Tight Bounds for Online Vector Scheduling

Sungjin Im∗ Nathaniel Kell† Janardhan Kulkarni‡ Debmalya Panigrahi§

UC Merced Duke University Duke University Duke University

Abstract

The list scheduling or online load balancing problem is a central question in scheduling theory — given a set of $m$ offline machines $i \in M$ and a sequence of $n$ jobs $j \in J$ arriving online with load $p_{i,j} \geq 0$ if assigned to machine $i$, find an assignment of each job to some machine so as to minimize a given norm of overall machine loads. For identical machines (i.e., $p_{i,j} = p_j$ for all machines $i \in M$), Graham (SIAM J. of Appl. Math. 1966) gave a 2-competitive greedy algorithm for the makespan (maximum load) norm (since improved to 1.9201). For unrelated machines, the optimal competitive ratio is $O(\log m)$ for the makespan norm (Aspnes et al., JACM 1997) and $O(p)$ for general $L_p$ norms (Awerbuch et al., FOCS 1995). Chekuri and Khanna (SIAM J. of Comp. 2006) generalized the list scheduling problem to vector loads (they called it vector scheduling), where the load of job $j$ on machine $i$ is now a $d$-dimensional vector $\langle p_{i,j}(k) : k \in [d] \rangle$. They gave an $O(\log^2 d)$ offline algorithm in the case of identical machines for the makespan norm, which was later improved to $O(\log d)$ by Azar et al. (STOC 2013) (and independently by Meyerson et al. (APPROX 2013)) using an online algorithm.

In this paper, we completely solve the online vector scheduling problem in both the identical and unrelated machines settings. Our main results are:

- **Identical machines:** For the makespan norm, we show that the optimal competitive ratio is $O(\log d / \log \log d)$. The main component of this result is a new information theoretic lower bound of $\Omega(\log d / \log \log d)$ for online algorithms. The only previously known lower bound was a computational lower bound of $\omega(1)$ due to Chekuri and Khanna for the offline problem, which does not rule out very weak dependence on $d$. While such approximation might still be possible in the offline model, our lower bound rules out the use of an online strategy (note that the current best offline approximation is via an online algorithm). We also show that our lower bound is asymptotically tight by giving an $O(\log d / \log \log d)$-competitive online algorithm, which slightly improves upon the results of Azar et al. and Meyerson et al. Our techniques also extend to asymptotically tight upper and lower bounds for general $L_p$ norms.

- **Unrelated machines:** For arbitrary (and non-identical) $L_p$ norms on each dimension, we give an algorithm with a competitive ratio of $O(p + \log d)$. We also show that this result is asymptotically tight by demonstrating a lower bound of $\Omega(p \log d)$ (an $\Omega(\rho)$ lower bound was previously shown by Aspnes et al.). For the makespan norm, our algorithm has a competitive ratio of $O(\log m + \log d)$, which is also asymptotically tight.

∗Email: sim3@ucmerced.edu. Supported in part by NSF grant CCF-1409130.
†Email: kell@cs.duke.edu. Supported in part by a Duke University startup grant and a Google Faculty Research Award.
‡Email: kulkarni@cs.duke.edu.
§Email: debmalya@cs.duke.edu. Supported in part by a Duke University startup grant and a Google Faculty Research Award.
1 Introduction

The list scheduling (or online load balancing) problem has been extensively studied in the scheduling literature over several decades. In this problem, a sequence of \( n \) jobs \( J \) arrive online and have to be assigned to one among a set of \( m \) machines \( M \). Job \( j \) has a load of \( p_{i,j} \) if assigned to machine \( i \), and the typical goal is to minimize the \( L_\infty \) norm (also called makespan) or the \( L_p \) norm (for a given \( p \)) of the machine loads. In the vector scheduling (VS) problem (introduced by Chekuri and Khanna [15]), the load of a job is a \( d \)-dimensional vector rather than a scalar and the goal is to simultaneously optimize the machine loads on all the dimensions. If every job has the same load (vector) on all machines, then we say that the machines are identical, else the machines are unrelated. In this paper, we completely solve the online vector scheduling problem by giving asymptotically matching upper and lower bounds for all \( L_p \) norms, including the makespan norm, in both the identical and unrelated machines settings.

The VS problem arises in practical applications such as cloud computing where each job typically has a multi-dimensional load on a cluster (dimensions include storage requirements, processor cycles, network bandwidth, etc.) and the goal is to assign jobs to clusters so as to balance the load on all resources simultaneously. This directly relates to the VS problem. Furthermore, it has been established that the usage of (or the load on) different resources cannot be modeled uniformly (e.g., the \( L_2 \) norm was used for disk storage in [15] [18]), which motivates the most general version where the VS algorithm is required to optimize different \( L_p \) norms in the different dimensions.

1.1 Problem Definition

As described earlier, we have \( n \) jobs (denoted \( J \)) that arrive online and must be immediately and irrevocably assigned on arrival to one among a fixed set of \( m \) machines (denoted \( M \)). We denote the \( d \)-dimensional load vector of job \( j \) on machine \( i \) by \( p_{i,j} = (p_{i,j}(k) : k \in [d]) \); the load vectors of a job are revealed on its online arrival. For identical machines, the load of job \( j \) in dimension \( k \) is identical for all machines \( i \), and we denote it \( p_{j}(k) \). Let us denote the assignment function of jobs to machines by \( f: J \rightarrow M \). An assignment \( f \) produces a load of \( \Lambda_i(k) = \sum_{j : f(j) = i} p_{i,j}(k) \) on dimension \( k \) of machine \( i \); we succinctly denote the machine loads in a dimension by an \( m \)-dimensional vector \( \Lambda(k) \). (For comparison, note that for the scalar list scheduling problem, there is only one such machine load vector, whereas there are \( d \) such vectors in the VS problem.)

Let us first consider the makespan norm. We assume (by scaling) that the optimal makespan norm on each dimension is 1. Then, the problem (denoted VS\text{MAX}) is defined as follows.

**Definition 1.** VS\text{MAX}: For any dimension \( k \), the objective is the maximum load over all machines, i.e.,

\[
\|\Lambda(k)\|_\infty = \max_{i \in M} \Lambda_i(k).
\]

An algorithm is said to \( \alpha \)-competitive if \( \|\Lambda(k)\|_\infty \leq \alpha \) for every dimension \( k \). As stated earlier, we consider two settings for this problem: identical machines (denoted VS\text{MAX}-I) and unrelated machines (denoted VS\text{MAX-U}).

Next, we consider the more general objective of arbitrary \( L_p \) norms. Let us first consider the case of identical machines. Our goal here will be to simultaneously optimize all norms on all dimensions. We denote this problem by VS\text{ALL}-I.

**Definition 2.** VS\text{ALL}-I: For dimension \( k \) and norm \( L_r \), \( r \geq 1 \), the objective is

\[
\|\Lambda(k)\|_r = \left( \sum_{i \in M} \Lambda_i^r(k) \right)^{1/r}.
\]

\(^{1}\)There are intermediate models as well, though we do not consider them in this paper. The interested reader is referred to the section on related work for some references and for pointers to relevant surveys and textbooks.
An algorithm is said to $\alpha_r$-competitive for the $L_r$ norm if $\|\Lambda(k)\|_r \leq \alpha_r$ for every dimension $k$ and every norm $L_r$, $r \geq 1$.

For unrelated machines, there is a polynomial lower bound for simultaneously optimizing multiple $L_p$ norms, even with scalar loads. This rules out an all norms approximation. Therefore, we focus on an any norm approximation, where the algorithm is given norms $r_1, r_2, \ldots, r_d$ (where each $r_k \geq 1$), and the goal is to minimize the $L_{r_k}$ norm for dimension $k$. The same lower bound also rules out the possibility of the algorithm being competitive against the optimal value of each individual norm in their respective dimensions. Essentially, the vector of optimal norms can be highly infeasible since the individual norms are minimized in the optimal vector disregarding all other dimensions. We use a standard trick in multi-objective optimization to circumvent this impossibility: we only require the algorithm to be competitive against any given feasible target vector $T = (T_1, \ldots, T_d)$. For ease of notation, we assume wlog (by scaling) that $T_k = 1$ for all dimensions $k$. [3] Now, we are ready to define the VS problem with arbitrary $L_p$ norms for unrelated machines — we call this problem VSANY-U.

**Definition 3. VSANY-U:** For dimension $k$, the objective is

$$\|\Lambda(k)\|_{r_k} = \left( \sum_{i \in M} \Lambda_i^{r_k}(k) \right)^{1/r_k}.$$  

An algorithm is said to $\alpha_{r_k}$-competitive in the $L_{r_k}$ norm if $\|\Lambda(k)\|_{r_k} \leq \alpha_{r_k}$ for every dimension $k$. Note the difference between the definitions of VSALL-I and VSANY-U: in the former, the algorithm must be competitive in all norms in all dimensions simultaneously, whereas in VSANY-U, the algorithm only needs to be competitive against a single norm in each dimension that is specified in the problem input.

### 1.2 Our Results

**Identical machines.** Our main technical result is a lower bound for the makespan norm.

**Theorem 4.** There is a lower bound of $\Omega\left(\frac{\log d}{\log \log d}\right)$ on the competitive ratio of online algorithms for the VSMAX-I problem.

This shows that the previous best upper bound of $O(\log d)$ obtained by Azar et al. [6] and Meyerson et al. [27] is almost tight. In fact, we improve this upper bound slightly to achieve asymptotic optimality of the competitive ratio.

**Theorem 5.** There is an online algorithm with a competitive ratio of $O\left(\frac{\log d}{\log \log d}\right)$ for the VSMAX-I problem.

Previously, Chekuri and Khanna [16] showed that it is NP-hard to obtain an $O(1)$-approximation for the offline VSMAX-I problem. We note that:

- The offline approximability of the VSMAX-I problem remains open. The lower bound in [16] asserts that there must be some dependence of the approximation ratio on the number of dimensions $d$. However, this dependence can be much weaker than the (roughly) logarithmic dependence that we have now shown to be tight for online algorithms. Interestingly, the best algorithms currently known for the offline problem are, in fact, online algorithms ([6], [27], and the slightly better algorithm in Theorem 5 of this paper). Our lower bound closes this line of work.

- Unlike [16], which applies only to polynomial-time algorithms since it is for the offline problem, our lower bound is information theoretic, i.e., even applies to online algorithms with unlimited running time.

---

[3] A target vector is feasible if there is an assignment such that for every dimension $k$, the value of the $L_{r_k}$ norm in that dimension is at most $T_k$. Our results do not rely heavily on the exact feasibility of the target vector; if there is a feasible solution that violates targets in all dimensions by at most a factor of $\beta$, then our results hold with an additional factor of $\beta$ in the competitive ratio.
Next, we consider the VSALL-1 problem, i.e., simultaneously optimizing all $L_p$ norms for identical machines. We show that this problem has a close connection to VSMAX-1, and utilize the VSMAX-1 algorithm in Theorem 5 to obtain the following upper bound.

**Theorem 6.** There is an online algorithm with a competitive ratio of $O\left(\left(\frac{\log d}{\log \log d}\right)^{\frac{1}{r-1}}\right)$ for the $L_r$ norm in the VSALL-1 problem.

We also adapt our VSMAX-1 lower bound (Theorem 4) to show that this competitive ratio is tight up to constants.

**Theorem 7.** There is a lower bound of $\Omega\left(\left(\frac{\log d}{\log \log d}\right)^{\frac{1}{r-1}}\right)$ on the competitive ratio of online algorithms for minimizing the $L_r$ norm in the VS problem on identical machines.

**Unrelated machines.** For the VSMAX-U problem with scalar loads, Aspnes et al. \[2\] gave an $O(\log m)$-competitive algorithm, based on the use of exponential potentials. This idea extends to the case of vector loads, and Meyerson et al. \[27\] showed that the natural extension has a competitive ratio of $O(\log m + \log d)$ with vector loads. To show an asymptotically matching lower bound, we only need to show the dependence on $d$ since a lower bound of $\Omega(\log m)$ is already known in \[2\]. The proof is straightforward, but we include it for completeness.

**Theorem 8.** There is a lower bound of $\Omega(\log m + \log d)$ on the competitive ratio of online algorithms for the VSMAX-U problem.

More interestingly, we consider the case of optimizing a given $L_p$ norm in each dimension, i.e., the VSANY-U problem. For scalar loads, Awerbuch et al. \[4\] obtained a competitive ratio of $O(p)$, and showed that it is asymptotically tight. However, unlike in makespan minimization, the extension to vector loads (the VSANY-U problem) is no longer straightforward. Each individual dimension may now be optimizing a different $L_p$ norm, and the goal is to simultaneously optimize their unrelated potentials. We achieve a delicate combination of the objectives using convexity arguments to balance the potentials carefully.

**Theorem 9.** There is an algorithm for the VSANY-U problem that simultaneously obtains a competitive ratio of $O(r_k + \log d)$ for each dimension $k$, where the goal is to optimize the $L_{r_k}$ norm in the $k$th dimension.

We also show that this result is asymptotically tight by extending the lower bound of $\Omega(\log d)$ we have shown to derive Theorem 8 to any $L_p$ norm; as mentioned, a lower bound of $\Omega(r)$ is already known \[4\].

**Theorem 10.** There is a lower bound of $\Omega(r + \log d)$ on the competitive ratio of online algorithms for the VSANY-U problem, even when the goal is to optimize the $L_r$ norm in all dimensions for any fixed $r \geq 1$.

### 1.3 Our Techniques

First, we outline the main techniques used for the identical machines setting. To obtain lower bounds for the VSMAX-1 and VSALL-1 problems, we derive our inspiration from a lower bound on the online vertex coloring (VC) problem due to Halldórsson and Szegedy \[23\]. Connections between VC and VSMAX-1 \[16\] (and related problems such as vector bin packing \[6\]) have previously been identified as well. The core idea in \[16\] is to encode a VC instance as a VSMAX-1 instance where the number of dimensions $d$ is (roughly) $n^B$ and show that an approximation factor of (roughly) $B$ for VSMAX-1 implies an approximation factor of (roughly) $n^{1-1/B}$ for VC. Suppose, we use $B = \log d / \log \log d$; a simple calculation shows that a competitive ratio of $B$ for VSMAX-1 implies a competitive ratio of $\frac{n}{1+o(1)}$ for online VC, which unfortunately is not ruled out by the best lower bound of $\Omega(n/\log^2 n)$ known \[23\].

A second approach is to explore the connection of VMAX-1 with online vector bin packing (VBP), where multi-dimensional items arriving online must be packed into a minimum number of identical multi-dimensional bins. Recently, Azar et al. \[6\] obtained strong lower bounds of $\Omega(d^{1/B})$ where $B \geq 1$ is the
capacity of each bin in every dimension (the items have a maximum size of 1 on any dimension). It would be tempting to conjecture that the inability to obtain a constant approximation algorithm for the VBP problem unless \( B = \Omega(\log d) \) should yield a lower bound of \( \Omega(\log d) \) for the VSMAX-I problem. Unfortunately, this is false. The difference between the two problems is in the capacity of the bins/machines that the optimal solution is allowed to use: in VSMAX-I, this capacity is 1 whereas in VBP, this capacity is \( B \). Therefore, a lower bound for VBP does not imply any lower bound for VSMAX-I. On the other hand, an upper bound of \( O(d^{1/(B-1)} \log d) \) for the VBP problem is obtained in [6] via an \( O(\log d) \)-competitive algorithm for VSMAX-I. Improving this ratio considerably for VSMAX-I would have been a natural approach for closing the gap for VBP; unfortunately, our lower bound of \( \Omega(\log d / \log \log d) \) rules out this possibility.

Our lower bound is obtained via a different approach from the ones outlined above. Instead of using the VC problem directly, we use the closely related problem of minimizing the size of the largest monochromatic clique given a fixed number of colors. Our main technical result is to show that this problem has a lower bound of \( \sqrt{t} \) for online algorithms, where \( t \) is the number of colors. For this purpose, we use a novel randomized encoding scheme to generate the hard instance for the coloring problem. We also show that the same idea can be used to obtain a lower bound for any \( L_p \) norm.

In order to obtain an upper bound for the VSMAX-I problem, we consider the following naïve strategy: assign every job independently to a machine chosen uniformly at random. Using standard tail bounds, the maximum load on any machine for any dimension is (roughly) \( \log dm \). (This observation was made in [15].) To avoid the factor of \( \log m \), let us use a simple modification: whenever the load on any dimension of any machine reaches (roughly) \( \log d \), we simply stop loading the machine further and replace it with a new machine. How many machines do we end up using? We show that this number is \( O(m) \), the intuition being that only about \( m/d \) machines are at a risk of reaching a load of \( \log d \) in a random assignment. In fact, we show that the volume of jobs that are not scheduled in the first random assignment is less than \( 1/d \) times the total volume of all jobs; then any reasonable packing can schedule all the remaining jobs using \( O(m) \) machines in expectation. This is sufficient for our upper bound since we can now combine (online) a constant number of machines into a single machine, thereby losing a constant factor in the maximum load but restricting the number of machines to \( m \). A more careful calculation along these lines reduces the maximum load from \( O(\log d) \) to \( O(\log d / \log \log d) \). Moreover, this algorithm can be derandomized losing only a constant in the competitive ratio. For the VSALL-I problem, we give a reduction to VSMAX-I by structuring the instance by “smoothing” large jobs and then arguing that for structured instances, a VSMAX-I algorithm is also optimal for other \( L_p \) norms.

Next, we turn our attention to the unrelated machines setting. Here, our main result is to obtain an upper bound for the VSANY-U problem. Our algorithm is greedy with respect to a potential function (as are algorithms for all special cases studied earlier [4, 2, 13, 27]). However, the choice of the potential function is less obvious than VSMAX-U. For each individual dimension \( k \), we use the \( L^2_k \) norm as the potential (following [2, 13]). The main challenge is to combine these individual potentials into a single potential. We use a weighted linear combination of the individual potentials for the different dimensions. This is somewhat counter-intuitive since the combined potential can possibly allow a large potential in one dimension to be compensated by a small potential in a different one. However, we observe that we are aiming for a competitive ratio of \( O(r_k + \log d) \) which allows some slack compared to scalar loads if \( r_k < \log d \). Suppose \( q_k = r_k + \log d \); then we use weights of \( q_k^q_k \) in the linear combination after changing the individual potentials to \( L^q_k \). Note that as one would expect, the weights are larger for dimensions that allow a smaller slack. Our main technical contribution in this result is to show that this combined potential simultaneously leads to the asymptotically optimal competitive ratio on every dimension.
1.4 Related Work

Due to the large volume of related work, we will only sample some relevant results in online scheduling and refer the interested reader to more detailed surveys (e.g., [5, 30, 31, 29]) and textbooks (e.g., [11]).

Scalar loads. Since the \((2 - 1/m)\)-competitive algorithm by Graham [22] for minimizing makespan in list scheduling, a series of papers [8, 26, 11] have led to the current best ratio of 1.9201 [20]. On the negative side, this problem was shown to be NP-hard in the strong sense by Faigle et al. [19] and has since been shown to have a competitive ratio of at least 1.880 [9, 1, 21, 25]. For \(L_p\) norms on identical machines, Avidor et al. [3] obtained competitive ratios of \(\sqrt{4/3}\) and \(2 - O\left(\frac{\log p}{p}\right)\) for the \(L_2\) and general \(L_p\) norms respectively.

For unrelated machines, Aspnes et al. [2] obtained a competitive ratio of \(O(\log m)\) for makespan minimization, which is asymptotically tight [7]. Scheduling for the \(L_2\) norm was considered by [15, 18], and Awerbuch et al. [4] obtained a competitive ratio of \(1 + \sqrt{2}\), which was shown to be tight [14]. For general \(L_p\) norms, Awerbuch et al. [4] (and Caragiannis [13]) obtained a competitive ratio of \(O(p)\), and showed that it is tight up to constants. Various intermediate settings such as related machines (machines have unequal but job-independent speeds) [2, 10] and restricted assignment (each job has a machine-independent load but can only be assigned to a subset of machines) [7, 14, 17, 32] have also been studied for the makespan and \(L_p\) norms.

Vector loads. The \(V_{\text{MAX}}-1\) problem was introduced by Chekuri and Khanna [16], who gave an offline approximation of \(O(\log^2 d)\) and observed that a random assignment has a competitive ratio of \(O\left(\frac{\log dm}{\log \log dm}\right)\). Azar et al. [6] and Meyerson et al. [27] improved the competitive ratio to \(O(\log d)\) using deterministic online algorithms. An offline \(\omega(1)\) lower bound was also proved in [16], though it does not rule out very weak dependence on \(d\). It remains open as to whether an approximation ratio with a substantially weaker than logarithmic dependence on \(d\) is achievable for the \(V_{\text{MAX}}-1\) problem in the offline setting. Our online lower bound asserts that such an algorithm must use a radically different approach from all the known algorithms for this problem.

For unrelated machines, Meyerson et al. [27] noted that the natural extension of the algorithm of Aspnes et al. [2] to vector loads has a competitive ratio of \(O(\log dm)\) for makespan minimization; in fact, for identical machines, they used exactly the same algorithm but gave a tighter analysis. It is not difficult to see that this competitive ratio is tight up to constants for the \(V_{\text{MAX}}-U\) problem. For the offline \(V_{\text{MAX}}-U\) problem, Harris and Srinivasan [24] recently showed that the dependence on \(m\) is no longer required by giving a randomized \(O(\log d/\log \log d)\) approximation algorithm.

1.5 Organization of the paper

In Section 2 we present our lower bound for the identical machine case (Theorems 4 and 7) and then give an algorithm with a matching upper bound (Theorems 5, 6). In Section 3 we give a lower bound for the unrelated machines problem (Theorem 10) and then again provide an algorithm with a matching upper bound (Theorem 9).

2 Identical Machines

2.1 Lower Bound

In this section we prove Theorem 4 i.e., we will show that any online algorithm for the \(V_{\text{MAX}}-1\) problem can be forced to construct a schedule such that there exists a dimension where one machine has load
Lemma 12. The competitive ratio of any online algorithm for \textsc{Monochromatic Clique} is \( \Omega(\sqrt{t}) \), where \( t \) is the number of available colors.

In other words, for any online algorithm \( A \), there is an instance on which \( A \) produces a monochromatic clique of size \( \Omega(\sqrt{t}) \), whereas the optimal solution can color the graph such that the size of the largest monochromatic clique is \( O(1) \).

**Construction:** We will frame the lower-bound as a game between the adversary and the online algorithm. We adopt the terminology used by Hallådórsson and Szegedy [23] for the online coloring problem to describe this game. Namely, the algorithm will place each vertex in one of \( t \) bins to define its color assignments, whereas we will use colors to refer to the color assignments defined by the optimal solution (controlled by the adversary). For each vertex arrival, the game is defined by the following 3-step process:

1. The adversary issues a vertex \( v_j \) and defines \( v_j \)'s adjacencies with vertices \( v_1, \ldots, v_{j-1} \).
2. The online algorithm places \( v_j \) in one of the available \( t \) bins.
3. The adversary selects a color for the vertex.

We further divide each bin into \( \sqrt{t} \) slots \( 1, 2, \ldots, \sqrt{t} \). Correspondingly, we partition the \( t \) colors into \( \sqrt{t} \) color sets \( C_1, \ldots, C_{\sqrt{t}} \), each of size \( \sqrt{t} \). Each vertex will reside in a slot inside the bin chosen by the algorithm, and all vertices residing in slot \( i \) across all bins will be colored by the optimal solution using a color from \( C_i \). The high-level goal of the construction will be to produce properly colored cliques (i.e., a proper coloring implies that the optimal solution does not give the same color to adjacent vertices) inside each slot of every bin.

Consider the arrival of vertex \( v_j \). Inductively assume the previous vertices \( v_1, \ldots, v_{j-1} \) have been placed in the bins by the algorithm, and that every vertex within a bin lies in some slot. Further assume that all the vertices in any particular slot of a bin form a properly colored clique.

To specify the new adjacencies formed by vertex \( v_j \) for step 1, we will use a \( t \)-length \( \sqrt{t} \)-ary string \( s_j \), where we connect \( v_j \) to every vertex in slot \( s_j[k] \) of bin \( k \), for all \( k = 1, 2, \ldots, t \). Next, for step 2, the algorithm places \( v_j \) in some bin \( b_j \). We say that \( v_j \) is then placed in slot \( q_j = s_j[b_j] \) in bin \( b_j \). Finally for step 3, the adversary chooses an arbitrary color for \( v_j \) from the colors in \( C_{q_j} \) that have not yet been used for any vertex in slot \( q_j \) of bin \( b_j \). The adversary will end the instance whenever there exists a slot in some bin that contains \( \sqrt{t} \) vertices. This ensures that as long as the game is running, there is always an unused color in every slot of every bin. Also observe that after this placement, the clique in slot \( q_j \) in bin \( b_j \) has grown in size by 1 but is still properly colored. So, this induction is well defined.

Note that the instance can be at most \( t^2 \) vertices long, since at that point there will some bin \( b \) containing at least \( t \) vertices, and therefore some slot in bin \( b \) containing at least \( \sqrt{t} \) vertices forming a clique of size

\( \Omega(\log d/\log \log d) \), whereas it is possible to schedule the jobs such that all machine loads across all dimensions are bounded by a constant. This lower-bound can also be extended to \textsc{Vsall-1} in order to establish Theorem [7].

Our lower-bound uses the \textsc{Monochromatic Clique} problem, formally defined as follows.

**Definition 11.** \textsc{Monochromatic Clique}: There are \( t \) colors that are given offline. The input graph is then revealed to the algorithm as an online sequence of \( n \) vertices \( v_1, \ldots, v_n \) that arrive one at a time. When vertex \( v_j \) arrives, we are given all edges between vertices \( v_1, v_2, \ldots, v_{j-1} \) and vertex \( v_j \). The algorithm must then assign \( v_j \) one of the \( t \) colors before it sees the next arrival. The objective is to minimize the size of the largest monochromatic clique in the final coloring.

Our main technical result is to prove the following lower bound for \textsc{Monochromatic Clique}.

**Lemma 12.** The competitive ratio of any online algorithm for \textsc{Monochromatic Clique} is \( \Omega(\sqrt{t}) \), where \( t \) is the number of available colors.
\(\sqrt{t}\). Since all the vertices in the clique are in the same bin, Lemma 12 follows. Thus, we are left to show that there exists a sequence of \(t^2\sqrt{t}\)-ary strings (recall that these define the adjacencies for each new vertex) such that the size of the largest monochromatic clique in the optimal coloring is \(O(1)\).

**Analysis:** First observe that monochromatic edges (i.e., edges between vertices of the same color) cannot form between vertices in slots \(s\) and \(s' \neq s\) (in the same or in different bins) since the color sets used for the slots are disjoint. Moreover, monochromatic edges cannot form within the same slot in the same bin since these vertices always form a properly colored clique. Therefore, monochromatic edges can only form between two adjacent vertices \(v_j\) and \(v_{j'}\) such that \(q_j = q_{j'}\) and \(b_j \neq b_{j'}\). We now make the following crucial observation:

**Observation 13.** Suppose \(K = \{u_{\phi(1)}, \ldots, u_{\phi(w)}\}\) is a \(w\)-sized monochromatic clique of color \(c \in C_\ell\) that forms during the instance, where \(\phi: [w] \rightarrow [t^2]\) maps \(k \in [w]\) to the index of the \(k\)th vertex to join \(K\) (note that \(b_{\phi(j)}\) are different for all \(i \in [w]\) from the above discussion). Then

\[
s_{\phi(j)}[b_{\phi(i)}] = \ell \quad \forall j \in \{1, \ldots, w\}, \forall i \in \{1, \ldots, j - 1\}.
\]

To see why Observation 13 is true, consider vertex \(u_{\phi(j)}\) (the \(j\)th vertex to join \(K\)). Since \(K\) is a clique, \(u_{\phi(j)}\) must be adjacent to vertices \(u_{\phi(1)}, \ldots, u_{\phi(j-1)}\). Since all these vertices are colored with \(c \in C_\ell\), they must have been placed in slot \(\ell\) in their respective bins. Therefore, the positions in \(s_{\phi(j)}\) that correspond to these bins must also be \(\ell\), i.e., \(s_{\phi(j)}[b_{\phi(i)}] = \ell\) for all pervious vertices \(u_{\phi(i)}\).

Using Observation 13 we show the existence of a good set of \(t^2\) strings using the probabilistic method. Suppose the adversary picks each \(s_j\) uniformly at random (i.e., for each character in \(s_j\) we pick \(w \in [\sqrt{t}]\) with probability \(t^{-1/2}\)). We define the following notation:

- Let \(K_{20}\) be the event that the adversary creates a monochromatic clique of size 20 or greater.
- Let \(K_{20}(S, c)\) be the event that a monochromatic clique \(K\) of color \(c\) and size 20 or greater forms such that the first 10 vertices to join \(K\) are placed in the bins specified by the set of 10 indices \(S = \{x_1, \ldots, x_{10}\}\).
- Let \(P_j(S, q)\) be a random variable that is 1 if \(s_j[i] = q\ \forall i \in S\) and 0 otherwise. Let \(P(S, q) = \sum_{j=1}^{t^2} P_j(S, q)\).
- Let \(q(c) \in [\sqrt{t}]\) to be the index of the color set to which color \(c\) belongs (i.e., \(c \in C_{q(c)}\)).
- Let \([n]_k := \binom{n}{k}\) denote the set of size-\(k\) subsets of \([n]\).

The next lemma follows from standard concentration inequalities (stated as Theorem 31 in Appendix A).

**Lemma 14.** If the adversary picks each \(s_j\) uniformly at random, then \(\Pr[P(S, q) \geq 10] \leq t^{-30}\).

**Proof.** First, we observe that for any set \(S \in [t]^{10}\) and any \(r \in \sqrt{t}\), we have \(\Pr[P_j(S, r) = 1] = (1/\sqrt{t})^{10} = t^{-5}\). Therefore by linearity of expectation, we have

\[
\mathbb{E}[P(S, r)] = \mathbb{E}\left[\sum_{j=1}^{t^2} P_j(S, r)\right] = t^2 \cdot t^{-5} = t^{-3}.
\]

Applying Theorem 31 to \(P(S, r)\) with setting \(X_i = P_i(S, r), \ a_i = 1, \ \delta = 10r^3 - 1\) and \(\mu = t^{-3}\) in Eqn. (1), we get

\[
\Pr[P(S, r) \geq 10] \leq \left(\frac{e^{10r^3-1}}{(10r^3)^{10^3}}\right)^{t^{-3}} \leq \left(\frac{e^{10}}{10^{10^3}}\right) \cdot \left(\frac{1}{t^{30}}\right) < t^{-30}.
\]

\(^{320}\) is an arbitrarily chosen large enough constant.
To show the existence of a good set of $t^2$ strings and establish Lemma 12, it is sufficient to show that $\Pr[K_{20}] < 1$; using Lemma 14, we in fact show this event occurs with low probability. Observe that

$$\Pr[K_{20}] \leq \sum_{c \in [V]} \sum_{S \in [I]} \Pr[K_{20}(S, c)] \leq \sum_{c \in [V]} \sum_{S \in [I]} \Pr[P(S, q(c)) \geq 10].$$

(2)

The first inequality is a straightforward union bound. The second inequality follows by Observation 13. If the event $K_{20}(S, c)$ occurs, then Observation 13 implies $s_j[b_i] = q(c)$ for $j = 11, \ldots, 20$, $i \in S$.

Since there are $\sqrt{t}$ possible colors and $|[I]| < t^{10}$, applying both (2) and Lemma 14 completes the proof of Lemma 12.

$$\Pr[K_{20}] \leq \sum_{c \in [V]} \sum_{S \in [I]} \Pr[P(S, q(c)) \geq 10] \leq \sum_{c \in [V]} \sum_{S \in [I]} t^{-30} \leq t^{1/2} \cdot t^{10} \cdot t^{-30} = t^{-39/2}.$$

We are now ready to use Lemma 12 to prove Theorem 4. We will describe a lower bound instance for VSMAX-1 whose structure is based on an instance of MONOCHROMATIC CLIQUE. This will allow us to use the lower bound instance from Lemma 12 as a black box to produce the desired lower bound for VSMAX-1.

We first set the problem definition of MONOCHROMATIC CLIQUE to be for $m$ colors where $m$ is also the number of machines used in the VSMAX-1 instance. Let $I_C$ be the lower-bound instance for this problem given by Lemma 12. This produces a graph $G$ of $m^2$ vertices such that the algorithm forms a monochromatic clique of size $\sqrt{m}$, whereas the largest monochromatic clique in the optimal solution is of size $O(1)$. Let $G_j = (V_j, E_j)$ be the graph in $I_C$ after vertices $v_1, \ldots, v_j$ have been issued (and so $G_n = G$). We define the corresponding lower-bound instance for VSMAX-1 as follows:

- There are $m^2$ jobs, which correspond to the vertices $v_1, \ldots, v_{m^2}$ from $I_C$.
- Each job has $d = \binom{m^2}{\sqrt{m}}$ dimensions, where each dimension corresponds to a specific $\sqrt{m}$-sized vertex subset of the $m^2$ vertices. Let $S_1, \ldots, S_d$ be an arbitrary ordering of these subsets.
- Job vectors will be binary. Namely, the $k$th vector entry for job $j$ is 1 if $v_j \in S_k$ and the vertices in $v_j \cap S_k$ form a clique (if $v_j \cap S_k = \{v_j\}$, then it is considered a 1-clique); otherwise, the $k$th entry is 0.
- Let $c_1, \ldots, c_m$ define an ordering on the available colors from $I_C$. We match each color from $I_C$ to a machine in our scheduling instance, and thus machine assignments are given by vertex colorings. Formally, if vertex $v_j$ is assigned color $c_i$, then job $j$ is placed on machine $i$ in the scheduling instance.

**Analysis:** Since machine assignments correspond to the colorings from $I_C$, it follows that the largest load in dimension $k$ is the size of the largest monochromatic sub-clique in $S_k$. $I_C$ is given by the construction in Lemma 12 therefore at the end of the instance, there will exist a dimension $k'$ such that the online algorithm colored every vertex in $S_{k'}$ with some color $c_i$. Thus, machine $i$ will have $\sqrt{m}$ load in dimension $k'$. In contrast, Lemma 12 ensures that all the monochromatic cliques in the optimal solution are of size $O(1)$, and therefore the load on every machine in dimension $k'$ is $O(1)$.

Finally, we show the following relationship between $m$ and $d$, which by the above discussion completes the proof of Theorem 4.

**Lemma 15.** If $d = \binom{m^2}{\sqrt{m}}$, then $\sqrt{m} = \Omega(\log d / \log \log d)$.

**Proof.** We will use the following well-known bounds on $\binom{n}{k}$: for integers $0 \leq k \leq n$, $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k \leq \left(\frac{em}{k}\right)^k$. First, we observe that

$$\log d = \log \left(\frac{m^2}{\sqrt{m}}\right) \leq \log \left(\frac{em^2}{\sqrt{m}}\right) = \log(e\sqrt{m} \cdot m^{3/2} \sqrt{m}) = \sqrt{m} \cdot (1 + (3/2) \log m).$$

(3)
We also have
\[
\log \log d = \log \log \left( \frac{m^2}{\sqrt{m}} \right) \geq \log \log \left( \frac{m^2}{\sqrt{m}} \right)^{\frac{1}{\sqrt{m}}} \geq \log((3/2)\sqrt{m}\log m) \geq (1/2)\log m.
\]

Hence, combining these two bounds Eqns. (3) and (4), we obtain
\[
\sqrt{m} \geq \frac{\log d}{1 + (3/2)\log m} \geq \frac{\log d}{1 + 3\log \log d},
\]
which implies that \( \sqrt{m} = \Omega(\log d / \log \log d) \), as desired. \( \square \)

We now shift our attention to proving Theorem [7], i.e., the lower bound for any \( L_p \) norm. As before, our lower bound construction forces any algorithm to schedule jobs so that there exists a dimension \( k' \) where at least one machine in \( k' \) has load at least \( \sqrt{m} \), whereas the optimal schedule maintains the property that every machine load is bounded by some constant \( C \). Since any dimension receives at most \( \sqrt{m} \) jobs with load 1, any assignment ensures that there are at most \( \sqrt{m} \) machines with non-zero load in a given dimension. Therefore, in the optimal solution, the \( L_r \)-norm of the load vector for dimension \( k' \) is at most \( (C' \cdot \sqrt{m})^{1/r} = C \cdot m^{1/(2r)} \).

Thus, the ratio between the objective of the solution produced by the online algorithm and the optimal solution is at least \( m^{1/2}/(C \cdot m^{1/(2r)}) = (1/C) \cdot m^{(r-1)/(2r)} \). Using Lemma[15] we conclude Theorem[7].

### 2.2 Upper Bound for VSALL-I

In this section we prove Theorem[6] we will give an online algorithm for VSALL-I and show it is \( O((\log d / \log \log d)^{1/\sqrt{d}}) \)-competitive, i.e., our algorithm guarantees this bound in any dimension if one choose to evaluate it with the \( L_r \)-norm. As mentioned earlier, we achieve this goal by reducing this problem to VSMAX-I. More specially, we show an \( O((\log d / \log \log d)) \)-competitive algorithm for VSMAX-I and that it can be translated to an \( O((\log d / \log \log d)^{1/\sqrt{d}}) \)-competitive algorithm for VSALL-I.

We assume in VSALL-I that the online algorithm is aware of the total volume of jobs on each dimension a priori. This assumption can easily be replaced with the one that the optimal makespan for VSMAX-I is known to the algorithm. We note that the lower bounds claimed in Theorems[4] and[7] are robust against this assumption since the optimal makespan is always a constant and this knowledge does not help the online algorithm. Further, the \( O((\log d / \log \log d)) \)-competitiveness we prove for VSMAX-I does not require the assumption. However, throughout this section, we present our results under the assumption to make our presentation more transparent while keeping notation simple.

**Algorithm:** Our algorithm will operate on a modified instance. For each job \( j \) that arrives online, we apply the following two transformations

- **Transformation 1:** Recall that for VSALL-I, we are given a volume vector \( V = \langle V_1, \ldots, V_d \rangle \) indicating that the total load in dimension \( k \) is \( V_k \); therefore, we normalize \( p_j(k) \) by dividing it by \( V_k / m \) (for ease of notation, we will still refer to this normalized value as \( p_j(k) \)).

- **Transformation 2:** Next, if \( p_j(k) > 1 \), we reduce \( p_j(k) \) to be 1. If this load reduction is applied in dimension \( k \) for job \( j \), we say \( j \) is large in \( k \); otherwise, \( j \) is small in dimension \( k \).
Note that we are applying these transformations in sequence (first Transformation 1 and then Transformation 2). Observe that now we have that \( \sum_{j \in J} p_j(k) = m \) for all \( k \in [d] \), and \( p_j(k) \leq 1 \) for all \( j \in J, k \in [d] \); we say that instance of \( \text{VSALL-1} \) is well-behaved if these two properties hold.

To make the actual job assignments on this modified instance, we will use a makespan scheduler as a black-box. For an \( \alpha \geq 1 \), we say an online algorithm for well-behaved instances is \( \alpha \)-competitive on makespan if for all instances we have that

\[
\max_{k \in [d]} \max_{i \in M} \Lambda_i(k) \leq \alpha,
\]

where \( \Lambda_i(k) \) denotes the load the algorithm has on machine \( i \) in dimension \( k \). Algorithms in previous work (such as Azar et al. \cite{azar1994finding} and Meyerson et al. \cite{meyerson2006note}) can be adapted to yield a \( O(\log d) \)-competitive algorithm for such instances; however, in order to match our lower-bound, we will give an algorithm that tightens this bound by achieving competitive ratio of \( O(\log d / \log \log d) \). This will prove Theorem \ref{thm:competitive} under the assumption discussed above; again, we do not need the assumption to derive Theorem \ref{thm:competitive}. We present the description and analysis of this algorithm in the following section (Section \ref{sec:algorithm}).

Thus for our algorithm, we simply use the makespan scheduler from Theorem \ref{thm:competitive} after transforming each job. To show our desired upper-bound for \( \text{VSALL-1} \), it is sufficient to establish the following theorem:

**Theorem 16.** If we use the assignments defined by an online algorithm \( A \) that is \( \alpha \)-competitive on makespan in the transformed instance, then the resulting online algorithm on the original instance is \( O(\alpha^{r-1}/r) \)-competitive, i.e., for any \( k \in [d] \), this is the bound we obtain on the ratio between \( \|\Lambda(k)\|_r \) and the optimal load vector when using the \( L_r \)-norm as the objective.

Clearly if we can establish competitiveness for the scaled instance (i.e. after just applying Transformation 1 to all jobs), then our algorithm is competitive in the original instance, as well. Let \( \text{OPT}'(k, r) \) be the cost of the optimal solution on the scaled loads in dimension \( k \). First observe that we can lower-bound \( \text{OPT}'(k, r) \) as follows.

**Proposition 17.** \( \text{OPT}'(k, r) \geq \max \left( \sum_{j \in J} p_j(k)^r, \left( \sum_{j \in J} p_j(k)/m \right)^r \cdot m \right) = \max \left( \sum_{j \in J} (p_j(k))^r, m \right) \).

**Proof.** Consider any fixed assignment of jobs, and let \( J'(i) \subseteq J \) be the set of jobs assigned to machine \( i \). Consider any fixed \( k \). The first lower bound follows since \( \sum_{i \in M} (\sum_{j \in J'(i)} p_j(k))^r \geq \sum_{i \in M} \sum_{j \in J'(i)} p_j(k)^r = \sum_{j \in J} p_j(k)^r \). The second lower bound is due to convexity of \( x^r \) when \( r \geq 1 \).

Let \( J(i) \subseteq J \) be the set of jobs assigned to machine \( i \). Let \( \ell(i, k) \) and \( s(i, k) \) be the set of jobs assigned to machine \( i \) that are large and small in dimension \( k \), respectively. Observe that since the black-box makespan algorithm \( A \) is \( \alpha \)-competitive on the well-behaved instance, we have that \( |\ell(i, k)| \leq \alpha \) and that \( \sum_{j \in \ell(i, k)} p_j(k) \leq \alpha \) for all \( i \in M, k \in [d] \).

We first bound how much the large jobs on machine \( i \) in dimension \( k \) can contribute to the online
algorithm’s objective. Let \( h = |\ell(i,k)| \). Then, it follows that

\[
\left( \sum_{j \in \ell(i,k)} p_j(k) \right)^r = \left( \frac{1}{h} \sum_{j \in \ell(i,k)} (p_j(k) \cdot h) \right)^r \\
\leq \frac{1}{h} \sum_{j \in \ell(i,k)} (p_j(k) \cdot h)^r \quad \text{(Due to convexity of } x') \\
= h^{r-1} \sum_{j \in \ell(i,k)} p_j(k)^r \\
\leq \alpha^{r-1} \sum_{j \in \ell(i,k)} p_j(k)^r \quad \text{(since } h = |\ell(i,k)| \leq \alpha \forall i \in M, k \in [d]).
\] (5)

We now bound the contribution of the small jobs in a given dimension \( k \). Our goal is to upper bound \( \sum_{i \in M} \left( \sum_{j \in s(i,k)} p_j(k) \right)^r \). Recall that \( \sum_{i \in M} \left( \sum_{j \in s(i,k)} p_j(k) \right) \leq m \). Further we know that \( \sum_{j \in s(i,k)} p_j(k) \leq \alpha \) for all \( i \). Hence \( \sum_{i \in M} \left( \sum_{j \in s(i,k)} p_j(k) \right)^r \) is upper bounded by the bound claimed in the following lemma whose proof easily follows from convexity of function \( x' \).

**Lemma 18.** Let \( f(x) = x^r \) for some \( r \geq 1 \) whose domain is defined over a set of variables \( x_1, \ldots, x_m \in [0, \alpha] \) where \( \alpha \geq 1 \) with another constraint \( \sum_{i=1}^{m} x_i \leq m \). Then, we have that \( \sum_{i=1}^{m} f(x_i) \leq 2m \alpha^{r-1} \).

**Proof.** We claim that the LHS is maximized when there is at most one \( i \in [m] \) with \( 0 < x_i < \alpha \). If we have two such variables \( x_i \) and \( x_j \) with \( 0 < x_i \leq x_j < \alpha \), it is easy to see that we can further increase the LHS by increasing \( x_i \) and decreasing \( x_j \) by an infinitesimal equal amount (i.e. \( x_i \leftarrow x_i - \varepsilon \) and \( x_j \leftarrow x_j + \varepsilon \)) due to convexity of \( f \).

Hence, the LHS is maximized when the multi-set \( \{x_i : i \in [m]\} \) has \( \lfloor m/\alpha \rfloor \) copies of \( \alpha \), and one copy of \( m - \alpha \lfloor m/\alpha \rfloor \) (which can be equal to \( \alpha \)), which gives,

\[
\sum_{i=1}^{m} f(x_i) \leq \lfloor m/\alpha \rfloor f(\alpha) + f(m - \alpha \lfloor m/\alpha \rfloor).
\]

If \( \lfloor m/\alpha \rfloor \geq 1 \), then it immediately follows that (LHS) \( \leq \lfloor m/\alpha \rfloor f(\alpha) \leq 2(m/\alpha)\alpha^{r-1} \). Otherwise, we know (LHS) \( \leq f(m) = m^r \leq m\alpha^{r-1} \) since \( m < \alpha \).

**Corollary 19.** For all \( k, \sum_{i \in M} \left( \sum_{j \in s(i,k)} p_j(k) \right)^r \leq 2m \alpha^{r-1} \).

Using (5) and Corollary 19, we obtain our desired bound

\[
\sum_{i \in M} \left( \sum_{j \in \ell(i)} p_j(k) \right)^r = \sum_{i \in M} \left( \sum_{j \in \ell(i,k)} p_j(k) + \sum_{j \in s(i,k)} p_j(k) \right)^r \\
\leq \sum_{i \in M} \left( 2 \max \left( \sum_{j \in \ell(i,k)} p_j(k), \sum_{j \in s(i,k)} p_j(k) \right) \right)^r \\
\leq 2^r \sum_{i \in M} \left( \left( \sum_{j \in \ell(i,k)} p_j(k) \right)^r + \left( \sum_{j \in s(i,k)} p_j(k) \right)^r \right) \\
\leq 2^r \left( \alpha^{r-1} \sum_{j \in \ell} p_j(k)^r + 2m \cdot \alpha^{r-1} \right) \quad \text{(by (5) and Lemma 18)} \\
\leq (2^{r+1} \alpha^{r-1}) \OPT^r(k, r)^r \quad \text{(by Proposition 17)}
\]
which, raising both the LHS and RHS to the $1/r$, gives us

$$||\Lambda(k)||_r \leq (2^{2^{1/r}} \alpha^{r-1})^{O(T'(k,r))} = O(\alpha^{r-1})^{O(T'(k,r))}.$$ 

### 2.3 Makespan for VSALL-I: Proof of Theorem [5]

We first discuss some preprocessing step that will be useful for our algorithms and analysis. Then, we present a simple randomized $O(\log d / \log \log d)$-competitive algorithm, and show how to derandomize it. We note that the preprocessing step is not needed for the randomized algorithm, but is required for derandomization. We can easily get rid of the assumption that the total volume of jobs on each dimension is known to the algorithm a priori by updating the threshold used in the first procedure, based on the maximum job load on any dimension and the total volume of jobs the algorithm has observed so far.

#### 2.3.1 Preprocessing

Before running our algorithm, we make another small change to the input instance. A similar preprocessing step was used in [6].

- **Transformation 3**: For each job $j$ and $k \in [d]$, if $p_j(k) < (1/d) \max_k p_j(k')$, then increase $p_j(k)$ to $(1/d) \max_k p_j(k')$.

**Lemma 20.** Transformation 3 increases the optimal makespan by a factor of at most 2.

**Proof.** Consider a fixed machine $i$. Let $J^*(i)$ denote the jobs assigned to machine $i$ by the optimal solution, and $\Lambda^*_i(k)$ the optimal solution’s load on machine $i$ in dimension $k$. We know that $\sum_{j \in J^*(i)} \sum_{k \in [d]} p_j(k)$, the total load on machine $i$ over all dimensions before the transformation, is at most $d \max_k p_j(k)$, and a job $j'$ can increase the load on each dimension $k$ by at most $(1/d) \max_k p_j(k')$. Since $\sum_{j \in J^*(i)} \sum_{k \in [d]} p_j(k) \leq \sum_{j \in J^*(i)} \sum_{k' \in [d]} p_j(k') \leq d \max_k \Lambda^*_i(k')$, the load on every dimension $k$ can increase by at most $(1/d) \cdot d \max_k \Lambda^*_i(k')$, which gives the lemma.

Lemma [20] implies that an algorithm that is $\alpha$-competitive for the modified instance yields a schedule that is $2\alpha$-competitive for the original instance since Transformation 3 only increases jobs loads. Recall that we had that all $k \in [d]$, $\sum_j p_j(k) \leq m$ after Transformations 1 and 2. We show that this property is preserved within a constant factor.

**Lemma 21.** After performing Transformation 3, we have that for all $k \in [d]$, $\sum_j p_j(k) \leq 2m$.

**Proof.** Consider any fixed dimension $k' \in [d]$. Due to Transformation 3, each job $j$’s load on dimension $k'$ increases by at most $(1/d) \max_k p_j(k)$. Hence the increase in the total load of jobs on dimension $k'$ is at most $\sum_j (1/d) \max_k p_j(k) \leq (1/d) \sum_j \sum_k p_j(k) \leq (1/d) md \leq m$. The following are the properties of jobs we have after performing Transformations 1, 2 and 3 in sequence.

- **Property 1.** For all $k \in [d]$, $\sum_j p_j(k) \leq 2m$.
- **Property 2.** For all $j$ and $k \in [d]$, $0 \leq p_j(k) \leq 1$.
- **Property 3.** For all $j$ and $k \in [d]$, $(1/d) \max_k p_j(k') \leq p_j(k) \leq \max_k p_j(k')$. 

12
- **Property 4.** The optimal makespan is at least 1.

Property 1 is a restatement of Lemma 21. Property 2 remains to hold true as was shown after the first two transformations since Transformation 3 has no effect on this property. Property 3 is a direct consequence of Transformation 3. To see why Property 4 is true, note that if Transformation 2 of decreasing \( p_j(k) \) to 1 was performed, then \( p_j(k) = 1 \) for some \( j \) and \( k \), which immediately implies Property 4. Otherwise, \( \sum_j p_j(k) = m \) for all \( k \), which again leads to Property 4 by a simple volume argument.

Since our goal is to obtain a \( O(\log d / \log \log d) \)-competitive algorithm and the transformations can be performed online, from the above discussion we can wlog assume that the original instance satisfies the above properties.

### 2.3.2 Randomized Algorithm

In this section, we give a simple randomized algorithm that yields a schedule whose expected makespan is \( O(\log d / \log \log d) \). The online algorithm consists of two procedures, random assignment and greedy packing. It will be convenient to assume that we have two disjoint sets \( M_1, M_2 \) of their respective \( m \) identical machines which will be used by the two procedures respectively. Each machine in \( M_1 \) is paired up with an arbitrary distinct machine in \( M_2 \), and the actual load on an actual machine will be evaluated as the sum of the loads on the corresponding pair of machines. Note that this “doubling” machines can only double the competitive ratio. Hence it suffices to show that each procedure produces a schedule with makespan \( O(\log d / \log \log d) \). Define a parameter \( \alpha := \frac{10 \log d}{\log \log d} \) which will be used in the algorithms and analysis.

- **First procedure (random assignment):** Assign each job to one of the machines in \( M_1 \) uniformly at random. Let \( J^1_i(i) \) denote the subset of jobs \( \{1, 2, ..., j\} \) that are assigned to machine \( i \) in this procedure and let \( \Lambda^1_{i,j}(k) \) denote the resulting load on machine \( i \) on dimension \( k \) due to jobs in \( J^1_i(i) \). If \( \Lambda^1_{i,j}(k) \geq 2 \alpha + 1 \) for some \( k \in [d] \), then we pass the job \( j \) to the second procedure. Here job \( j \)'s contribution to \( \Lambda^1_{i,j}(k) \) remains the same although job \( j \) actually is scheduled by the second procedure to a certain machine in \( M_2 \).

- **Second procedure (greedy packing):** This procedure is only concerned with the jobs \( J^2 \) that are passed from the first procedure. It allocates each job in \( J^2 \) (in the order jobs arrive) to one of the machines in \( M_2 \) such that the resulting makespan, \( \max_{i \in M_2, k \in [d]} \Lambda^2_{i,j}(k) \) is minimized; \( \Lambda^2_{i,j}(k) \) is analogously defined for this second procedure as above.

This completes the description of the algorithm. We will let \( J^1_i(i) := J^1_i \) and \( \Lambda^1_{i,k}(k) := \Lambda^1_{i,n}(k) \), and define \( J^2(i) \) and \( \Lambda^2_{i,k}(k) \) similarly. We emphasize again that jobs in \( J^2 \) are scheduled on \( M_2 \), and the other jobs are on \( M_1 \).

We start with an easy observation that the makespan of the jobs scheduled on machines \( M_1 \) is not a concern.

**Observation 22.** The first procedure schedules jobs, \( [n] \setminus J^2 \) on machines \( M_1 \) such that the makespan is \( O(\log d / \log \log d) \).

We shift our attention to the makespan the second procedure yields. The following key lemma shows that the second procedure receives only a tiny fraction of volume of jobs which will allow us to use a quite inefficient greedy packing that can be only \( O(d) \)-competitive in general. The proof is a simple consequence of a standard Chernoff-Hoeffding bound. For simplicity, we will assume that \( d \) is greater than a sufficiently large constant.
Lemma 23. The probability that a job \( j \) is passed to the second procedure is at most \( 1/d^3 \), i.e. \( \Pr[j \in \mathcal{J}^2] \leq 1/d^3 \).

Proof. Fix \( i, j \) and \( k \). Say job \( j \) was assigned to \( i \). To upper bound \( \Pr[A^1_{i,j}(k) \geq 2\alpha + 1] \), it suffices to bound \( \Pr[A^1_{i,j-1}(k) \geq 2\alpha] \) since \( p_j(k) \leq 1 \) due to Property 2. We will show

\[
\Pr[A^1_{i,j-1}(k) \geq 2\alpha] \leq 1/d^3.
\]

Here the probability space is over the random choices of jobs 1, 2, ..., \( j-1 \). The lemma will follow from this upper bound via a union bound over all dimensions \( k \in [d] \).

To show (6), we use the Chernoff-Hoeffding bound (see Theorem \[31\]). To apply the inequality, create random variables \( X_1, X_2, ..., X_{j-1} \) where \( X_j \) is a 0-1 random variable that is 1 if job \( j \) is assigned to machine \( i \). Set \( a_j = p_j(k) \), and \( \mu = 2 \). Note that \( E\Lambda^1_{i,j-1}(k) \leq 2 \) due to Property 1 and the fact that each job is assigned to one of the \( m \) machines uniformly at random. Also set \( \delta = \alpha - 1 \). Then, we have

\[
\Pr[A^1_{i,j-1}(k) \geq 2\alpha] = \Pr[\sum_{j' \in [j-1]} a_{j'}X_{j'} \geq \alpha\mu] = \Pr[\sum_{j' \in [j-1]} a_{j'}X_{j'} \geq (1 + \delta)\mu]
\]

\[
\leq \left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu
\]

\[
\leq \frac{e^\delta}{(1 + \delta)^{1+\delta}}
\]

\[
\leq 1/(5\log d/\log \log d)^{(5\log d/\log \log d)}
\]

\[
\leq 1/d^3
\]

as desired. \( \Box \)

The following lemma upper bounds the second procedure’s makespan in terms of the total volume of jobs it gets, i.e. \( \sum_{j \in \mathcal{J}^2, k \in [d]} p_j(k) \), denoted as \( V_2 \).

Lemma 24. \( \max_{i \in \mathcal{M}_2, k \in [d]} \Lambda^2_i(k) \leq \frac{1}{m}V_2 + 1 \)

Proof. For the sake of contradiction, suppose this is not the case. Since jobs are greedily assigned so as to minimize the makespan, the values \( \Lambda^2_i(k) \) over different \( i \) and \( k \) must differ by at most \( \max_{j,k} p_j(k) \) which is at most 1 due to Property 2. This implies that every machine must have load greater than \( \frac{1}{m}V_2 \) on some dimension, meaning that the total load over all machines and dimensions is greater than \( V_2 \), which is a contradiction to the definition of \( V_2 \). \( \Box \)

We are now ready to complete the analysis. From Lemma 23, we know that \( EV_2 \leq \frac{1}{d^3} \sum_{j,k \in [d]} p_j(k) \leq \frac{1}{d^3} \cdot 2dm = \frac{m}{d} \) due to the linearity of expectation and Property 1. Combining this with Lemma 24 shows that the second procedure yields an expected makespan of \( O(1) \).

2.3.3 Derandomization

Our derandomization borrows the technique developed in [12]. To derandomize the algorithm, we replace the first procedure, random assignment with a deterministic assignment guided by the following potential
Let $f(x) := \alpha^x$ for notational simplicity. Recall that $\alpha := 10\log d / \log \log d$.

$$
\Phi_{i,k}(j) := f \left( \Lambda_{i,j}(k) - \frac{\alpha}{m} \sum_{j \in [j]} p_j(k) \right) \quad \forall i \in M_1, j \in [n], k \in [d]
$$

$$
\Phi(j) := \sum_{i \in M} \sum_{k=1}^d \Phi_{i,k}(j)
$$

- **(New deterministic) first procedure.** Each job $j$ is assigned to a machine $i$ such that $\Phi(j)$ is minimized. If $\Lambda_{i,j}(k) \geq 3\alpha + 1$, then $j$ is added to queue $J^2$ so that it can be scheduled by the second procedure. As before, each job is scheduled by either the first procedure or the second, and contributes to the “virtual” load $\Lambda_{i,j}(k)$ in either case.

**Lemma 25.** $\Phi(j)$ is non-increasing in $j$.

**Proof.** Fix the assignment of jobs $1, 2, \ldots, j - 1$ which determines the value of $\Phi(j - 1)$. We do the thought process of assigning job $j$ to one of the $m$ machines in $M_1$ uniformly at random. We will show that $E[\Phi(j)] \leq \Phi(j - 1)$. This will imply the existence of a machine $i$ such that assigning job $j$ to the machine $i$ leads to $\Phi(j) \leq \Phi(j - 1)$, and such an assignment is actually found by the algorithm since its assignment minimizes the potential increase. For any fixed $i$, $j$, and $k$, we derive

$$
E[\Phi_{i,k}(j)] = \frac{1}{m} \left( \Lambda_{i,j-1} + \alpha \cdot p_j(k) - \alpha \cdot \frac{1}{m} \sum_{j \in [j-1]} \Lambda_{i,j-1}(k) \right)
$$

$$
+ \left( 1 - \frac{1}{m} \right) f \left( \Lambda_{i,j-1} - \alpha \cdot \frac{1}{m} p_j(k) - \frac{\alpha}{m} \sum_{j \in [j-1]} \Lambda_{i,j-1}(k) \right)
$$

$$
= \Phi_{i,k}(j - 1) \cdot \alpha \cdot \frac{1}{m} p_j(k) \cdot \left( \frac{1}{m} \right) \left( \alpha p_j(k) - 1 + 1 \right)
$$

$$
\leq \Phi_{i,k}(j - 1) \cdot \alpha \cdot \frac{1}{m} p_j(k) \cdot \left( \frac{p_j(k)}{m} \right) \left( \alpha - 1 + 1 \right)
$$

$$
\leq \Phi_{i,k}(j - 1) \cdot \exp \left( -(\alpha \log \alpha) \cdot \frac{p_j(k)}{m} \right) \cdot \exp \left( \frac{p_j(k)}{m} \cdot (\alpha - 1) \right)
$$

$$
\leq \Phi_{i,k}(j - 1)
$$

Inequality (7) follows since $e^x - 1 \leq (\alpha - 1)x$ for $x \in [0, 1]$, and $p_j(k) \leq 1$ due to Property 2. The inequality (8) is due to the fact $x + 1 \leq e^x$. This, due to linearity of expectation, proves the claim $E[\Phi(j)] \leq \Phi(j - 1)$, thereby completing the proof of the lemma.

Lemma 25 and the simple observation that $\Phi(0) = md$ give the following bound.

**Corollary 26.** $\Phi(n) \leq md$.

As before, it is straightforward to see that jobs that are actually scheduled by the first procedure has makespan $O(\alpha)$. Hence we focus on the second procedure. Our main goal is to bound $V_2 = \sum_{j \in J^2} \sum_{k \in [d]} p_j(k)$.

**Lemma 27.** $V_2 \leq m/d$. 

15
Proof. For each \( j \in J_2 \), define \( k(j) \) to be an arbitrary dimension \( k \) with \( \Lambda^1_{i,j}(k) \geq 3\alpha + 1 \) — such a dimension \( k \) exists since \( j \) was added to \( J^2 \) to be scheduled by the second procedure. Then, we upper bound

\[
V_2 = \sum_{j \in J^2} \sum_{k' \in [d]} p_j(k')
\]

\[= \sum_{i \in M_1} \sum_{k \in [d]} \sum_{j \in J^1} \sum_{k' \in [d]} \sum p_j(k') \]

\[= \sum_{i \in M_1} \sum_{k \in [d]} \sum_{j \in J^1} \sum_{k' \in [d]} d^2 p_j(k) \]

\[\leq d^2 \sum_{i \in M_1} \sum_{k \in [d]} (\Lambda^1_i(k) - 3\alpha)_+ \quad (11)\]

Equation (\(9\)) follows since each job \( j \) in \( J^2 \) is associated with a unique \( k(j) \in [d] \) and is assigned to a unique machine \( i \) by the first procedure. Inequality (\(10\)) is due to Property 3. Recall that \( \Lambda^2_{i,j}(k) \geq 3\alpha + 1 \) when \( j \in J^1(i) \) and \( k = k(j) \). This can happen only when \( \Lambda^1_{i,j-1}(k) \geq 3\alpha \) since \( p_j(k) \leq 1 \) due to Property 2. Since \( \Lambda^1_{i,j}(k) \) is non-decreasing in \( j \), the sum of \( p_j(k) \) over all such jobs \( j \) is at most \( (\Lambda^1_i(k) - 3\alpha)_+ \); here \( (x)_+ := \max\{0, x\} \).

We claim that for all \( i \in M_1, k \in [d], \)

\[\Phi_{i,k}(n) \geq \alpha^\alpha (\Lambda^1_{i,j}(k) - 3\alpha)_+ \quad (12)\]

If \( \Lambda^1_{i,j}(k) - 3\alpha \leq 0 \), then the claim is obviously true since \( \Phi_{i,k}(n) \) is always non-negative. Otherwise, we have \( \Phi_{i,k}(n) \geq \alpha^{\Lambda^1_{i,j}(k) - 3\alpha} = \alpha^{\alpha (\Gamma_{i,j}(k) - 3\alpha)} \) where the first inequality follows from Property 1. So in either case, (12) holds true.

By combining (9), (11), Corollary 26 and and recalling \( \alpha = \frac{10 \log d}{\log \log d} \), we have

\[V_2 \leq d^2 \sum_i \sum_k (\Lambda^1_i(k) - 3\alpha)_+ \leq \frac{d^2}{\alpha^\alpha} \sum_i \sum_k \Phi_{i,k}(n) \leq \frac{d^3}{\alpha^\alpha} m \leq \frac{m}{d} \]

\[\Box\]

By Lemma 24 we have \( \max_{i \in M_2, k \in [d]} \Lambda^2_i(k) \leq \frac{1}{m} V_2 + 1 \leq O(1) \). Thus, we have shown that each of the two deterministic procedures yields a makespan of \( O(\alpha) = O(\log d / \log \log d) \), thereby proving Theorem 5.

3 Unrelated Machines

3.1 Lower Bound

In this section we prove Theorem 10, i.e., we show that we can force any algorithm to make an assignment where there exists a dimension that has cost at least \( \Omega(\log d + r_k) \). We model the proof as a contest between an adversary and an online algorithm \( A \). The adversary begins by issuing a job sequence similar to those used in the lower bounds in [13] and [4]; however, we restrict \( A \) to a single machine in each dimension.

Let \( d = 2^h \) for a sufficiently large integer \( h \). Fix the number of machines \( m \) to be \( d \). Our construction will ensure that the only machine accumulates non-zero load in dimension \( k \) is machine \( k \). Intuitively, each job \( j \) will only have two machines to choose from: \( m_1(j), m_2(j) \in \{1, \ldots, m\} \). If \( j \) is scheduled on a machine \( i \not\in \{m_1(j), m_2(j)\} \), then infinite load is added to \( i \) in all dimensions (therefore the algorithm is forced to avoid this assignment). If \( i \in \{m_1(j), m_2(j)\} \), then a load of one is added to machine \( i \) in dimension \( i \), and in all other dimensions 0 load is added. Formally, \( p_{i,j}(k) \) is defined as follows
that their respective dimension scheduled. Our goal is to prove Theorem 9, i.e.,

\[
p_{i,j}(k) = \begin{cases} 
0 & \text{if } i \neq k \text{ and } i \in \{m_1(j), m_2(j)\} \\
1 & \text{if } i = k \text{ and } i \in \{m_1(j), m_2(j)\} \\
\infty & \text{otherwise.}
\end{cases}
\]

The adversary begins by setting the target parameter for each dimension to be 1, i.e., \(T_k = 1\) for \(k = 1, \ldots, d\); by the end of the proof, it will be clear that it is possible to schedule the jobs in the given instance such that the cost on each dimension is one. Note that because each dimension is equivalent with respect to the given target parameters, we need not divide the proof into cases that consider how \(A\) might use this information.

The adversary then proceeds by issuing jobs in \(h + 1\) phases \(1, \ldots, h + 1\). From phase to phase, we maintain a set of “active” machines to which we will still schedule jobs, where all machines are initially set to be active. When a machine \(i\) does not receive a job in a given phase, we inactivate \(i\), i.e., for the rest of the instance there will be no job \(j\) such that \(m_1(j) = i\) or \(m_2(j) = i\) (which, based on the above discussion, forces \(A\) to stop scheduling jobs on \(i\)).

More specifically, let \(S_\ell\) denote the active machines in phase \(\ell\). In the \(\ell\)th phase for \(\ell \leq h\), we issue a set of jobs \(J_\ell\) where \(|J_\ell| = 2^{h-\ell}\). To define these jobs, we arbitrarily pair off the machines in \(S_\ell\) and use each machine pair as \(m_1(j)\) and \(m_2(j)\) for a unique job \(j \in J_\ell\). As discussed above, the algorithm will be forced to pick between scheduling \(j\) on \(m_1(j)\) or \(m_2(j)\), and therefore exactly \(2^{h-\ell}\) machines will accumulate an additional load of 1 in phase \(\ell\). Machines that receive jobs in phase \(\ell\) remain active in phase \(\ell + 1\), and machines that do not receive jobs become inactive and remain that way for phases \(\ell + 1, \ldots, h + 1\). The last phase \(h + 1\) is slightly different from previous ones. In phase \(h + 1\), we will simply issue a single job that must be scheduled on the last machine remaining active (and so in this case, there is no choice between two machines). Call this last job \(j'\).

Based on this construction, there will exist a dimension \(k'\) at the end of the job sequence that has load \(h + 1\) on machine \(k'\) and 0 on all other machines (the dimension with the machine that remains active through every phase). Observe that in the optimal schedule, jobs are assigned so that each machine \(i\) has load one in their respective dimension \(i\). This schedule can be obtained by reversing the assignments made by \(A\), i.e., if \(A\) scheduled \(j\) on \(m_1(j)\), then the optimal schedule assigns \(j\) to \(m_2(j)\) (and vice versa), with the exception that \(j'\) is assigned to the machine which \(A\) also assigned \(j'\).

In the case that \(\log d \geq r_{k'}\), the adversary stops. Since \(T_{k'} = 1\) and \(L_{k'} = h + 1 = 1 + \log d\), we have that \(L_{k'} \geq \alpha T_{k'}\) where \(\alpha = \log d\). Since \(\log d \geq r_{k'}\), it follows \(\alpha = \Omega(\log d + r_{k'})\).

If \(\log d < r_{k'}\), then the adversary stops the current instance and begins a new instance. In the new instance, we simply simulate the lower-bound instance in \(d\) in dimension \(k'\) (i.e., the only dimension that receives load is dimension \(k'\); we would also reset our load vectors accordingly). Here, the adversary forces the algorithm to be \(\Omega(r_{k'})\)-competitive, which, since \(\log d < r_{k'}\), gives us the desired bound of \(\Omega(\log d + r_{k'})\).

### 3.2 Upper Bound

Our goal is to prove Theorem \([\square]\) i.e., \(\|A(k)\|_{r_k} = O(\log d + r_k) \cdot T_k\) for all \(k \in [d]\) where for dimension \(k\), \(A(k)\) is the load vector produced by our online algorithm, \(r_k\) is the norm that we are optimizing, and \(T_k\) is the target objective.

First, we normalize \(p_{i,j}(k)\) to \(p_{i,j}(k)/T_k\) for all dimensions \(k\); to keep the notation simple, we will also denote this normalized load as \(p_{i,j}(k)\). This ensures that the target objective is 1 in every dimension. (We assume \(wlog\) that \(T_k > 0\) for all \(k \in [d]\). If \(T_k = 0\), the algorithm discards all assignments that put non-zero load on dimension \(k\).

**Algorithm.** As described in the introduction, our algorithm is greedy with respect to a potential function defined on modified \(L_{r_k}\) norms. Let \(L_k = \|A(k)\|_{r_k}\) denote the \(L_{r_k}\) norm of the machine loads in the \(k\)th
dimension, and \( q_k = r_k + \log d \) denote the desired competitive ratio.\footnote{All logs are base 2.} We define the potential for dimension \( k \) as \( \Phi_k = L_k^q \). The potentials for the \( d \) different dimensions are combined using a weighted linear combination, where the weight of dimension \( k \) is \( \alpha_k = (3q_k)^{-q_k} \). Note that dimensions that allow a smaller slack in the competitive ratio are given a larger weight in the potential. We denote the combined potential by \( \Phi = \sum_{k=1}^{d} \alpha_k \cdot \Phi_k \). The algorithm assigns job \( j \) to the machine that minimizes the increase in potential \( \Phi \).

**Analysis.** Let us fix a solution satisfying the target objectives, and call it the optimal solution. Let \( \Lambda_i(k) \) and \( \Lambda^*_i(k) \) be the load on the \( i \)th machine in the \( k \)th dimension for the algorithmic solution and the optimal solution respectively. We also use \( L_k^* \) to denote the \( L_k \) norm in the \( k \)th dimension for the optimal solution; we have already asserted that by scaling, \( L_k^* \leq 1 \).

Similar to \cite{2,13}, we compare the actual assignment made by the algorithm (starting with zero load on every machine in every dimension) to a hypothetical assignment made by the optimal solution starting with the final algorithmic load on every machine (i.e., load of \( \Lambda_i(k) \) on machine \( i \) in dimension \( k \)). Using greediness of the algorithm and convexity of the potential function, we argue in Lemma 28 that the change in potential in the former process is upper bounded by that in the latter process.

**Lemma 28.** The total change in potential in the online algorithm satisfies

\[
\Phi(n) - \Phi(0) = \sum_{k=1}^{d} \alpha_k L_k^q \leq \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda^*_i(k) \right)^{r_k} \right)^{q_k/r_k} \right) - \sum_{k=1}^{d} \alpha_k L_k^q.
\]

**Proof.** We can express the resulting change in potential from scheduling job \( j \) as follows.

\[
\Phi(j) - \Phi(j-1) = \sum_{k=1}^{d} \alpha_k \left( L_k^q(j) - L_k^q(j-1) \right) = \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \Lambda^*_i(k) \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \Lambda_i(k) \right)^{q_k/r_k} \right)
\]

\[
= \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + p_{i,j}(k) \cdot y_{i,j} \right)^{r_k} \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \Lambda^*_i(k) \right)^{q_k/r_k} \right). \tag{13}
\]

Since the online algorithm schedules greedily based on \( \Phi(j) \), using optimal schedule’s assignment for job \( j \) must result in a potential increase that is at least as large. Therefore by \cite{13} we have

\[
\Phi(j) - \Phi(j-1) \leq \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + p_{i,j}(k) \cdot y_{i,j}^r \right)^{r_k} \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \Lambda^*_i(k) \right)^{q_k/r_k} \right). \tag{14}
\]

For the next two bounds, we will use the following fact:

**Fact 29.** The function \( f(x_1, x_2, \ldots, x_m) = (\sum_i x_i^z + a_i)^w - (\sum_i x_i^z)^z \) is non-decreasing if for all \( i \in [m] \) we restrict the domain of \( x_i \) to be \([0, \infty)\), \( w \geq 1, z \geq 1 \), and \( a_i \geq 0 \).

Fact 29 follows directly from the observation that all parameters are positive, the function is continuous in the domain, and its derivative is non-negative.

As loads are non-decreasing, \( \Lambda_i(k) \geq \Lambda_i(k-1) \). Also note that \( r_k \geq 1 \) and \( q_k/r_k = (r_k + \log d)/r_k > 1 \). Thus, we can apply Fact 29 to \cite{13} (setting \( w = r_k, z = q_k/r_k, a_i = p_{i,j}(k) \cdot y_{i,j}^r \)) to obtain

\[
\Phi(j) - \Phi(j-1) \leq \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + p_{i,j}(k) \cdot y_{i,j}^r \right)^{r_k} \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \Lambda^*_i(k) \right)^{q_k/r_k} \right). \tag{15}
\]

We can again use Fact 29 to further bound the potential increase (using the same values of \( a_i, w, \) and \( z \), but now \( \Delta x_i = \Lambda^*_i(k) \)):
\[ \Phi(j) - \Phi(j-1) \leq \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j-1}^s(k) + p_{i,j}(k) \cdot y_{ij}^s \right) \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j-1}^s(k) \right) \right)^{q_k/r_k} \right) \]

\[ = \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j}^s(k) \right) \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j-1}^s(k) \right) \right)^{q_k/r_k} \right). \tag{16} \]

Observe that for a fixed \( k \), the RHS of (16) is a telescoping series if we sum over all jobs \( j \):

\[ \sum_{j=1}^{n} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j}^s(k) \right) \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j-1}^s(k) \right) \right)^{q_k/r_k} \right) = \alpha_k \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j}^s(k) \right) \right)^{q_k/r_k} - \left( \sum_{i=1}^{m} \left( \Lambda_i(k) \right) \right)^{q_k/r_k} \tag{17} \]

We have \( \sum_{j=1}^{n} (\Phi(j) - \Phi(j-1)) = \Phi(n) - \Phi(0) \) since this is also a telescoping series. By definition, \( \Phi(0) = 0 \) and \( \Phi(n) = \sum_{k=1}^{d} \alpha_k L_k^{q_k} \). Using these facts along with (16) and (17) give us

\[ \sum_{k=1}^{d} \alpha_k L_k^{q_k} = \sum_{j=1}^{n} (\Phi(j) - \Phi(j-1)) \quad \text{(since \Phi telescopes, \Phi(0) = 0, and \Phi(n) = \sum_{k=1}^{d} \alpha_k L_k^{q_k})} \leq \sum_{k=1}^{d} \alpha_k \left( \sum_{i=1}^{m} \left( \Lambda_i(k) + \Lambda_{i,j}^s(k) \right) \right)^{q_k/r_k} - \sum_{k=1}^{d} \alpha_k L_k^{q_k} \quad \text{(by (16) and (17)).} \tag{18} \]

Next, we use Minkowski inequality (e.g., (33)), which states that for any two vectors \( v_1 \) and \( v_2 \), we have \( \|v_1 + v_2\|_r \leq \|v_1\|_r + \|v_2\|_r \). Applying this inequality to the RHS in Lemma 28 we obtain

\[ \sum_{k=1}^{d} \alpha_k L_k^{q_k} \leq \sum_{k=1}^{d} \alpha_k \left( \left( \sum_{i=1}^{m} \left( \Lambda_i^q(k) \right)^{1/r_k} + \left( \sum_{i=1}^{m} \left( \Lambda_i^q(k) \right) \right)^{1/r_k} \right)^{q_k/r_k} - \sum_{k=1}^{d} \alpha_k L_k^{q_k} \right) \]

\[ = \sum_{k=1}^{d} \alpha_k (L_k^{q_k} + L_k^{q_k}) - \sum_{k=1}^{d} \alpha_k L_k^{q_k} \tag{19} \]

The next lemma bounds the first term in the RHS of (19).

**Lemma 30.** For any \( k \), it holds that \( (L_k + L_k^*)^{q_k} \leq e^{1/2} L_k^{q_k} + (3q_k \cdot L_k^*)^{q_k} \).

**Proof.** We consider two cases. If \( L_k < 2q_k \cdot L_k^* \), it immediately implies that \( (L_k + L_k^*)^{q_k} \leq (3q_k \cdot L_k^*)^{q_k} \). On the other hand, if \( L_k \geq 2q_k \cdot L_k^* \), then we have

\[ (L_k + L_k^*)^{q_k} \leq (1 + 1/(2q_k))^{q_k} \cdot L_k^{q_k} \leq \left( e^{1/(2q_k)} \right)^{q_k} \cdot L_k^{q_k} = e^{1/2} L_k^{q_k}. \]

By rearranging (19) and applying Lemma 30 we have

\[ 2 \sum_{k=1}^{d} \alpha_k L_k^{q_k} \leq \sum_{k=1}^{d} \alpha_k (L_k + L_k^*)^{q_k} \leq e^{1/2} \sum_{k=1}^{d} \alpha_k L_k^{q_k} + \sum_{k=1}^{d} \alpha_k (3q_k \cdot L_k^*)^{q_k} = e^{1/2} \sum_{k=1}^{d} \alpha_k L_k^{q_k} + \sum_{k=1}^{d} (L_k^*)^{q_k}. \tag{20} \]

Note that the last equality is due to the fact that \( \alpha_k^{-1} = (3q_k)^{q_k} \). By initial scaling, \( L_k^* \leq 1 \) for all \( k \). Therefore, after rearranging (20), we obtain

\[ (2 - e^{1/2}) \sum_{k=1}^{d} \alpha_k L_k^{q_k} \leq \sum_{k=1}^{d} (L_k^*)^{q_k} \leq d. \]
which for any fixed $k$ implies

$$L_k \leq \frac{1}{(2 - e^{1/2})^{1/q_k}} \left( \frac{d}{\alpha_k} \right)^{1/q_k} \quad \text{(since $q_k \geq 1$ and $2 - e^{1/2} < 1$)}$$

$$\leq \frac{1}{2 - e^{1/2}} \left( \frac{d}{\alpha_k} \right)^{1/q_k} = \frac{3}{2 - e^{1/2}} \cdot \left( d^{1/q_k} \right) q_k < 10 \cdot d^{1/q_k} \cdot q_k = 20q_k = O(r_k + d).$$
References

[1] Susanne Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999.

[2] James Aspnes, Yossi Azar, Amos Fiat, Serge A. Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[3] Adi Avidor, Yossi Azar, and Jiri Sgall. Ancient and new algorithms for load balancing in the $l_p$ norm. *Algorithmica*, 29(3):422–441, 2001.

[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. In *Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996)*, pages 178–195, 1996.

[6] Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and F. Bruce Shepherd. Tight bounds for online vector bin packing. In *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 961–970, 2013.

[7] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, 1995.

[8] Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.

[9] Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.

[10] Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35(1):108–121, 2000.

[11] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[12] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing problems. In *ESA*, pages 689–701, 2005.

[13] Ioannis Caragiannis. Better bounds for online load balancing on unrelated machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 972–981, 2008.

[14] Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. Tight bounds for selfish and greedy load balancing. *Algorithmica*, 61(3):606–637, 2011.

[15] Ashok K. Chandra and C. K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM J. Comput.*, 4(3):249–263, 1975.

[16] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004.

[17] George Christodoulou, Vahab S. Mirrokni, and Anastasios Sidiropoulos. Convergence and approximation in potential games. *Theor. Comput. Sci.*, 438:13–27, 2012.
[18] R. A. Cody and Edward G. Coffman Jr. Record allocation for minimizing expected retrieval costs on drum-like storage devices. *J. ACM*, 23(1):103–115, 1976.

[19] Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta Cybern.*, 9(2):107–119, 1989.

[20] Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, pages 202–210, 2000.

[21] Todd Gormley, Nick Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA.*, pages 564–565, 2000.

[22] R. L. Graham. Bounds for certain multiprocessing anomalies. *Siam Journal on Applied Mathematics*, 1966.

[23] Magnús M. Halldórsson and Mario Szegedy. Lower bounds for on-line graph coloring. *Theor. Comput. Sci.*, 130(1):163–174, 1994.

[24] David G. Harris and Aravind Srinivasan. The moser-tardos framework with partial resampling. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 469–478, 2013.

[25] J. F. Rudin III. Improved bound for the on-line scheduling problem. *PhD thesis, The University of Texas at Dallas*, 2001.

[26] David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *J. Algorithms*, 20(2):400–430, 1996.

[27] Adam Meyerson, Alan Roytman, and Brian Tagiku. Online multidimensional load balancing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 287–302, 2013.

[28] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[29] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis*, pages 15–1, 2004.

[30] Jiri Sgall. On-line scheduling. In *Online Algorithms*, pages 196–231, 1996.

[31] Jiri Sgall. Online scheduling. In *Algorithms for Optimization with Incomplete Information, 16.-21. January 2005*, 2005.

[32] Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Selfish load balancing and atomic congestion games. *Algorithmica*, 47(1):79–96, 2007.

[33] Wikipedia. Minkowski inequality — Wikipedia, the free encyclopedia.
A Concentration Inequalities

Theorem 31. (Chernoff Bounds [28]) Let $X_1, X_2, ..., X_n$ be independent binary random variables and let $a_1, a_2, ..., a_n$ be coefficients in $[0, 1]$. Let $X = \sum_i a_i X_i$. Then,

- For any $\mu \geq E[X]$ and any $\delta > 0$, $\Pr[X > (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$.

- For any $\mu \leq E[X]$ and $\delta > 0$, $\Pr[X < (1 - \delta)\mu] \leq e^{-\mu \delta^2/2}$.