$*. In particular, for* $c = 3.55829$*, the randomized algorithm has a competitive ratio* 0.5664*.*

3.1 Analysis Outline

We briefly outline the analysis strategy before describing the details. An instance of the problem consists of a set of jobs and a set of machines. The (offline) optimal solution to an instance is obtained by ordering machines from fastest to slowest, ordering jobs from largest to smallest and assigning the ith largest job to the ith fastest machine. Say the machines are numbered $1, 2, \ldots n$, from fastest to slowest. Let OPT_i denote the value of the optimal solution for the instance seen by the machines from i onwards, i.e. the instance consisting of machines $i, i+1, \ldots n$, and the set of jobs passed by the $(i-1)$st machine to the ith machine in the online algorithm. Then $OPT_1 = OPT$, the value of the optimal solution for the original instance. Even though we defined OPT_i to be the value of the optimal solution, we will sometimes use OPT_i to denote the optimal assignment, although the meaning will be clear from context. Define OPT_{n+1} to be 0. For $2 \leq i \leq n$, OPT_i is a random variable that depends on the random values $x_{i'}$ picked by the algorithm for $i' < i$. In the analysis, we will define random variables Δ_i such that $\Delta_i \geq OPT_i - OPT_{i+1}$ (see Lemma 1 later). Let A_i denote the profit of the online algorithm derived from machine i (i.e. the size of the largest job assigned to machine i times the speed of machine i). Let $A = \sum_{i=1}^{n} A_i$ be the value of the solution produced by the online algorithm. We will prove that for $1 \leq i \leq n$,

$$\mathbb{E}[A_i] \geq \alpha\, \mathbb{E}[\Delta_i] \geq \alpha(\mathbb{E}[OPT_i] - \mathbb{E}[OPT_{i+1}]) \tag{1}$$

for a suitable choice of $\alpha > 0.5$. The expectations in (1) are taken over the random choices of machine $1, \ldots i$. Note that $OPT_i - OPT_{i+1}$ is a random variable, but the sum of these quantities for $1 \leq i \leq n$ is $OPT_1 - OPT_{n+1} = OPT$, a deterministic quantity. Summing up (1) over $i = 1, \ldots n$, we get $\mathbb{E}[A] \geq \alpha \cdot OPT$, proving that the algorithm gives an α approximation.

Inequality (1) applies to a recursive application of the algorithm to the subinstance consisting of machines $i, \ldots n$ and the jobs passed from machine $i-1$ to machine i. The subinstance is a function of the random choices made by the first $i-1$ machines. We will prove that for any instance of the random choices made by the first $i-1$ machines,

$$\mathbb{E}[A_i] \geq \alpha\, \mathbb{E}[\Delta_i]. \tag{2}$$

Here, the expectation is taken over the random choice of machine i. (2) immediately implies (1) by taking expectation over the random choices made by the first $i-1$ machines.

We need to establish (2). In fact, it suffices to do this for $i = 1$ and the proof applies to all values of i since (2) is a statement about a recursive application of the algorithm.

Wlog, we normalize so that the fastest machine has speed 1 and the largest job is c. Note that this is done by simply multiplying all machine speeds by a suitable factor and all job sizes by a suitable factor – both the LHS and the RHS of (2) are scaled by the same quantity.

In order to compare Δ_1 with the profit of the algorithm, we decompose the instance into a convex combination of simpler threshold instances in Lemma 4. Here, the speeds are either all the same or take only two different values, 0 and 1. It suffices to compare the profit of the algorithm to OPT on such threshold instances.

Intuitively, if there are so few fast machines that even a relatively large job (job of weight at least 1) got assigned to a slow machine in OPT, then the original instance is mostly comparable to the threshold instance where only a few machines have speed 1 and the rest have speed 0. Even if the fastest machine gets jobs assigned to machines of speed 0 in OPT, this does not affect the profit of the algorithm relative to OPT because OPT does not profit from these jobs either. Thus we only care about jobs of weight at least 1. Because a single machine can get at most two jobs of value in the range $[1, c]$, handling this case only requires analyzing at most two jobs. The proof for this case is contained in Lemma 3.

On the other hand, if there are a lot of fast machines so that all large jobs are assigned to fast machines in OPT, then the original instance is comparable to the threshold instance where all machines have speed 1. In this case, the fastest machine can get assigned many jobs that all contribute to OPT. However, because all speeds are the same, we can deduce the worst possible sequence of jobs: after the first few jobs, all other jobs have weights forming a geometric sequence. The rest of the proof is to analyze the algorithm on this specific sequence. The detailed proof is contained in Lemma 4.

The proofs of both Lemmata 3 and 4 use the decomposable structure of the edge weights.

3.2 Analysis Details

Recall that OPT_1 is the value of the optimal solution for the instance, and OPT_2 is the value of the optimal solution for the subinstance seen by machine 2 onwards. Assume wlog that all job sizes are distinct (by perturbing job sizes infinitesimally). For $y \leq c$, let $j(y)$ be the size of the largest job $\leq y$ or 0, if no such job exists. Let $s(y)$ be the speed of the machine in the optimal solution that $j(y)$ is assigned to or 0 if $j(y) = 0$. If there is a job of size y then $s(y)$ is the speed of the machine in the optimal solution that this job is assigned to. Note that $s(y) \in [0, 1]$ is monotone increasing with $s(c) = 1$. We refer to the function s as the *speed profile*. Note that s is not a random variable. Let the *assignment sequence* $\mathbf{w} = (w, w_1, w_2, \ldots)$ denote the set of jobs assigned to the fastest machine by the algorithm where $w > w_1 > w_2 > \ldots$. Let $\max(\mathbf{w})$ denote the maximum element in the sequence $\mathbf{w}$, i.e. $\max(\mathbf{w}) = w$. In Lemma 3, we bound $OPT_1 - OPT_2$ by a function that depends *only* on $\mathbf{w}$, s, and c. Such a bound is possible because of the fact that any job can be assigned to any machine, i.e. the graph is a complete graph. The value we take for the aforementioned random variable Δ_1 turns out to be exactly this bound.

Lemma 1. $OPT_1 - OPT_2 \leq c - (c - w)s(w) + \sum_{k \geq 1} w_k \cdot s(w_k)$

Proof. Let I_1, I_2 be the instances corresponding to OPT_1 and OPT_2. I_2 is obtained from I_1 by removing the fastest machine and the set of jobs that are assigned to the fastest machine by the algorithm. Let us consider changing I_1 to I_2 in two steps: (1) Remove the fastest machine and the largest job w assigned by the algorithm to the fastest machine. (2) Remove the jobs $w_1, w_2, \ldots$. For each step, we will bound the change in the value of the optimal solution resulting in a feasible solution for I_2 and computing its value – this will be a lower bound for OPT_2.

First we analyze Step 1: OPT_1 assigns the largest job c to the fastest machine, contributing c to its value. The algorithm assigns w to the fastest machine instead of c. In OPT_1, w was assigned to a machine of speed $s(w)$. When we remove w and the fastest machine from I_1, one possible assignment to the resulting instance is obtained by placing c on the machine of speed $s(w)$. The value of the resulting solution is lower by exactly $(c + w \cdot s(w)) - c \cdot s(w) = c - (c - w)s(w)$.

Next, we analyze Step 2: Jobs $w_1, w_2, \ldots$ were assigned to machines of speeds $s(w_1), s(w_2), \ldots$ in OPT_1. When we remove jobs $w_1, w_2, \ldots$, one feasible assignment for the resulting instance is simply not to assign any jobs to the machines $s(w_1), s(w_2), \ldots$, and keep all other assignments unchanged. The value of the solution drops by exactly $\sum_{k \geq 1} w_k \cdot s(w_k)$.

Thus we exhibited a feasible solution to instance I_2 of value V where

$$OPT_1 - V = c - (c - w)s(w) + \sum_{k \geq 1} w_k \cdot s(w_k).$$

But $OPT_2 \geq V$. Hence, the lemma follows. □

We define the random variable Δ_1, a function of the assignment sequence $\mathbf{w}$ and the speed profile s, to be

$$\Delta_1(\mathbf{w}, s) = c - (c - w)s(w) + \sum_{k \geq 1} w_k \cdot s(w_k).$$

As defined, $\Delta_1(\mathbf{w}, s) \geq OPT_1 - OPT_2$. We note that even though OPT_1 and OPT_2 are functions of all the jobs in the instance, Δ_1 only depends on the subset of jobs assigned to the fastest machine by the algorithm. Our goal is to show $\mathbb{E}[A_1] = E[\max(\mathbf{w})] \geq \alpha\, \mathbb{E}[\Delta_1]$.

First, we argue that it suffices to restrict our analysis to a simple set of step function speed profiles s_t, $0 \leq t \leq 1$: For $t \in (0, 1]$, $s_t(y) = 1$ for $y \in [c^t, c]$ and $s_t(y) = 0$ for $y < c^t$. For $t = 0$, $s_0(y) = 1$ for all $y \leq c$. The proof is omitted.

Lemma 2. *Suppose that for $t = 0$ and for all $t \in (0, 1]$ such that there exists a job of weight c^t, we have*

$$\mathbb{E}[\max(\mathbf{w})] \geq \alpha\, \mathbb{E}[\Delta_1(\mathbf{w}, s_t)] \tag{3}$$

Then, $\mathbb{E}[\max(\mathbf{w})] \geq \alpha(\mathbb{E}[OPT_1] - \mathbb{E}[OPT_2])$.

Note that since we scaled job sizes, the thresholds (i.e interval boundaries) c^{k+x_1} should also be scaled by the same quantity (say γ). After scaling, let

$x \in (0,1]$ be such that c^x is the unique threshold from the set $\{\gamma c^{k+x_1}, k \text{ integer}\}$ in $(1,c]$. Since x_1 is uniformly distributed in $(0,1]$, x is also uniformly distributed in $(0,1]$. Having defined x thus, the interval boundaries picked by the algorithm for the fastest machine are c^{x+k} for integers k.

We prove (3) for $\alpha = \min\left(\frac{c-1}{c\ln(c)}, h(c)\right)$ in two separate lemmata, one for the case $t > 0$ (Lemma 3) and the other for the case $t = 0$ (Lemma 4). Recall that the expression for Δ_1 only depends on the subset of jobs assigned to the fastest machine. We call a job a *local maximum* if it is larger than all jobs preceding it. Since the algorithm assigns a new job to the fastest machine if and only if it falls in a larger interval than the current largest job, it follows that any job assigned to the fastest machine must be a local maximum.

Define $m_S(y)$ to be the minimum job in the sequence of all local maxima in the range $(y, cy]$, i.e., the first job larger than y and at most cy, if such a job exists and 0 otherwise. We use $m_S(y)$ in two ways. (1) We define $u_0 = m_S(1)$. Note that u_0 is not a random variable. We use u_0 in Lemma 3 to prove the desired statement for $t > 0$. Specifically, we use u_0 to compute (i) a lower bound for $\mathbb{E}[w]$ as a function of u_0 (and not of any other jobs) and (ii) an upper bound for $\mathbb{E}[\Delta_1]$ as a function of u_0. Combining (i) and (ii) we prove that the desired inequality holds for all u_0. (2) In Lemma 4 we bound $\mathbb{E}[\sum_{k\geq 1} w_k]$ by a sum of $m_S(y)$ over suitable values of y. This simplifies the analysis since the elements in the subsequence of *all* local maxima are *not* random variables, while the values in $\mathbf{w}$ are random variables.

We first prove some simple properties of u_0 that we will use:

Claim. (1) $u_0 \leq w$ and (2) $u_0 \geq w_1$.

Proof. $u_0 \leq w$ as u_0 is the minimum element in the sequence of all local maxima in $(1,c]$ and w is the element from the interval $(1,c]$ picked by the algorithm.

w_1 is the minimum element in the sequence of local maxima in the range $(c^{x-k-1}, c^{x-k}]$ for $x \in (0,1]$ and k a non-negative integer. Either $u_0 \geq c^{x-k} \geq w_1$, or u_0 also falls into $(c^{x-k-1}, c^{x-k}]$ and $u_0 \geq w_1$ follows from the fact that w_1 is the smallest local maximum in this range, while u_0 is an arbitrary local maximum in this range. □

The next lemmata conclude our algorithm analysis. Proofs are omitted.

Lemma 3. *For $c \geq e$, $t \in (0,1]$ such that there exists a job of weight c^t, and $\alpha = \min\left(\frac{c-1}{c\ln(c)}, h(c)\right)$, we have*

$$\alpha\, \mathbb{E}[\Delta_1(\mathbf{w}, s_t)] \leq \mathbb{E}[\max(\mathbf{w})]$$

Lemma 4. *For $s_0(x) \equiv 1$ and $\alpha = h(c)$, we have $\mathbb{E}[\max(\mathbf{w})] \geq \alpha\, \mathbb{E}[\Delta_1(\mathbf{w}, s_0)]$.*

4 Upper bound for deterministic algorithms

To prove an upper bound of a, we construct an instance such that any deterministic algorithm has competitive ratio at most a for some prefix of the request

sequence. The instance has one *fast* machine of speed $r > 1$ and n *slow* machines of speed 1. The request sequence has non-decreasing job sizes satisfying a certain oscillatory recurrence relation. The full proof is in the full version.

Theorem 3. *The competitive ratio of any deterministic algorithm is at most* $(\sqrt{5}-1)/2 + \epsilon \approx 0.618034 + \epsilon$ *for any* $\epsilon > 0$.

5 Upper bound for randomized algorithms

To establish the bound for randomized algorithms, we use Yao's principle and show an upper bound on the expected competitive ratio of any deterministic algorithm on a distribution of instances. The construction uses one fast machine of speed 1 and n slow machines of speed $1/4$. The request sequence has non-decreasing sizes 2^i. The prefix of this sequence ending with size 2^i is presented to the algorithm with probability $c/2^i$, where c is a normalizing constant. We show that the best algorithm for this sequence achieves at most $cn + 1$ while the optimal algorithm achieves roughly $5nc/4$. The proof is in the full version.

Theorem 4. *The competitive ratio of any randomized algorithm against an oblivious adversary is at most* $0.8 + \epsilon$ *for any* $\epsilon > 0$.

Acknowledgments. MC was supported by NSF awards CCF 0832797, AF 0916218 and a Google research award. MH's support: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506 and the Austrian Science Fund (FWF) grant P23499-N23. HN was supported by NSF awards CCF 0832797, AF 0916218, a Google research award, and a Gordon Wu fellowship.

References

1. G. Aggarwal, G. Goel, C. Karande, and A. Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *SODA*, pages 1253–1264, 2011.
2. S. Agrawal, Z. Wang, and Y. Ye. A dynamic near-optimal algorithm for online linear programming. *CoRR*, abs/0911.2974, 2009.
3. B. V. Ashwinkumar. Buyback problem-approximate matroid intersection with cancellation costs. *Automata, Languages and Programming*, pages 379–390, 2011.
4. B. V. Ashwinkumar and R. Kleinberg. Randomized online algorithms for the buyback problem. In *WINE*, pages 529–536, 2009.
5. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM (JACM)*, 44(3):486–504, 1997.
6. Y. Azar. On-line load balancing. *Online Algorithms*, pages 178–195, 1998.
7. M. Babaioff, J. D. Hartline, and R. D. Kleinberg. Selling ad campaigns: online algorithms with cancellations. In *EC*, pages 61–70, 2009.
8. A. Beck and D. Newman. Yet more on the linear search problem. *Israel journal of mathematics*, 8(4):419–429, 1970.

9. P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1):108–121, 2000.
10. B. E. Birnbaum and C. Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008.
11. N. Buchbinder, K. Jain, and J. Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *ESA*, pages 253–264, 2007.
12. S. Chakrabarti, C. Phillips, A. Schulz, D. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. *Automata, Languages and Programming*, pages 646–657, 1996.
13. F. Constantin, J. Feldman, S. Muthukrishnan, and M. Pál. An online mechanism for ad slot reservations with cancellations. In *SODA*, pages 1265–1274, 2009.
14. N. R. Devanur and T. P. Hayes. The adwords problem: Online keyword matching with budgeted bidders under random permutations. In *EC*, pages 71–78, 2009.
15. N. R. Devanur, K. Jain, and R. Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *SODA*, 2013.
16. L. Epstein, A. Levin, D. Segev, and O. Weimann. Improved bounds for online preemptive matching. *CoRR*, abs/1207.1788, 2012.
17. L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, 26(1):17–22, 2000.
18. J. Feldman, N. Korula, V. S. Mirrokni, S. Muthukrishnan, and M. Pál. Online ad assignment with free disposal. In *WINE*, pages 374–385, 2009.
19. J. Feldman, A. Mehta, V. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 117–126. IEEE, 2009.
20. S. Gal. Search games, volume 149 of mathematics in science and engeneering, 1980.
21. G. Goel and A. Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, pages 982–991, 2008.
22. G. Goel and P. Tripathi. Matching with our eyes closed. In *FOCS*, pages 718–727, 2012.
23. M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82(1):111–124, 1998.
24. B. Haeupler, V. S. Mirrokni, and M. Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *WINE*, pages 170–181, 2011.
25. C. Karande, A. Mehta, and P. Tripathi. Online bipartite matching with unknown distributions. In *STOC*, pages 587–596, 2011.
26. R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, pages 352–358, 1990.
27. M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *STOC*, pages 597–606, 2011.
28. V. Manshadi, S. Gharan, and A. Saberi. Online stochastic matching: Online actions based on offline statistics. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1285–1294. SIAM, 2011.
29. A. McGregor. Finding graph matchings in data streams. *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 611–612, 2005.
30. A. Mehta and D. Panigrahi. Online matching with stochastic rewards. In *FOCS*, pages 728–737, 2012.
31. A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani. Adwords and generalized on-line matching. In *FOCS*, pages 264–273, 2005.
32. M. Poloczek and M. Szegedy. Randomized greedy algorithms for the maximum matching problem with new analysis. In *FOCS*, pages 708–717, 2012.