Online Load and Graph Balancing for Random Order Inputs

Sungjin Im
University of California Merced
Merced, USA
sim3@ucmerced.edu

Ravi Kumar
Google Research
Mountain View, USA
ravi.k53@gmail.com

Shi Li
Nanjing University
Nanjing, China
shili@nju.edu.cn

Aditya Petety
University of California Merced
Merced, USA
apetety@ucmerced.edu

Manish Purohit
Google Research
Mountain View, USA
mpurohit@google.com

ABSTRACT

Online load balancing for heterogeneous machines aims to minimize the makespan (maximum machine workload) by scheduling arriving jobs with varying sizes on different machines. In the adversarial setting, where an adversary chooses not only the collection of job sizes but also their arrival order, the problem is well-understood and the optimal competitive ratio is known to be $\Theta(\log m)$, where m is the number of machines. In the more realistic random arrival order model, the understanding is limited. Previously, the best lower bound on the competitive ratio was only $\Omega(\log \log m)$.

We significantly improve this bound by showing an $\Omega(\sqrt{\log m})$ lower bound, even for the restricted case where each job has a unit size on two machines and infinite size on the others. On the positive side, we propose an $O(\log m/\log \log m)$-competitive algorithm, demonstrating that better performance is possible in the random arrival order model.

CCS CONCEPTS

• **Theory of computation** → **Scheduling algorithms**.

KEYWORDS

Scheduling, Load Balancing, Random Order

ACM Reference Format:
Sungjin Im, Ravi Kumar, Shi Li, Aditya Petety, and Manish Purohit. 2024. Online Load and Graph Balancing for Random Order Inputs . In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '24), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626183.3659983

1 INTRODUCTION

Online load balancing is a fundamental problem encountered in parallel/distributed computing and network communication, appearing in various forms. It focuses on effectively distributing limited resources to handle tasks that arrive over time (online). Due to its importance, this problem has been extensively studied in both practice and theory [5, 15, 22].

The model of unrelated machines has received significant attention due to its effective representation of heterogeneous processing capabilities. In this setting, the processing time of a job can differ based on the assigned machine. Minimizing the makespan, the maximum workload across all machines, is one of the most common objectives in this realm.

In many practical scenarios, the sequence of job arrivals is unpredictable, necessitating online load balancing strategies. For minimizing makespan in this online setting, an $O(\log m)$-competitive algorithm is known [3, 7], where m represents the number of machines. This algorithm guarantees a solution within a logarithmic factor of the optimal offline solution, which has full knowledge of the entire job sequence in advance. Notably, this competitive ratio is known to be tight under adversarial job arrival orders [7], where the algorithm has no information about future jobs.

One way to circumvent the strong lower bounds is to consider the random arrival order model. In this model, the adversary first defines the machines and jobs, then the jobs arrive in a random permutation. The appeal of this model is its simplicity: no assumptions are made about the overall input, and only the order of arrival is randomized. Crucially, the optimal offline solution remains unchanged. The random arrival model has been exceptionally effective in breaking many lower bound barriers, as seen in the secretary problem, packing integer problem, online facility location problem, and many others. For a comprehensive overview of the results in the random arrival model, see [20].

Despite the success of the random arrival model in other contexts, research on online load balancing within this framework remains surprisingly thin. To the best of our knowledge, the only non-trivial result for this problem is a lower bound of $\Omega(\log \log m)$ [9].

1.1 Our Results

In this paper, we make significant strides in online load balancing for unrelated machines with the random arrival model by establishing the following new bounds.

Lower Bounds. We show lower bounds for the special case of graph balancing [14]. Here, edges representing jobs arrive online, and we must immediately orient each arriving edge towards one of its endpoints. The objective is to minimize the maximum in-degree of any vertex, i.e., the highest number of incoming edges for a single vertex. This problem is equivalent to the online load

SPAA '24, June 17–21, 2024, Nantes, France

ACM ISBN 979-8-4007-0416-1/24/06…
https://doi.org/10.1145/3626183.3659983

balancing scenario where each job has a unit size on exactly two machines and an infinite size on the rest.

Even in this restricted setting, we establish a new lower bound of $\Omega(\sqrt{\log m})$ for any randomized algorithm, where m denotes the number of machines (or the number of vertices in the graph balancing problem). This is an exponential improvement over the previously known lower bound of $\Omega(\log\log m)$. Interestingly, our bound holds even when the instance is a tree and the algorithm knows the structure a priori. This result is presented in Section 4.1.

We further demonstrate that the intuitive greedy algorithm, which assigns each job to the less loaded machine (breaking ties randomly), has a competitive ratio of $\Omega(\log m/\log\log m)$. Expressed in graph balancing terminology, this algorithm orients each edge towards the endpoint with the lower in-degree. This highlights the necessity of deviating from the straightforward greedy approach to achieve a competitive ratio substantially better than the $\Theta(\log m)$ bound, which is the best possible in the adversarial order setting. This result can be found in Section 4.2.

Upper Bounds. On the positive side, we present an algorithm for online load balancing in the random arrival order model that achieves a competitive ratio of $O(\log m/\log\log m)$. Notably, this result applies to the general setting of unrelated machines. While the improvement over the $O(\log m)$ bound in the adversarial model is modest, it nevertheless establishes a meaningful separation between the two models for this problem. This result is in Section 3.

Remark. In the online makespan minimization for unrelated machines problem, we use m to denote the number of machines and n the number of jobs. This is the standard notation used in the scheduling literature. However, in the graph balancing problem, m is the number of edges and n is the number of nodes. Since nodes correspond to machines, a lower bound $f(n)$ for the graph balancing implies a lower bound $f(m)$ for makespan minimization. In fact, this differentiation is not critical as all lower bound instances we use for the graph balancing are trees; thus, we have $n = m + 1$.

1.2 Our Techniques

As a warm-up, we review an instance that demonstrates a lower bound of $\Omega(\log m)$ on the competitive ratio of any deterministic algorithm for adversarial arrivals. Consider a setting with machines indexed from 1 to m, where m is assumed to be a power of 2 for simplicity. Jobs arrive in rounds, with each job having a size of 1 and being assignable to only two specific machines. We can represent this constraint by representing jobs as pairs of machine indices. For example, a pair $(1, 2)$ indicates that the job can be assigned to either machine 1 or machine 2.

The instance unfolds as follows. In the first round, jobs with pairs $(1, 2), (3, 4), \ldots, (m-1, m)$ arrive. Without loss of generality, assume the algorithm assigns these jobs to the even-indexed machines. The subsequently arriving jobs can only be assigned to the even-indexed machines, which all already have one job. By focusing on the even-indexed machines and halving their indices, we effectively reduce the problem to $m/2$ machines. Each recursion of this process increases the load on the machines considered in the current round by 1. Therefore in the $(\log m)$th round, the algorithm's makespan reaches $\log m$, while an optimal solution has a makespan of 1, establishing the $\Omega(\log m)$ lower bound.

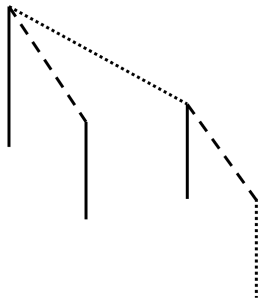

Figure 1: Illustration of a lower bound instance giving $\Omega(\log m)$ lower bound in the adversarial arrival model. Edges arrive in the order of solid, dashed, and dotted, and they are wlog assumed to be oriented towards the root.

This instance also can be viewed as a tree. See Figure 1.

Lower Bounds. In the random arrival order model, orchestrated job sequences that intentionally concentrate load on specific machines are no longer viable, as we cannot dictate the order in which jobs appear. While the aforementioned instance cannot establish a lower bound for arbitrary algorithms within this model, a minor tweak of it enables us to demonstrate a lower bound of $\Omega(\log m/\log\log m)$ for the intuitive greedy algorithm, which assigns each job to the machine with the lesser current load.

To 'reproduce' the adversarial order even in the random arrival order model, we use a 'fat' tree where every node has a polylogarithmic degree except the leaf nodes; here the tree has an $\Omega(\log m/\log\log m)$ height. The key idea is that in the fat tree, a majority of leaf edges (jobs) arrive before the other edges and we can show that this bad event can occur recursively with a high probability, resulting a high load on the root node.

Unfortunately, we cannot use the same fat tree to show a strong lower bound for arbitrary algorithms. This is because of the following reason. To obtain a good load balancing, it is important to orient edges towards the leaf nodes. However, it is easy for an algorithm to distinguish the leaf nodes from the others early in the random order model because the non-leaf nodes have high degrees.

To overcome this issue, our recursive construction is done more carefully. The main property the tree has is that when an edge arrives online, no algorithm should be able to distinguish between the two end points as to which is the parent node for a large number of edges. However, this lower bound instant construction comes at a cost. The tree become significantly fatter and has a height $\Theta(\sqrt{\log m})$, which translates to our lower bound.

Upper Bound. For the upper bound on makespan minimization for unrelated machines, we draw inspiration for the algorithm from [29]. The idea is to keep track of a potential function over all machines and reset these potentials after half of the inputs have arrived [19]. The potential function is essentially the softmax function of the machine loads. The algorithm then assigns an arriving job to the machine that incurs the least increase in the potential. The

algorithm is identical in both phases so it suffices to analyze only one half. This restarting helps as it ensures sufficient randomness for jobs that arrive later, which is crucial for the analysis.

1.3 Other Related Work

As mentioned before, online load balancing finds numerous applications in parallel/distributed computing and network communication. Thus, a vast literature exists on this topic [5, 11, 15, 22, 24] and we only cover the most related work.

The online load balancing problem was introduced by Graham in his seminal work [16, 17]. The work showed the list scheduling algorithm that assign an arriving job to the least loaded machine is $(2-1/m)$-competitive when all machines are identical. The current best known competitive ratio for this problem is 1.916 [1]. When the machines are uniformly related, meaning that machines have different speeds, $O(1)$-competitive algorithms are known [8]. For unrelated machines, $O(q)$-competitive algorithms are known for minimizing the q-norm of machine loads [4, 12]. For extension to multidimensional load balancing, see [6, 21, 28].

There has been a load of work on beyond-worst-case analysis in the recent years. The random arrival order model is a prime example of this framework. Here, we present a selection of recent advances within this model. For the online load balancing problem on identical machines, when jobs arrive in random order, there is a 1.8478-competitive algorithm [2] (which is better than known lower bounds in the adversarial setting). When the elements are revealed in a random order, the online set cover admits a competitive ratio of $O(\log mn)$ [18], which matches the offline bound of $O(\log n)$ when m is polynomial in n; here m is the number of sets and n is the number of elements. The online facility location problem admits a competitive ratio of 3 in the random order model [23]. As mentioned, the reader is referred to the survey [20] for work in the random arrival order.

Molinaro [29] provides algorithms that are simultaneously good both in the adversarial arrival model and in the random arrival model. In particular, for the makespan objective, the work gives an algorithm that is $\Theta(\log m/\epsilon)$-competitive in the adversarial order, while simultaneously giving a makespan of $(1+\epsilon)\textsc{opt} + O(m\log m/\epsilon^2)$ in random order. Unfortunately, the analysis requires $\epsilon \in (0,1]$ and therefore does not give a competitive ratio better than $O(\log m)$.

Online load balancing has been recently studied in the presence of ML predictions. The algorithms are given certain compact predictions on the input and can achieve competitive ratios significantly better than $O(\log m)$ when the predictions are almost accurate while asymptotically retaining the worst case guarantees [25, 27].

We now discuss related work on the offline load balancing problems. Offline, the load balancing problem for unrelated machines does not admit a better than 1.5-approximation unless P = NP and a 2-approximation is known for the problem due to the seminal work by Lenstra et al. [26]. For the graph balancing problem, when G is known offline, [13] gives a 1.75-approximation.

2 PRELIMINARIES AND NOTATION

We consider the online makespan minimization problem on unrelated machines. Let $\mathcal{J}$ be a set of n jobs and let $\mathcal{M}$ be a set of m machines. Let $p_{i,j} \in \mathbb{R}^+ \cup \{\infty\}$ denote the processing load of job j on machine i; in the online setting the values of $\{p_{i,j}\}$ for a job j become known to the algorithm when job j arrives. For an assignment $\sigma : \mathcal{J} \to \mathcal{M}$ of jobs to machines, we define its *makespan* as $\text{makespan}(\sigma) = \max_{i\in\mathcal{M}} \sum_{j\in\sigma^{-1}(i)} p_{i,j}$. The goal of an online algorithm is to find an assignment σ that minimizes $\text{makespan}(\sigma)$.

In this paper we consider the online setting with uniformly random job arrivals. Let $\mathcal{A}$ be an online algorithm. Let π be a uniformly random permutation of jobs in $\mathcal{J}$. At each time step $t \in [n]$, a job $\pi(t) \in \mathcal{J}$ arrives and $\mathcal{A}$ needs to assign it to exactly one machine in $\mathcal{M}$; let $\text{makespan}(\mathcal{A}; \{p_{i,j}\}, \pi)$ be the random variable denoting its makespan. The number of jobs, n, is known to the algorithm before any of the jobs arrive.

Let σ^* be the assignment of an optimal, offline algorithm. For any algorithm $\mathcal{A}$, we define its competitive ratio as the worst-case ratio of the expected makespan incurred by the algorithm to the optimal makespan, i.e.,

$$\text{competitive-ratio}(\mathcal{A}) = \max_{\{p_{i,j}\}} \frac{\mathbb{E}_\pi[\text{makespan}(\mathcal{A}; \{p_{i,j}\}, \pi)]}{\text{makespan}(\sigma^*)}.$$

Here, we will parameterize the competitive ratio by the number of machines, m. So, the competitive ratio is defined over the instances that have at most m machines.

We also consider a very special case of the makespan minimization problem called *graph balancing*. Let $G = (V, E)$ be an undirected graph. The goal is to orient the edges of the graph G so as to minimize $\max_{v\in V}$ indegree(v). It can be readily seen that this is a special case of makespan minimization on unrelated machines by considering each vertex as a machine and an edge as a job where each job has a unit load on exactly two machines and an infinite load otherwise. Again, in this paper, we study the graph balancing problem with edges arriving uniformly at random.

3 UNRELATED MACHINES: UPPER BOUND

In this section we give an improved upper bound on the competitive ratio for the online makespan minimization on unrelated machines, under uniformly random job arrivals. We assume that $\textsc{opt} \geq 1$. We further assume wlog that $p_{i,j} \in [0,1] \cup \{\infty\}$. This is because we can guess the optimum objective within a constant factor by the standard doubling trick: if the guess turns out to be wrong (we can solve the offline problem within a factor of 2 [26]), then we double our guess and pretend that all machines have zero load. When the current guess is $g \geq 2\textsc{opt}$, j cannot be assigned to machine i if $p_{i,j} \geq g$; therefore, we can replace the value of $p_{i,j}$ with ∞. For all other $p_{i,j} \leq g$, by scaling them down by g, we have the assumption.

3.1 An $O(\frac{\log m}{\log\log m})$ Upper Bound

3.1.1 Algorithm. The algorithm has two phases that are identical [19, 29]. In the second phase, which starts when the $(\lfloor n/2\rfloor+1)$st job arrives, it pretends that no jobs have arrived so far. The algorithm uses a softmax of machine loads as the potential, and assigns a new job to the machine that increases the potential the least.

To describe the algorithm, it will be convenient to define load vector $x \in \mathbb{R}^m_{\geq 0}$, which describes the current load of each machine, i.e., if J_i is the set of jobs currently assigned to machine i, $x_i = \sum_{j\in J_i} p_{i,j}$. For easy indexing, say job j^t arrives at time t. If we

assign j^t to machine i, it adds to the current load vector a vector w^t, whose ith entry is $p_{i,j} \in [0,1]$ and all other entries are 0. Notice that w^t has only one non-zero coordinate. Let s^t be the load vector right after the algorithm assigned j^t, i.e., $s^t = w^1 + \cdots + w^t$.

We are now ready to define our potential function:

$$\psi(x) = \frac{1}{a} \ln \sum_{i \in \mathcal{M}} e^{ax_i},$$

where $x \in \mathbb{R}^m_{\geq 0}$ is the load vector and $a > 0$ is a parameter to be decided later. As mentioned above, a new job is assigned to the machine that increases the potential the least.

As observed in [19, 29], it suffices to only consider the first half of the jobs. This is because the first half and the second half have the same distribution, and the algorithm "resets" after seeing the first half of the jobs. This will increase the makespan only by a factor of two. Henceforth, we will only consider the first $n/2$ jobs.

Define $v^t = \nabla\psi(s^{t-1})$. Notice that $v^t_i = \frac{1}{a} \frac{a \cdot e^{ax_i}}{\sum_{i'} e^{ax_{i'}}} = \frac{e^{ax_i}}{\sum_{i'} e^{ax_{i'}}}$. The following is immediate by straightforward calculations.

Claim 1. *For all $t \geq 1$, we have $||v^t||_1 = 1$ and $v^t \leq e^a v^{t-1}$, where the inequality holds for each coordinate.*

Using the convexity of the ψ function, we have

$$\psi(s^t) - \psi(s^{t-1}) \leq \langle w^t, \nabla\psi(s^t) \rangle = \langle w^t, v^{t+1} \rangle \leq e^a \langle w^t, v^t \rangle.$$

Then, adding the inequalities over all t's in $\{1, 2, \ldots, n/2\}$, we have

$$\psi(s^{n/2}) - \psi(0) \leq e^a \sum_{t=1}^{n/2} \langle w^t, v^t \rangle.$$

Noticing that $\psi(s^{n/2}) \geq ||s^{n/2}||_\infty$ and $\psi(0) = \frac{\ln m}{a}$, we have

$$||s^{n/2}||_\infty \leq e^a \sum_{t=1}^{n/2} \langle w^t, v^t \rangle + \frac{\ln m}{a}.$$

Let $o^t \in [0,1]^m$ be the load vector induced by the job j^t in the optimum solution. Then $o^t_{i'} = p_{ij_t}$ if $i' = i$ and 0 otherwise. Here, we are considering the fixed optimum solution that assigns all the n jobs in J to machines. Although each job is assigned to a fixed machine in the optimum solution, o^t is stochastic as j^t is the tth job in the random order.

Claim 2. *For all $t \geq 1$, we have $\langle v^t, w^t \rangle \leq e^a \langle v^t, o^t \rangle$.*

Proof.

$$\begin{aligned}
\langle w^t, v^t \rangle &= \langle w^t, \nabla\psi(s^{t-1}) \rangle \\
&\leq \psi(s^t) - \psi(s^{t-1}) && \triangleright\ s^t = s^{t-1} + w^t \text{ and } \psi \text{ is convex} \\
&\leq \psi(s^{t-1} + o^t) - \psi(s^{t-1}) && \triangleright\ \text{Algorithm is greedy} \\
&\leq \langle o^t, \nabla\psi(s^{t-1} + o^t) \rangle && \triangleright\ \psi \text{ is convex} \\
&\leq \langle o^t, e^a \nabla\psi(s^{t-1}) \rangle && \triangleright\ \text{Since } ||o^t||_\infty \leq 1 \\
&= e^a \langle o^t, v^t \rangle. && \square
\end{aligned}$$

Thanks to this claim, we have

$$||s^{n/2}||_\infty \leq e^{2a} \sum_{t=1}^{n/2} \langle v^t, o^t \rangle + \frac{\ln m}{a}. \tag{1}$$

Consider an arbitrary time step t. So far, only jobs $j^1, \ldots, j^{t-1}$ have been revealed and they completely determine v^t. Then, j^t is sampled from the other jobs in J uniformly at random. Note that $o^t + o^{t+1} + \cdots + o^n \leq \text{OPT} \cdot \mathbf{1}$. Therefore, we have $\mathbb{E}[o^t \mid v^t] \leq \frac{\text{OPT}}{n-(t-1)} \cdot \mathbf{1}$. Moreover, $||v^t||_1 \leq 1$; see Claim 1. So, we have $\mathbb{E}[\langle v^t, o^t \rangle \mid v^t] \leq \frac{\text{OPT}}{n-(t-1)}$. Taking expectation over Eq. (1), we get

$$\begin{aligned}
\mathbb{E}[||s^{n/2}||_\infty] &\leq e^{2a} \text{OPT} \left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n/2} \right) + \frac{\ln m}{a} \\
&\leq 2e^{2a} \text{OPT} + \frac{\ln m}{a}.
\end{aligned}$$

The last inequality above is why we run the algorithm in two phases (otherwise, the summation leads to a $\Theta(\log n)$ term).

By setting $a = \frac{\ln \ln m}{6}$, we have

$$\begin{aligned}
2e^{2a} \text{OPT} + \frac{\ln m}{a} &= 2(\ln m)^{\frac{1}{3}} \text{OPT} + \frac{6 \ln m}{\ln \ln m} \\
&= O\left(\frac{\ln m}{\ln \ln m} \right) \text{OPT},
\end{aligned}$$

where the last equation follows since $\text{OPT} \geq 1$ and $2(\ln m)^{\frac{1}{3}} < \frac{6 \ln m}{\ln \ln m}$ for $m \geq 2$. Thus, we have shown that the maximum load is $O(\frac{\log m}{\log \log m})\text{OPT}$ in expectation for the first half of the arriving jobs.

As discussed, our algorithm restarts after assigning the first half jobs, and the makespan for the second half of jobs can be upper bounded symmetrically. Thus, we have the following.

Theorem 1. *There exists an $O(\log m / \log \log m)$-competitive algorithm for minimizing makespan on unrelated machines, when the jobs arrive in uniformly random order.*

4 GRAPH BALANCING: LOWER BOUNDS

In this section we obtain new lower bounds for the online graph balancing problem, which is a special case of unrelated machines, in the random edge arrival model. Interestingly, our lower bounds hold for tree instances even when the structure of the tree is fully known to the algorithm a priori. It can be easily observed that in any tree instance, an optimal offline solution always attains a maximum load of 1 by simply rooting the tree at an arbitrary node and orienting every edge away from the root. We first present our stronger lower bound, which only holds for the greedy algorithm, and then present a lower bound that holds for any algorithm.

4.1 Lower Bound for a Greedy Algorithm

Consider the following natural greedy algorithm. When an edge $e = (u, v)$ arrives, the algorithm (Greedy) assigns e to the endpoint that has the least load at that time, breaking ties uniformly at random. Here, a node's load is the number of already-arrived edges that have been oriented towards it. Greedy is known to have the optimal competitive ratio of $\Theta(\log n)$ for online graph balancing in the adversarial edge arrival model.

In this section we show a lower bound of $\Omega(\log n / \log \log n)$ for Greedy in the random arrival model even when the input graph is a tree. The lower bound instance is the following complete tree.

Lower Bound Instance. Let T be a rooted tree with root r and depth k. Each internal node of T has $d = k^4$ children. The total

number of nodes in T is thus $n = \Theta(d^k) = \Theta(k^{4k})$; so we have $k = \Theta(\log n/\log\log n)$. We assume that k is sufficiently large.

Before we discuss the intuition and the analysis, we first set up some notation and terminology. For ease of analysis, we pretend that edges adjacent to r also have a parent edge (or equivalently, the tree is rooted at a degree 1 node r' that connects to r). An edge's *depth* is defined as the number of edges including itself on the path from the root to the edge. Thus, the leaf edges have depth k and the edges incident to the root have depth 1. A node u's *height* is defined as the number of edges between u to a closest leaf node. Thus, a leaf edge has two end points of height 1 and 0 respectively. Similarly, an edge's height is defined as the height of the node of its end point that is closer to the root.

4.1.1 Intuition. Suppose edges of T arrive in a specific order, from bottom to top—so all leaf edges appear first, and the edges of height two arrive, and so forth. Consider the d leaf edges that share a parent node. With high probability, one of these d edges will be oriented towards the parent (and all the others arriving afterwards will be directed towards the leaves). So after all the leaf edges arrive, (almost) all nodes of height 1 will have a load of 1. Then when all the edges of the upper level—equivalently the edges of height 2—arrive, a similar argument shows that two of the edges that share a parent will be directed towards the parent, so nodes of height 2 get a load of 2. Continuing this way, the root node will have a load of $k = \Omega(\log n/\log\log n)$. The primary intuition is that by making the degree of each internal node large enough—even when edges arrive in random order—there is a subtree of a similar structure where edges arrive in this specific leaf-to-root order.

4.1.2 Analysis. For the sake of analysis, we let each edge e be associated with a random number r_e sampled from $[0, 1]$ uniformly at random. We assume that edge e arrives at time r_e. Note that this way we can simulate the random arrival order of the edges. As discussed before, the greedy algorithm suffers the most when edges arrive in the bottom-to-top order. Thus, we will seek to show that there is a k^2-ary subtree where edges arrive in the bottom-to-top order. To make the analysis more convenient, we will only look at trees where edges arrive in a more structured manner. Let $I_1, \dots I_k$ be an equal partition of $(0, 1)$ into intervals, i.e., $I_1 = (0, 1/k), I_2 = (1/k, 2/k), \dots$ and so on.[1]

Definition 1 (Bad node)**.** *We say that a node u of height h is* bad *if at least k^2 of its child edges arrive during I_h; let B_u denote this bad event (for the algorithm).*

The following lemma is a straightforward consequence of the Chernoff bound. We bound the probability that the bad event does not occur, i.e., for a node u of height h, at most k^2 of its child edges arrive during I_h.

Lemma 1. *For a non-leaf node u, $\Pr[B_u] \geq 1 - \frac{1}{e^{k^3/8}}$.*

Proof. Consider a non-leaf node u with height h. We want to calculate the probability that at most k^2 child edges arrive in the interval I_h. Let $\{e_i\}_{i=1}^{k^4}$ denote the k^4 child edges of node u. For any child edge e_i, let X_i denote the random variable that is 0 if e_i arrives in I_h and 1 otherwise. Define $X = \sum_{i=1}^{k^4} X_i$. Since each edge e_i arrives at $r_{e_i} \in [0, 1]$ independently and uniformly at random, we have $\mathbb{E}[X_i] = 1/k,\ \forall e_i$. Applying the standard Chernoff bound (Theorem 4 in Appendix A with $\mu = k^3$ and $\delta = 1 - \frac{1}{k}$), we get $\Pr[X < k^2] = e^{-k^3\delta^2/2} \leq e^{-k^3/8}$ for any $k \geq 3$. □

Definition 2 (Bad subtree)**.** *We say a rooted subtree T' of T is* bad *if it is a full k^2-ary tree of height k whose internal nodes are all bad.*

By definition, the edges of a *bad subtree* arrive in a leaf-to-root order, i.e, for any internal node in the tree, all child edges arrive before the parent edge. We now argue that at least one bad subtree exists with high probability.

Lemma 2. *There exits a bad subtree T' with probability at least $1 - \frac{2k^{2k}}{e^{k^3/8}}$.*

Proof. The root is good with probability at most $\rho := 1/e^{k^3/8}$. Conditioned on the root being bad, consider some arbitrary k^2 child edges that all arrive during interval I_k and let $u_1, \dots u_{k^2}$ be the corresponding child nodes. By Lemma1, each of those u_i's is good with probability at most ρ. So by a union bound, the probability that at least one of those nodes is good is at most $k^2\rho$.

Using this argument recursively and applying a union bound over all levels, the probability that we have no bad subtree is at most $\rho(1 + k^2 + \cdots + k^{2k}) \leq 2k^{2k}/e^{k^3/8}$. □

Lemma 3. *Suppose the input graph is a bad tree, with no other edges. In other words, suppose the input graph is a full k^2-ary tree of height k where edges in the bottom-to-top order. Then, the maximum load of a node is at least k with a probability at least $1 - 1/n$.*

Proof. We say a node u of height h is fully loaded if its load is at least h. We denote the event as F_u. Let $f_h := \Pr[F_u \mid F_v \ \ \forall v \in C_u]$ for a node of height h; here C_u denotes u's children (nodes).

We show that $c_h \geq 1 - \frac{1}{2^{k^2-h}}$. Indeed, consider an arbitrary node u of height h. Since we assume that edges arrive in the bottom-to-top order, before any child edges of u arrives, we know that every child node v of u has load at least $h - 1$ from the condition $F_v, \forall v \in C_u$. So, the first $h - 1$ child edges of u will orient towards u. For any other edge (u, v) arriving later, if v already has load more than $h - 1$, the claim is immediate. So, suppose not. Then, u can have load $h - 1$ only when all the child edges of u, except the first $h - 1$, are not oriented towards u, which occurs with probability $\frac{1}{2^{k^2-h}} \leq \frac{1}{2^{k^2-k}}$.

The bad tree has at most $(k^2)^{k+1}$ internal nodes, and the root has load at least h if F_u occurs for every internal node u. Thus, by a simple union bound, the root has load at least $k = \Theta(\log n/\log\log n)$ with probability at least $1 - (k^2)^{k+1}\frac{1}{2^{k^2-k}} > 1 - 1/n$. □

We now explain why we continue to have the lower bound of k, independent of the other edges different from the bad tree. This follows from the monotonicity of the greedy algorithm. In other words, inserting an edge (job) to an input sequence can only increase the maximum load.

Since $k = \Omega(\log n/\log\log n)$, we have the following theorem.

[1]Here, we do not distinguish between closed intervals and open intervals as almost surely no edge will arrive at a time that is a multiple of $1/k$.

Theorem 2. *There exists a tree T with n nodes such that, for uniform random edge arrivals, the greedy algorithm outputs an orientation with maximum load $\Omega(\log n/\log\log n)$ with high probability.*

4.2 Lower Bound for Arbitrary Algorithms

In this section we show a lower bound of $\Omega(\sqrt{\log n})$ for the graph balancing problem with random edge arrivals. Unlike Section 4.1, this lower bound is not restricted to the greedy algorithm and holds for all online algorithms.

Lower Bound Instance. We denote the lower bound instance of height D as T_D, where $D \geq 0$ is an integer. For every $0 \leq d \leq D$, T_d is constructed recursively as follows: T_0 is a singleton node. Let r be the root of T_d. For every $d' = 0, 1, \ldots, d-1$, there are $2^{D-d'}$ copies of $T_{d'}$, each of which is a subtree of r. Here, it is important to note that $T_1, \ldots, T_D$ have dependency on D. See Figure 2 for an illustration of the instance T_2.

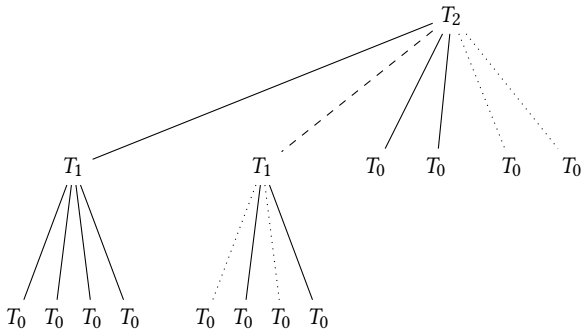

Figure 2: Example of a lower bound instance for $D = 2$. Consider the time when the dashed edge appears. Edges are dotted if they have already arrived, solid otherwise. For the dashed edge, no algorithm can distinguish which of its end points is the parent node.

We label each node v in T_D with an integer in $[0, D]$: if the subtree rooted at v is a copy of T_d, then we set $\ell(v) = d$ to be the label of v. Therefore, for two integers $0 \leq d' < d \leq D$, any node u with label d has $2^{D-d'}$ children with label d'.

4.2.1 Intuition. The main idea behind this lower bound instance is that when some edges arrive online, the algorithm cannot distinguish between their parent node and the child node. Therefore, we can assume that the algorithm assigns such an edge $e = (u, v)$ to u with probability $1/2$ and to v with probability $1/2$. In particular, consider any node v of with label d. Then, for each $d' \in [0, d-1]$, there is at least one edge (v, u) with $\ell(u) = d'$, such that with probability at least $1/2$ the algorithm cannot distinguish the subtree rooted at u consisting of the edges that have arrived so far, and the analogously defined subtree rooted at v. Using this observation and the fact that $D = \Omega(\sqrt{\log n})$, we can lower bound the maximum load on the root of the tree.

4.2.2 Analysis. We begin by defining the event when the algorithm cannot distinguish between the parent and child node.

Definition 3. *Let π be a random permutation over all edges in T_D. Consider an edge $e = (u, v)$ with u being the parent. The permutation π is* bad *for e, if the following hold:*

- *the parent edge of u appears after e in the order π, and*
- *all other child edges (u, v') of u with $\ell(v') \geq \ell(v)$ appear after (u, v) in the order π.*

Consider the arrival time of an edge $e = (u, v)$ where $\ell(u) = d$ and $\ell(v) = d' < d$. We observe that conditioned on the event that the permutation π is bad for edge (u, v), the subtrees rooted at u and v comprised only of the edges that have arrived so far are stochastically identical. Thus, we can assume wlog that any fixed algorithm orients the edge (u, v) towards each of its end points with probability $\frac{1}{2}$, which can be formally shown in the following claim. Note that it suffices to consider a fixed deterministic algorithm thanks to Yao's minmax principle [10].

Claim 3. *Conditioned on the event that the permutation π is bad for edge (u, v), any fixed deterministic algorithm orients the edge towards each of its end points with probability $\frac{1}{2}$.*

Proof. Let ψ be a random permutation of the node identities that is generated by the adversary, which is unknown to the deterministic algorithm. Let π be the random permutation of edges that denotes the arrival order. We condition on the event that the arrival order π is bad for some specific edge (u, v). Note that the algorithm makes its decisions based on revealed edges of the form $(\psi(x), \psi(y))$. By our construction, the subtrees T_u and T_v rooted at u and v respectively consisting of arrived edges are stochastically identical. Suppose the algorithm orients the revealed edge $(\psi(u), \psi(v))$ towards u in a specific instantiation. Since T_u and T_v are stochastically identical, they will have their instantiations swapped equiprobably, meaning that the algorithm must orient (u, v) towards v with the same probability (over the random choice of ψ and π). □

In the following lemma, we calculate the probability that a permutation is bad for an edge. For our purpose, it is enough to consider an edge with the root r of T_D as one of its end points.

Lemma 4. *A permutation π over the edges of T_D is bad for the first appearing edge (r, u) with $\ell(u) = d$ with probability at least $\frac{1}{2}$.*

Proof. The probability can be easily calculated as

$$\frac{\#\text{edges with } \ell(v) = d}{\#\text{edges with } \ell(v) \geq d} = \frac{2^{D-d}}{2^{D-d} + \cdots + 2^{D-(D-1)} + 1} \geq \frac{1}{2}. \quad \square$$

Focus on the root r with $\ell(r) = D$, and an integer $d < D$. Consider the first edge (r, u) with $\ell(u) = d$. With probability $1/2$, the permutation is bad for this edge. Conditioned on the permutation being bad, the algorithm assigns the edge to r with probability $1/2$. Therefore, the algorithm assigns the edge to r with probability at least $1/4$. Consider all values of $d < D$, in expectation the root will get a load of $\Omega(D)$. The size of the tree is $2^{O(D^2)}$, as we will see below. Therefore we can obtain a lower bound of $\Omega(\sqrt{\log n})$.

Lemma 5. *If n is the size of the tree T_D, then $D = \Omega(\sqrt{\log n})$.*

Proof. Let $n(d)$ denote the number of nodes in T_d. From the recursive construction of the trees, we have

$$n(d) = \begin{cases} 1 & d = 0 \\ 2^{D-0}n(0) + \cdots + 2^{D-(d-1)}n(d-1) + 1 & d \in [1, D-1] \end{cases}$$

Since $n(d)$ is increasing in d, when $d \geq 1$, we have $n(d) \leq 2^D D \cdot n(d-1) \leq 4^D \cdot n(d-1)$. Therefore, $n = n(D) \leq 4^{D^2}$. This gives us the desired result $D = \Omega(\sqrt{\log n})$.

□

Theorem 3. *There exists a tree T with n nodes such that for uniform random edge arrivals, any algorithm outputs an orientation with maximum load $\Omega(\sqrt{\log n})$, with high probability.*

5 CONCLUDING REMARKS

In this paper, we revisit the classic online load balancing problem for unrelated machines in the random arrival order model. While we achieve an exponential improvement over the previous lower bound, a substantial gap remains between the lower bound of $\Omega(\sqrt{\log m})$ and the upper bound of $O(\log m / \log\log m)$. Importantly, the question of whether an algorithm with a better competitive ratio exists even for tree inputs in the online graph balancing problem remains open. This holds true even when the entire tree structure is known a priori (with the identities of nodes and edges hidden).

ACKNOWLEDGMENTS

Petety is supported in part by NSF grants CCF-1844939 and CCF-2121745. Im is supported in part by an ONR grant N00014-22-1-2701 as well as them. Li is supported in part by the State Key Laboratory for Novel Software Technology, and the New Cornerstone Science Laboratory.

REFERENCES

[1] Susanne Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999.
[2] Susanne Albers and Maximilian Janke. Scheduling in the random-order model. *Algorithmica*, 83(9):2803–2832, 2021.
[3] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *JACM*, 44(3):486–504, 1997.
[4] Baruch Awerbuch, Yossi Azar, Edward F Grove, Ming-Yang Kao, P Krishnan, and Jeffrey Scott Vitter. Load balancing in the L_p norm. In *FOCS*, pages 383–391, 1995.
[5] Yossi Azar. On-line load balancing. *Online Algorithms: The State of the Art*, pages 178–195, 2005.
[6] Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and Bruce Shepherd. Tight bounds for online vector bin packing. In *STOC*, pages 961–970, 2013.
[7] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18(2):221–237, 1995.
[8] Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1):108–121, 2000.
[9] Plank B.M. Online scheduling problems in the random order model. *Bachelor's Thesis*, 2017.
[10] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.
[11] Niv Buchbinder and Joseph Naor. Fair online load balancing. In *SPAA*, pages 291–298, 2006.
[12] Ioannis Caragiannis. Better bounds for online load balancing on unrelated machines. In *SODA*, pages 972–981, 2008.
[13] Tomás Ebenlendr, Marek Krcál, and Jiri Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *SODA*, pages 483–490, 2008.
[14] Tomáš Ebenlendr, Marek Krčál, and Jiří Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68:62–80, 2014.
[15] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*, 88:50–71, 2017.
[16] R. L. Graham. Bounds for certain multiprocessing anomalies. *SIAM J. Appl. Math.*, 1966.
[17] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:416–429, 1969.
[18] Anupam Gupta, Gregory Kehne, and Roie Levin. Random order online set cover is as easy as offline. In *FOCS*, pages 1253–1264, 2022.
[19] Anupam Gupta and Marco Molinaro. How experts can solve LPs online. In *ESA*, pages 517–529, 2014.
[20] Anupam Gupta and Sahil Singla. *Random-Order Models*, page 234–258. Cambridge University Press, 2021.
[21] Sungjin Im, Nathaniel Kell, Janardhan Kulkarni, and Debmalya Panigrahi. Tight bounds for online vector scheduling. *SIAM J. Comput.*, 48(1):93–121, 2019.
[22] Yichuan Jiang. A survey of task allocation and load balancing in distributed systems. *IEEE TPDS*, 27(2):585–599, 2015.
[23] Haim Kaplan, David Naori, and Danny Raz. Almost tight bounds for online facility location in the random-order model. In *SODA*, pages 1523–1544, 2023.
[24] David R Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA*, pages 36–43, 2004.
[25] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *SODA*, pages 1859–1877, 2020.
[26] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Prog.*, 46:259–271, 1990.
[27] Shi Li and Jiayi Xian. Online unrelated machine load balancing with predictions revisited. In *ICML*, pages 6523–6532, 2021.
[28] Adam Meyerson, Alan Roytman, and Brian Tagiku. Online multidimensional load balancing. In *APPROX-RANDOM*, pages 287–302, 2013.
[29] Marco Molinaro. Online and random-order load balancing simultaneously. In *SODA*, pages 1638–1650, 2017.

A CHERNOFF BOUND

Theorem 4. *Let $X_1, \ldots, X_n$ be n independent, Bernoulli random variables, where $\mathbb{P}[X_i = 1] = p_i$, for all i. For $X = \sum X_i$, its expectation is $\mu = \mathbb{E}[X] = \sum_i p_i$. Then for any $0 < \delta < 1$, we have*

$$P[X < (1-\delta)\mu] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu}. \tag{2}$$

Further, the upper bound is at most $e^{-\mu\delta^2/2}$ when $\delta \in [0, 1)$.