Online Scheduling via Learned Weights

Silvio Lattanzi[*] Thomas Lavastida[†] Benjamin Moseley[†] Sergei Vassilvitskii[*]

Abstract

Online algorithms are a hallmark of worst case optimization under uncertainty. On the other hand, in practice, the input is often far from worst case, and has some predictable characteristics. A recent line of work has shown how to use machine learned predictions to circumvent strong lower bounds on competitive ratios in classic online problems such as ski rental and caching. We study how predictive techniques can be used to break through worst case barriers in online scheduling.

The makespan minimization problem with restricted assignments is a classic problem in online scheduling theory. Worst case analysis of this problem gives $\Omega(\log m)$ lower bounds on the competitive ratio in the online setting. We identify a robust quantity that can be predicted and then used to guide online algorithms to achieve better performance. Our predictions are compact in size, having dimension linear in the number of machines, and can be learned using standard off the shelf methods. The performance guarantees of our algorithms depend on the accuracy of the predictions, given predictions with error η, we show how to construct $O(\log \eta)$ competitive fractional assignments.

We then give an online algorithm that rounds any fractional assignment into an integral schedule. Our algorithm is $O((\log\log m)^3)$-competitive and we give a nearly matching $\tilde{\Omega}(\log\log m)$ lower bound for online rounding algorithms.[1] Altogether, we give algorithms that, equipped with predictions with error η, achieve $O(\log \eta\ (\log\log m)^3)$ competitive ratios, breaking the $\Omega(\log m)$ lower bound even for moderately accurate predictions.

1 Introduction

Modern machine learning has had unprecedented success in speech and language understanding, vision, and perception. More recently, researchers have shown how to use machine learning to solve classical combinatorial optimization questions, for example finding the optimal decision strategy for the secretary problem [16], computing Optimal Auctions [12], or building faster look up tables [17].

The above approaches achieve notable empirical success, but come without any theoretical analysis. A complementary line of work has looked into ways to incorporate machine learned predictions to provide formal guarantees, for instance improving competitive ratios for online algorithms [20, 26]. These methods bound the performance of the algorithm in terms of the (ex-post) error of the predictor, and show that, with good, but *imperfect*, predictions, one can circumvent strong worst case lower bounds.

In this work we continue this line of research and extend it to one of the central scheduling problems: minimizing makespan under restricted assignment. In this problem there are m machines, and jobs arrive one at a time, each annotated with its size and the subset of machines it can run on. The goal is to allocate jobs to machines online, to minimize the *makespan*, i.e. the maximum total size of the jobs on any machine. This is a key problem in scheduling, but has strong, $\Omega(\log m)$, lower bounds on the competitive ratio.

Online algorithms operate under worst case assumptions about the input, but, in practice, the input often has a repetitive nature to it. As an example, when scheduling jobs in a large data center, we may observe that some jobs happen daily around the same time (for instance, regular backup jobs), and so their presence is very predictable. Other jobs, for instance, payroll analysis jobs, may have some variability in terms of arrival time, but are known to happen weekly, or at the end of each month; overall traffic may be lighter on holidays, or during extreme weather events, and so on. Thus, it makes sense that we can predict something about the jobs and the congestion of machines using standard features, such as day of week, weekday vs. holiday, weather, etc.

While the predictions are not going to be perfect, they can still guide the allocation algorithm about which machines will be in contention in the future. The nature of the prediction then becomes part of the algorithm design process. There is a fine balance between offloading

[*]Google Research. Email: {silviol, sergeiv}@google.com

[†]Carnegie Mellon University. Email: {tlavasti, moseleyb} @andrew.cmu.edu. T. Lavastida and B. Moseley were supported in part by a NSF Grants CCF-1830711, CCF-1733873, CCF-1733873 and CCF-1845146, a Google Research Award, a Bosch junior faculty chair and an Infor faculty award.

[1]We use $\tilde{O}(f(m))$ and $\tilde{\Omega}(f(m))$ notations to suppress factors of $\log^c f(m)$ for constant c.

too much of the complexity onto the predictor on one side, versus having the prediction not provide any insights on the other. The former makes the algorithmic problem trivial, but the learning problem virtually impossible, the latter gives little additional benefit over worst case analysis. Thus, we strive for sparse representations that capture the complexity of the problem. For instance, for online bipartite matching, a problem closely related to our work (except for the choice of the objective function), previous work [11, 29] has shown that the dual variables associated with machines in the LP formulation of the problem are enough to reconstruct the optimum assignment. As we will show in Section 2.3 this representation is not sufficient when minimizing makespan, nor are other immediate quantities, such as the total load of each machine in the optimum solution, or the number of jobs that could potentially be assigned.

Our first contribution is in identifying a specific quantity that we will predict. We follow the work of [1] and associate a *weight*, w_i with each machine i. We show that even if we are only given access to estimates, $\hat{w}_i$ of the optimal w_i, we can recover a near optimal fractional solution, with the approximation guarantee that scales logarithmically with the error.

Theorem 1.1. (Theorem 3.1 restated) *Let w be a set of machine weights that lead to a fractional solution with makespan T. Let $\hat{w}$ be predictions of w and let $\eta = \max_i \hat{w}_i/w_i$ be the maximum error in our predictions. Then there exists an online algorithm that generates a fractional assignment of jobs to machines with makespan at most $O(T \log \eta)$.*

The learned weights allow us to recover an approximately optimal *fractional* solution; however, new algorithmic challenges arise when rounding this solution online to an *integral* solution. Our second technical contribution is an online rounding procedure that loses an $O((\log \log m)^3)$ factor in the makespan. Thus converting from fractional to integer solutions is not as hard as obtaining good fractional solutions in the first place. We note that our rounding procedure can be used for the more general unrelated machines problem, where job sizes are machine dependent and uncorrelated.

Rounding a fractional solution online requires several new ideas. Prior rounding algorithms need access to the overall instance, and are inherently offline. For instance, the 2-approximation of Lenstra, Shmoys and Tardos [19] requires preprocessing the fractional solution, iteratively transforming the assignment by shifting probability mass found in cycles in the corresponding bipartite graph. Obviously the full problem instance and assignment is necessary to perform this kind of rebalancing. Other methods utilize sophisticated configuration LPs [15, 28], which cannot be easily modified to fit the online setting. Thus known makespan rounding methods are not good candidates to adapt to the online setting.

In an effort to design a good rounding algorithm, we first observe that any deterministic rounding procedure has a competitive ratio of $\Omega(\log m)$ as compared to the fractional solution (See Appendix 6.1). We resort to randomized approaches. Since simple randomized rounding will lead to a poor approximation ratio, we introduce a new rounding algorithm that involves carefully transforming the instance to ensure certain structural properties, then classifying jobs into different categories, and running different rounding algorithms for each category. The classification is critical, as rounding algorithms that are good for one category perform poorly for others. However, coupled together in the right way, we establish our results. See Section 4 for a proof overview and Section 5 for the whole proof.

Theorem 1.2. (Theorem 5.1 restated) *Let x be a fractional assignment of restricted assignment jobs that is revealed online and let T be the fractional makespan of x.*[2] *There exists a randomized online algorithm that rounds a fractional assignment to an integer assignment such that the resulting makespan is at most $O((\log \log m)^3 T)$ with high probability.*

In addition to this, we show that our online rounding algorithm is *nearly optimal*. We show that no randomized online rounding algorithm can achieve a competitive ratio better than $\tilde{\Omega}(\log \log m)$ when rounding a given fractional solution online (See Section 6.2).

Theorem 1.3. (Theorem 6.1 restated) *Let x be a fractional assignment of restricted assignment jobs that is received online and let T be the fractional makespan. No deterministic algorithm for converting x to an integer assignment can be $o(\log / \log \log mm)$-competitive with respect to T. Further, no randomized algorithm for the same task can be $o(\log \log m / \log \log \log m)$-competitive with respect to T.*

Combining the two upper bound results, we show that by learning approximate weights and applying our rounding algorithm one obtains an $O(\log \eta (\log \log m)^3)$-competitive algorithm, an exponential improvement over an algorithm that does not use any predictions, whenever the predictions are reasonably accurate. We also show that it's easy to fall back to the traditional algorithm in case when the error is detected to be large. More formally:

[2]We assume that T is at least the size of any job to deal with instances having poor integrality gap. See Section 5 for a proper definition of T

Corollary 1.1. *For any restricted assignment job sequence there exist machine weights w and an online algorithm that when given access to predictions $\hat{w}$ of the weights, assigns the jobs with competitive ratio $O(\min\{\log \eta (\log\log m)^3, \log m\})$ with respect to the makespan, where $\eta = \max_i \hat{w}_i / w_i$. The online algorithm is randomized and succeeds with high probability.*

1.1 Related Work

There are multiple instances of augmenting classic online algorithms with additional information in order to improve competitive ratios. Broadly, the extra information can come in the form of assumptions on the data, or in terms of explicit hints about the future given to the algorithm.

The most prominent example of the former is the random arrival model, where the full set of arrivals is chosen adversarially, but the exact sequence seen by the algorithm is a random permutation of the worst case instance. These lead to a family of algorithms that use the first part of the input to learn the structure of the input, often via simple empirical risk minimization (ERM) methods, and then use the learned structure to guide the algorithm's choices on the rest of the input. Notable examples include the secretary problem, where the learned structure is just the best candidate seen so far, to more delicate arguments, such as those in the work by Devanur and Hayes [11] and Vee et al. [29] for online bipartite matching. A line of work initiated by Cole and Roughgarden [10] and continued by Balkanski et al. [5] asks explicitly what kind of information can be learned just from a sample of the data, and strives to get tight bounds on the size of the sample necessary to achieve good results.

A related structural assumption is that input comes from a stochastic distribution. This too has been studied in the context of online matching [23] and bandit learning [8], where the authors show how to get improved guarantees if the assumption holds, and retain the worst case guarantees if it does not. Mahdian et al. [21] generalize this even further, allowing for an arbitrary *optimistic* algorithm, rather than assuming that the input is stochastic, and showing how to recover a constant fraction of either the optimistic or worst case performance.

In a new line of work, Kumar et al. [18] assume that the online input is a mixture of adversarial elements and elements arriving from a known sequence. They show how to achieve near optimal results for the online matching in this "semi-online" model of computation.

The work described above assumes that the input is restricted in some manner. Additionally, there has been increased interest in a model, where the input is not explicitly restricted, but some information about it is available to the algorithm. The seminal work by Ailon et al. [2] considered "self-improving" algorithms that effectively learn the input distribution, and adapt to be nearly optimal in that domain.

More recently, Lykouris and Vassilvitskii [20] formulated the Online with ML advice model, and showed how to improve the competitive ratio of caching algorithms given a prediction of the subsequent arrival time of an element when it appears. Purohit et al. [26] extended this work to the classical "ski rent or buy" problem, where the algorithm is given an estimate on the number of skiing days, and to non clairvoyant scheduling, where the prediction is on the length of each job. In a slightly different setting, Medina and Vassilvitskii [22] use predictions about bidders' valuations to set good reserve prices in an auction, and Hsu et al. [14] use predictions on the expected frequency of an element to improve the space complexity of heavy hitters sketches.

These kinds of predictions have also been used to improve classical data structures, Kraska et al. [17] show how to speed up indices using neural networks, while Mitzenmacher [24] shows how to reduce the space complexity of Bloom Filters given a prediction of set membership.

Finally, in the online algorithms with advice model, the algorithm has access to an oracle that knows the exact input. The goal then is to reduce the amount of information the oracle needs to communicate to the algorithm, see Boyar et al. [7] for a recent survey. Critically, in this model the oracle is perfect, and makes no mistakes, whereas we explicitly assume that the predictions are going to be error-prone, and our goal is to tie the competitive ratio of the algorithm to the quality of the predictions.

The other closely related area of work is that of approximation and online algorithms for scheduling. There has been a considerable amount of work on approximate makespan minimization. The work of Lenstra, Shmoys and Tardos [19] gives a 2-approximation algorithm by rounding a natural LP relaxation and this result holds for the unrelated machines case. The breakthrough work of [15, 28] improves the approximation ratio for the restricted assignment problem. The work of Svensson [28] shows a 1.9412 approximation and Jansen and Rohwedder [15] improves this to a $2 - \frac{1}{6} + \epsilon$ approximation. This additionally bounds the integrality gap of the configuration LP by $2 - \frac{1}{6}$. For the unrelated machine case, the best known approximation is 2. Recently, Chakrabarty, Khanna and Li [9] show improved results for a special case. In the online setting there are tight bounds of $\Theta(\log m)$ on the competitive ratio for the restricted assignment problem [4].

2 Preliminaries

2.1 Problem Definition and Notation We study the online restricted assignment problem. In this problem there are m machines (indexed from 1 to m) and a sequence of jobs that arrive online. Job j has an integer size $p_j > 0$ and a subset of machines $\emptyset \subsetneq N(j) \subseteq [m]$ where j can be feasibly assigned. Throughout this paper we refer to $N(j)$ as the *neighborhood* of job j. Similarly, we can define the neighborhood of a machine i, $N(i)$ as the set of jobs that can feasibly be assigned to i. The neighborhood structure induces a bipartite graph on the set of machines and the set of jobs. The algorithm must irrevocably assign each job j to some machine $i \in N(j)$, and it must do so online, i.e. with no knowledge of the future jobs in the sequence. This assignment incurs a load on each machine, equal to the total size of the jobs assigned to each machine. The objective is to minimize the makespan, or maximum load across all machines. If J_i is the set of jobs assigned to machine i, then $L_i = \sum_{j \in J_i} p_j$ and the makespan, $T = \max_{i \in [m]} L_i$.

A more general version of this problem is the online makespan minimization on unrelated machines problem. The setup for this problem is mostly the same as above, except that instead of jobs having a single size and a subset of feasible machines, the size of a job is completely machine dependent and uncorrelated between machines. That is, for each machine-job pair i, j, there is a size $p_{ij} > 0$ that determines the size of job j if it were assigned to machine i. The objective is still to minimize the makespan. The restricted assignment problem can be reduced to unrelated machines by taking $p_{ij} = p_j$ if $i \in N(j)$ and $p_{ij} = \infty$ otherwise.

We study our algorithms in the setting of competitive analysis. If T is the optimal makespan for a sequence of jobs in hindsight, then we seek to give online algorithms that output assignments with makespan at most αT, for $\alpha \geq 1$ as small as possible. If an online algorithm achieves such a guarantee, then we say that the algorithm is α-competitive. For the case of the online rounding problem we define competitiveness similarly, but instead with $T = \max\{\max_{i \in [m]} \sum_{j \in N(i)} p_j x_{ij}, \max_j p_j\}$. In the case of randomized algorithms, we compare our online algorithm to oblivious adversaries. The adversary does not have access to the randomness used by our algorithm to make assignments. That is, the adversary fixes the (worst case) sequence of jobs in the beginning, then our algorithm runs and assigns the jobs.

In analyzing randomized algorithms we will often say that some event occurs with high probability. We take this to mean that the event occurs with probability at least $1 - 1/m^c$, for any constant $c > 0$. This implies that we can union bound over any polynomially in m many "bad" events and retain high probability. We will use concentration inequalities to establish high probability bounds several times in our analysis, see Section A for the exact statements of the bounds we use.

Technical Assumptions. Throughout the rest of this paper, we assume that our algorithms know the optimal makespan T. In the offline setting this assumption can typically be removed by a simple binary search. In Appendix D, we show how to remove this assumption in the online setting via a standard doubling argument. We also assume that the job sizes are at least 1 and are polynomially bounded, i.e. $p_j = O(m^k)$ for some constant k. This assumption can be made with negligible increase in the competitive ratio.

2.2 ML Oracles When incorporating predictions into an online algorithm, an important consideration is deciding what to predict. At a high level, the predictions should give a parsimonious representation of the problem instance. Specifically, we want the prediction to satisfy three properties:

- The performance of the algorithm should degrade gracefully with the error in predicted quantities.
- The predictions should be robust to inconsequential changes in the problem instance.
- The predictions should be efficiently learnable, that is they should be concise and come from a limited domain.

The first point is critical to good algorithm design. Machine learned approaches are never perfect, and errors are to be expected, and any algorithm that is not robust to errors in the predictions is bound to perform poorly in practice. In their definition of the model, Lykouris and Vassilvitskii [20] further insist that algorithms are *consistent*, that is they recover the optimal solution as the prediction error goes to zero.

The next two properties describe what it means for the learning to be meaningful. For instance, some quantities may be easy to predict, but give no insight into the structure of the problem instance. For instance, predicting the total load on each machine in the optimal solution is *not* a good prediction—one can easily extend any example to make sure that the total load on each machine is identical. We give other examples of poor options for predictions, and further describe the qualities of a meaningful prediction in Section 2.3.

In contrast, the last point puts limits on the nature of the predictions, making sure they can be efficiently learned. For instance, one may propose predicting the exact instance that will appear and then execute the offline algorithm on the learned prediction. The field of computational learning theory has developed strong bounds on the number of examples needed to effectively learn a function from a given class. Specifically, it is well known that if the hypothesis class contains N functions, one needs at least $\log N$ examples to learn the best function (this is the weakest lower bound, typically many more examples are necessary) [25]. In the case of restricted assignment, each of the jobs can be matched to 2^m machines, therefore an instance with n jobs arriving, can take on 2^{mn} different values. Thus learning the jobs requires at least $\Omega(mn)$ examples, which is prohibitively expensive even for relatively small m and n. On the other hand, learning a single parameter per machine is a much easier learning task, requiring many fewer examples.

In much of the previous work the choice of the prediction was relatively straightforward. For instance for the classic ski rental problem, Purohit et al. [26] predict the number of skiing days, and then base their decision on the value of the prediction vis-à-vis the cost of buying the skis. In streaming heavy hitters, Hsu et al. [14] predict whether an element is likely to be a heavy hitter, and if so, maintain its count exactly, rather than resorting to a sketch. Finally, for the online caching problem, Lykouris and Vassilvitskii [20] focus on predicting the subsequent arrival time of each element, and then modify the Marking algorithm to take advantage of this new information. In contrast, in online scheduling, the question of what to predict is not obvious.

2.3 Predictions for Online Scheduling The decision of what to predict obviously influences the design of the algorithm using these predictions. However, even without an algorithm in mind, we can eliminate some choices because they fail to satisfy one of the criteria listed above.

For the online scheduling problem, the quantity we predict should intuitively guide us how congested a particular machine is going to be. Consider a restricted version of the problem, where the optimal makespan has value one. Then each instance is equivalent to a bipartite graph between jobs and machines, and the offline problem is to compute a matching between jobs and machines. Given the full instance, what is a good representation that can be used to guide the online algorithm?

One natural approach is to look at the degree of each machine, i.e. the number of jobs that *could* be assigned to it. However, by adding a small number of dummy jobs and machines it is easy to modify each instance so that all machines have identical degree, in which case, this prediction does not carry any additional information. A similar fate befalls predictions of the load of each machine in the optimal solution, the degree of the jobs, and other simple heuristics: for each of these there exists a simple transformation that makes this additional information vacuous.

A more robust approach is to look at the dual problem, and consider learning the dual variables corresponding to machines. This kind of a setting has been successfully used for online bipartite matching—Devanur and Hayes [11] showed how to use duals learned on a random sample of the input to give approximately optimal solutions in an online setting. Critically, however, the choice of the objective plays a large role: while using $1+\epsilon$ approximate duals gives a $(1-O(\epsilon))$ solution to the bipartite matching, the same approach does not yield a constant competitive approximation to the makespan. The reason is that in job scheduling, we must match *all* of the jobs to machines and compare the resulting makespans, whereas in online matching, we only try to match as many jobs as possible to empty machines and compare the cardinality of the matching.

Another approach for online matching, advocated by Vee et al. [29], was to formulate the online matching problem as a quadratic program, and then look at the dual variables in that space. In addition to the mismatch in objective described above, we note that the duals in the Vee et al. formulation are extremely sensitive, and approximately correct duals may no longer lead to near optimal solutions on the primal.

3 A Robust Online Algorithm via Machine Weights

In this section we identify a quantity that compactly captures the structure of an offline instance of the problem. We give an online algorithm to compute fractional solutions using these quantities, and show that they are robust to errors, if we were to predict them.

The key idea is to assign a *weight* to each machine, and then allocate each job proportionally to the weights of the machines that it can be assigned to. Intuitively, the machine weight is inversely proportional to the contentiousness of the machine, i.e. machines with very high demand have small weights.

Formally, let $w \in \mathbb{R}^m_+$ be a vector of non-negative weights, one per machine. Let $x_{ij}(w)$ denote the fractional assignment of job j on machine i when using weights w, we define the assignment function as:

$$x_{ij}(w) = \frac{w_i}{\sum_{i' \in N(j)} w_{i'}} \tag{3.1}$$

We first need to show that the weights capture enough information for us to reconstruct a good solution. In other words, we need to ensure that for any offline instance there are a set of weights such that the corresponding fractional assignment has a near optimal makespan. We build upon the work of Agrawal et al [1] who showed the existence of such weights for b-matchings. Formally, we say that a set of weights w is c-**good** if for every machine i, $\sum_{j\in N(i)} p_j x_{ij}(w) \le cT$ for some constant $c \ge 1$. Good weights exist for arbitrarily small c for the restricted assignment problem and the proof is omitted to a full version of the paper.

Given that weights are enough to reconstruct approximately optimal solutions, they will be our target for predictions. Before analyzing what happens when weights are predicted incorrectly, we observe that the allocation given by w is *scale invariant*, i.e. that γw yields the same allocation as w for any $\gamma \in \mathbb{R}$.

Remark 3.1. *The fractional assignment produced by w is scale invariant, i.e. for any $\gamma \in \mathbb{R}$, $x_{ij}(w) = x_{ij}(\gamma w)$.*

3.1 Constructing Fractional Solutions Online Using Learned Weights Suppose that $\hat{w}$ is a prediction of good weights w. Due to scale invariance, we can assume that $\hat{w}_i \ge w_i$ for all i. First we consider using $\hat{w}$ directly to compute allocations online using Equation 3.1. To analyze this procedure we define $\mu_i = \hat{w}_i/w_i \ge 1$ to be the relative error with respect to the i'th machine. We define the total error in the prediction to be $\eta = \max_i \mu_i$. Consider the allocation $x_{ij}(\hat{w})$ given by the predictions. Intuitively, this allocation is locally a good approximation to $x_{ij}(w)$, which implies that the makespan of our complete solution is bounded. The following claim makes this intuition precise and implies that the naive algorithm will have a makespan of $O(\eta T)$.

Claim 3.2. *For all i and j we have that $x_{ij}(\hat{w}) \le \eta x_{ij}(w)$.*

Proof. Using equation (3.1) we have:

$$x_{ij}(\hat{w}) = \frac{\hat{w}_i}{\sum_{i'\in N(j)} \hat{w}_{i'}} = \frac{\mu_i w_i}{\sum_{i'\in N(j)} \mu_{i'} w_{i'}} \le \eta \left(\frac{w_i}{\sum_{i'\in N(j)} w_{i'}} \right) = \eta x_{ij}(w).$$

□

Thus we see that the error in our prediction cleanly shows up in the competitive ratio of our algorithm. However we can use standard techniques from online algorithms to improve on this result exponentially. The key idea is that we do not have to continue using the current predictions $\hat{w}$ if we believe the error is large. Rather than use the predictions statically, we update them over time to account for errors that have been detected. The following observation is important in formalizing this idea. If for all i $1/2 \le \hat{w}_i/w_i \le 1$, then $x_{ij}(\hat{w}) \le 2x_{ij}(w)$. This follows by applying the same style of analysis as above. This motivates the design of Algorithm 1.

Algorithm 1 Improved algorithm for computing fractional assignments online

```
Let $\hat{w}$ be predictions of $w$
Initialize $L_i \leftarrow 0$ for each machine $i$ ▷ $L_i$ = fractional load of machine $i$
for each job $j$ do
    For all $i \in N(j)$, $L_i \leftarrow L_i + p_j x_{ij}(\hat{w})$ ▷ Compute fractional assignment
    for $i = 1, \dots, m$ do
        if $L_i > 2T$ then
            $\hat{w}_i \leftarrow \hat{w}_i/2$, $L_i \leftarrow 0$ ▷ Update $\hat{w}$, start new phase
```

Algorithm 1 keeps track of the load of each machine in phases. At the start of a machine's phase its load L_i is initialized to 0. As each job j arrives we update L_i by adding $x_{ij}(\hat{w})$. At this point if $L_i \le 2T$ we do nothing, as we have no reason to believe that $\mu_i = \hat{w}_i/w_i$ is very large. However, if $L_i > 2T$, then by our observation we know that μ_i is large, so we update $\hat{w}_i$ by dividing it by 2 and start a new phase by resetting L_i to 0. Once the condition in our observation is satisfied, we know that this will be the last phase for all machines. So the question becomes, how many phases will each machine go through until the condition is satisfied. Let k_i be the number of phases that machine i starts. The condition is satisfied for machine i when $1/2 \le \frac{\hat{w}}{2^{k_i} w_i} \le 1$ which implies that $k_i \le \lceil \log_2(\hat{w}_i/w_i) \rceil + 1$. All machines satisfy the condition after $k = \max_i k_i = \max_i \log_2(\hat{w}_i/w_i) = \max_i \log_2(\mu_i)$ phases.

How much does this algorithm lose in terms of the makespan? Each machine phase incurs a factor of 2 loss in the makespan, and each machine has at most $k = \max_i \log_2(\mu_i)$ phases. Thus we see that the resulting makespan is $O(kT) = O(\max_i \log_2(\mu_i)T)$. Since $\mu_i \ge 1$, we have that $\max_i \log_2(\mu_i) = \log_2(\max_i \mu_i) = \log_2(\eta)$. Thus the competitive ratio is $O(\log_2(\eta))$, an exponential improvement over naively using the predictions. We state the above results in Theorem 3.1.

Theorem 3.1. *Let $\hat{w}$ be predictions of a set of good machine weights w and let $\eta = \max_i \hat{w}_i/w_i$ be the maximum error in our predictions. Then Algorithm 1 is an $O(\min\{\log \eta, \log m\})$-competitive algorithm for minimizing the fractional makespan online.*

In order to get the stated competitive ratio of $O(\min\{\log \eta, \log m\})$, we run Algorithm 1 normally until assigning some job causes our algorithm to have makespan $> 2\log mT$. In this case, the predictions are not helpful and we switch to a $O(\log m)$-competitive algorithm from the literature, such as those in [3, 4].

3.2 Learning the Weights Thus far we have shown that individual machine weights are a good candidate for a learnable summary for the restricted assignment problem: they can be used to reconstruct a near-optimal fractional solution, and the assignment is robust to errors. Here we give a brief insight into the learning problem itself.

As in all machine learning scenarios, we assume that each machine is annotated with a set of features representing its characteristics. The features may capture hardware specifications of the machine (e.g. its memory, number of cores, processing speed, etc.), its location (e.g. the name of the datacenter), software considerations (e.g. operating systems and packages are installed), and so on. For a machine $i \in [m]$, let f_i represent the vector of features associated with the specific machine. Next we define the labels. Given an offline instance of the restricted assignment problem (set of machines, jobs, and the assignment graph), we compute good weights w for every machine.

Finally, we set up the learning task. If F is the set of all possible features, the learning task is to find a function $h : F \to \mathbb{R}^+$ from features to weights. The exact choice of h is up to the practitioner. For instance, when using linear regression, the learning task is finding a vector θ such that $h_\theta(f) = \sum_{i\in[m]}(f \cdot \theta - w_i)^2$ is minimized. In Appendix B we give other ways to learn the weights.

The question of how well one can learn the weights is at its heart an empirical question. As always, by increasing the set of features, and using richer hypothesis classes from which to draw the best h will lead to lower errors. We do highlight that our allocation algorithm also has some natural properties. For instance, any two machines that are identical (that have the same features), will always have the same weight and thus the same fractional assignment for a job feasible for both of them. Similarly, any two jobs that have the same neighborhoods will have the same fractional assignment as well.

4 Rounding Algorithm Overview

Before delving into the technical details, we first describe an overview of the rounding algorithm. Recall that jobs arrive online and when job j arrives the algorithm learns x_{ij} for all machines i. We assume that $\sum_{i\in[m]} x_{ij} = 1$ and the goal is to be competitive with the final fractional makespan. $T := \max_{i\in[m]} \sum_{j\in[n]} p_{ij}x_{ij}$. To make the exposition simpler, we discuss the case of restricted assignment with unit sized jobs and the algorithm knows the exact fractional makespan T a priori. In this case, each job has size 1 or ∞ on each machine.

First observe that if $T \geq \Omega(\log m)$ then the rounding is easy. Each job j independently performs randomized rounding, selecting a machine i with probability x_{ij}. Because the contribution of each job is much smaller than the total makespan, standard concentration bounds ensure that the makespan is bounded by $O(T)$ with high probability. The challenging case is when T is small compared to $\log m$. In the proof, we denote this the “large” job case.

We further break up the analysis into two cases. Let $B_j = \{i \mid x_{ij} \geq \frac{1}{\log^2 m}\}$ and $S_j = \{i \mid x_{ij} < \frac{1}{\log^2 m}$ and $x_{ij} > 0\}$. The set B_j contains the machines where x_{ij} is big and S_j are the machines in the support where x_{ij} is small. Let $\mathcal{B}$ be the set of jobs j where $\sum_{i\in B_j} x_{ij} \geq \frac{1}{2}$. These are jobs mostly assigned using large x_{ij} values. Let $\mathcal{S}$ be the remaining jobs. These are jobs assigned using mostly small x_{ij} values.

Jobs in $\mathcal{B}$. We begin by simplifying the instance. We show that at the cost of losing a factor of $O(\log\log m)$ in the total cost, we can transform the instance to one with $x_{ij} \in \{0, \frac{1}{\lambda}\}$ for a single value $\frac{1}{\log^2 m} \leq \frac{1}{\lambda} \leq 1$. This further implies that each job j has $\Theta(\lambda)$ machines in the support of x_{ij}. Let $N(j)$ be this set of machines. Notice that each machine can have at most $\tilde{O}(T\lambda) = O(\operatorname{poly}\log m)$ jobs that can be assigned to it. This case is hard because the fractional solution is revealing very little information. Indeed, each job is uniformly split across a neighborhood of size λ.

To reason about the rounding in this case, consider the bipartite graph G corresponding to the problem instance. Nodes representing jobs are on one side, and machines are on the other, with an edge between a job and machine if the (job, machine) pair is in the support. By construction the maximum degree in this graph, $\Delta = O(\operatorname{poly}(\log m))$. Now to allocate jobs we use the following algorithm: when job j arrives, it selects machines independently at random from $N(j)$, selecting machine i with probability $\Theta(\log\log(m) \cdot x_{ij})$[3]. The job can assign itself to any machine chosen so long as the machine has been assigned at most $O(T \log\log m)$ jobs so far. If the job does not select a machine or the machines are overloaded then the job “fails” and enters a set F of failed jobs.

Let G_F be the induced subgraph consisting only

[3]Note that machines are chosen independently and this independence is crucial for the proof. A job may select more than one machine or no machines. It easily follows that the probability a job fails to chose a machine is bounded by $\exp(\Theta(-\log\log m))$

of the failed jobs and all of the machines. The key to the proof is showing that with high probability *every* connected component in G_F is small. In particular, each connected component has fewer than $\operatorname{poly}\log m$ nodes. Intuitively, this is because the graph G has maximum degree at most $\operatorname{poly}\log m$ and therefore the graph is broken into small pieces. This is reminiscent of the shattering idea in the parallel graph algorithms community [6, 13].

If this is the case, and the components are small then the problem becomes easy. Each failed job assigns itself greedily to the least loaded machine. It is known [4] that the greedy deterministic algorithm is a $O(\log m')$-approximation for any input with m' machines. We can think of each component as an individual instance, resulting in $m' \leq \operatorname{poly}\log m$ and these jobs contribute at most a $O(T \log\log m)$ amount to the makespan.

Finally, we argue that the components are indeed small. Notice that if there is a connected component of size $\operatorname{poly}\log m$ then there should exist a path in the graph of length at least $\frac{\operatorname{poly}\log m}{\lambda}$ because the maximum degree is λ. Thus, it suffices to show that no such path survives. The proof begins by establishing that each job fails with probability at most $\frac{1}{\log^c m}$ for some constant c by simple concentration bounds. This means that every edge in G remains in G_F with probability at most $\frac{1}{\log^c m}$.

For sake of intuition assume that each edge were to be removed *independently* with this probability (this is not true and we will remove this assumption shortly). The probability a fixed path of length ℓ survives is at most $(\frac{1}{\log^c m})^\ell = \frac{1}{\log^{c\ell} m}$. Hence we can union bound over all possible paths to show that no long paths survive. More precisely, assume that the path starts at a machine node in G_F and there are m starting positions. Recall that the maximum degree is Δ, thus the total number of paths of length ℓ is bounded by $m\Delta^\ell$. Ensuring that $\Delta \leq \log^3 m$ and choosing $c \geq 4$ and $\ell \geq \log m$ ensures no path exists with good probability. This implies there is no large connected component, completing the analysis of jobs in $\mathcal{B}$.

The only issue that remains is the assumption on the independence of the edges. The proof establishes that edges or nodes sufficiently far apart survive to be in G_F independently. By carefully counting 'special' sets of edges that survive independently because they are well separated, but still reachable in a few hops, allows us to effectively use the above argument.

Jobs in $\mathcal{S}$. When we rely on machines with small assignment, we will run randomized rounding in phases. In phase k, each job j that makes it to the phase selects a machine with probability x_{ij}. If the chosen machine's makespan is smaller than $O(T)$ from jobs assigned during phase i then the job goes to the machine. If not, then the job goes to phase $k+1$.

Define the fractional makespan of phase k to be the maximum fractional makespan on the machines only counting jobs that survive to phase k. Using concentration bounds, we can show that the fractional makespan decreases rapidly with each phase. Intuitively, this is because most jobs have a good probability of being successfully assigned. After $O(\log\log m)$ number of phases, the fractional makespan will drop below $O(\frac{1}{\log^2 m})$. This is the last phase. At this point, if a job still survives then the job chooses $\log m$ machines in $N(j)$ uniformly at random. Then the job goes to a machine that no other job from this phase selected. Because the fractional makespan is so small, concentration bounds will imply that with high probability one of the machines that each job picks with be chosen only by that job. Thus, the overall the makepsan will be $O((\log\log m)T)$ from rounding jobs in $\mathcal{S}$ with high probability.

5 Online Rounding Algorithm & Analysis

In this section we give a formal analysis of the online rounding algorithm. For this section we assume the more general unrelated machine problem. Recall the setup of the problem. Jobs arrive over time online. When each job j arrives, the value of the fractional assignment, x_{ij}, and the job size p_{ij} is revealed for all machines i. That is, at each time t we know the fractional assignment x_{ij} for all jobs j that have arrived up to time t and have no information about the future jobs. We assume that $\sum_i x_{ij} = 1$. That is, all jobs are fully fractionally assigned.

The goal is to assign jobs online to machines integrally using the fractional values as a guide so that the final makespan is as close as possible to the makespan of the fractional schedule. Since the integrality gap can be bad for the underlying linear relaxation[4], we define the quantity $T := \max\{\max_{i\in[m]} \sum_{j\in[n]} p_{ij}x_{ij}, p^*\}$, where $p^* = \max\{p_{ij} \mid x_{ij} > 0\}$. Note that this assumption enforces $p_{ij} \leq T$ whenever $x_{ij} > 0$. It is known that the integrality gap is large if this condition is not met [30]. Since we apply the result of this section to the case of restricted assignment, we note that the definition of T above reduces to $T = \max\{\max_{i\in[m]} \sum_{j\in N(i)} p_j x_{ij}, \max_j p_j\}$ for this case.

An interesting challenge in our setting is that the assignment needs to be online so the algorithm only has partial knowledge of the instance. We assume no structural properties on the fractional solution. In particular, we do *not* assume that the fractional

[4]If there is 1 unit size job with $x_{ij} = 1/m$ for all $i \in [m]$ then any assignment has makespan 1, a factor of m larger than the fractional makespan.

assignment corresponds to a vertex of the linear program for makespan on unrelated machines, a key property used in offline rounding procedures [19, 27].

Now we present an online randomized algorithm for rounding fractional assignments which achieves a competitive ratio of $O(\text{poly}(\log\log m))$ with high probability. Throughout the analysis, we will assume that T is known. Later we discuss how we can remove the assumption on the knowledge of T using a standard doubling analysis. We state our result formally as the following theorem.

THEOREM 5.1. *Let x be a fractional assignment of unrelated machines that is received online and let T be the fractional makespan of x, i.e. $T := \max_i \sum_{j\in N(i)} p_{ij}x_{ij}$. Further, $x_{ij} = 0$ if $p_{ij} > T$. There exists a randomized online algorithm that rounds a fractional assignment to an integer assignment such that the resulting makespan is at most $O((\log\log m)^3 T)$ with high probability.*

5.1 Instance Transformation The first step in our analysis is to convert the instance into a number of simpler instances as we receive it online. Depending on the properties of the job, it will be sent to a procedure for that particular job type.

We redefine the neighborhood for the unrelated case as $N(j) := \{i \mid x_{ij} > 0\}$ be the set of machines in the support for job j. We will refer to this as the **neighborhood** of job j. Recall that job sizes are bounded in the following way: $p_{ij} \le T$ for all $i \in N(j)$. Now the first breakdown we make is to separate jobs into a notion of small and large jobs. For a job j let $S_j = \{i \in N(j) \mid p_{ij} \le T/\log m\}$. We say that a job is small if $\sum_{i\in S_j} x_{ij} \ge 1/2$, and otherwise it is large. Intuitively, a job is small if most of its fractional weight is on machines with small p_{ij} as compared to T. Note that this separation can easily be done online because it only depends on p_{ij} and x_{ij} for a job j.

The interesting case is the large jobs, which we discuss next. The small jobs can be assigned by using randomized rounding as we show in Section 5.2.5. Because the jobs are small, Chernoff bounds ensure no machine is overloaded with high probability.

5.1.1 Transforming Large Jobs We first consider how to round the large jobs. For this, we further break the jobs into cases. For each job j let $B_j = N(j) \setminus S_j$ be the set of machines i in $N(j)$ where $p_{ij} > T/\log m$. Let $\mathcal{B}$ be the set of large jobs. For each large job j we have $\sum_{i\in B_j} x_{ij} \ge 1/2$. We now preprocess the large jobs online creating a new fractional solutions x' where the following properties hold.

LEMMA 5.1. *At a loss of increasing the makespan by a $O(\log\log m)$ factor, the fractional solution x can be converted to a fractional solution x' where the following properties hold:*

- $x'_{ij} \ge 0$ *and* $\sum_{i\in N(j)} x'_{ij} = 1$
- $x'_{ij} \le 2\log\log(m) x_{ij}$
- *If* $x'_{ij} > 0$ *then* $p_{ij} = 2^k T/\log m$ *for some fixed* $k \in [\log\log m]$

This modification can be done for each job individually in an online manner.

This preprocessing step will allow us to assume that the size of the job is the same on all machines in the support of x' for the job.

Proof. Consider the intervals $I_k = [2^{k-1}\frac{T}{\log m}, 2^k \frac{T}{\log(m)}]$ for $k \in [\log\log m]$. We have that $[T\log m, T] = \bigcup_{k=1}^{\log\log m} I_k$. Let $B_{j,k} = \{i \in B_j \mid p_{ij} \in I_k\}$. Since all large jobs have most of their fractional weight on machines with $p_{ij} \in [T/\log m, T]$, by averaging there is a $k \in [\log\log m]$ such that $\sum_{i\in B_{j,k}} x_{ij} \ge \frac{1}{2\log\log m}$. That is, a large fraction, at least $\frac{1}{2\log\log m}$, of a job's fractional assignment is to machines where the sizes are within a factor 2 of each other. Set $x'_{ij} = 0$ for $i \notin B_{j,k}$ and $x'_{ij} = x_{ij}/\sum_{i'\in B_{j,k}} x_{i'j}$ for $i \in B_{j,k}$. It is simple to verify the above properties for this transformation.

Since for all i such that $x'_{ij} > 0$ we have that $p_{ij} \le 2^k T/\log m$, we can think of the job as having a single size $p'_j = 2^k T/\log m$ on its neighborhood of machines for some $k \in [\log\log m]$ by rounding the size up by at most a factor two. ☐

Thus we have reduced the more general unrelated machines instance to an instance of restricted assignment. In the new restricted assignment instance, a job has a fixed size, but can only be assigned to a subset of machines.

Let C_k be the set of large jobs in the k'th class that now have size $2^k T/\log m$. We say that $j \in C_k$ is of **class** k. In the remainder of this section we show how to round the jobs in the k'th class with small loss in the makespan. Since there are $O(\log\log m)$ such classes and we increased each fractional value by at most an $O(\log\log m)$ factor, we lose an extra factor of $O((\log\log m)^2)$ overall. For simplicity we assume throughout the rest of the analysis that the solution x has the properties stated in the claim.

5.2 Rounding A Single Class of Large Jobs We now focus on a single class C_k of large jobs. All jobs in this class have the same size $p'_j = 2^k T/\log(m)$, but a job specific neighborhood $N(j)$ of feasible machines. We break these down into two more cases.

For a job j let $S'_j = \{i \in N(j) \mid x_{ij} \leq \frac{1}{\log^2 m}\}$. We say that a job j's fractional assignment has small support if $\sum_{i \in S'_j} x_{ij} \geq 1/2$, and otherwise it has large support. (This inference can be easily done online). We start by analyzing the jobs with large support.

5.2.1 Jobs with Large Support For jobs with large support, we further preprocess the instance to give it more structural properties. In particular, we will show that by increasing the makespan by a $\log\log m$ factor we can assume that for a fixed job j the values of x_{ij} are either 0 or a single positive value.

LEMMA 5.2. *Fix a class C_k of large jobs. The fractional solution can be modified by increasing the makespan by a factor $O(\log\log m)$ to ensure the following property holds. For each job $j \in C_k$, for all i either $x_{i,j} = 0$ or $x_{i,j} = \frac{2^\ell}{\log^2 m}$ for some fixed $\ell \in [\log\log m]$. This modification can be done for each job individually in an online manner.*

Proof. For a job with large support we have that most of its fractional assignment is on machines i with $\frac{1}{\log^2 m} \leq x_{ij} \leq 1$. Fix a job j. Consider grouping machines by their fractional values in powers of 2. A machine is in **group** $\ell \in [2\log\log m]$ for job j if $x_{ij} \in [\frac{2^\ell}{\log^2 m}, \frac{2^{\ell+1}}{\log^2 m}]$. Let $\mathcal{G}_{j,\ell}$ contain all such machines.

By an averaging argument, there is a group $\mathcal{G}_{j,\ell}$ of machines where the job j has at least a $1/2\log\log m$ of its fractional assignment. That is, there is an $\ell \in [2\log\log m]$ where $\sum_{i \in \mathcal{G}_{j,\ell}} x_{ij} \geq \frac{1}{4\log\log m}$. Let λ_j be the number of machines in this group. Since all the fractional assignments in this group are off by at most a factor of 2 from each other, we might as well consider them to be the same at the cost of a factor of 2. We set $x'_{ij} = 1/\lambda_j$ to be the new fractional assignment for machines in this group and $x'_{ij} = 0$ for machines outside of this group. By construction we have that $\frac{\log^2 m}{2^\ell} \leq \lambda_j \leq \frac{\log^2 m}{2^{\ell-1}}$ for some ℓ. Let D_ℓ be the set of large support jobs with λ_j in the interval $[\frac{\log^2 m}{2^\ell}, \frac{\log^2 m}{2^{\ell-1}}]$. We will refer to a set D_ℓ of jobs as a **group** for some fixed ℓ. Since there are $O(\log\log m)$ such groups of large support jobs, it suffices to consider only a single such group at the cost of increasing the makespan by a $O(\log\log(m))$ factor. □

5.2.2 Rounding a Single Group of Large Support Jobs This section gives the algorithm for the case where jobs have large support. Fix a class D_ℓ of large support jobs. All of these jobs have a neighborhood of size at most λ for some $\lambda \leq \log^2 m$. Our aim is to show that a single iteration of randomized rounding followed by a deterministic greedy assignments suffices to assign these jobs in a good way.

The algorithm we use to round these jobs is as follows. Each job j chooses a random machine in its neighborhood $N(j)$, then checks the machine it chose. If the load incurred by other jobs in class D_ℓ on this machine is greater than $101\log\log mT$, then the job rejects this assignment. In this case the job is added to the set F_ℓ of failed jobs for class ℓ. The jobs in F_ℓ are assigned using a deterministic greedy algorithm.

The greedy algorithm works as follows. For a machine $i \in N(j)$ let its ℓ-load be the number of jobs in F_ℓ that have already been assigned to it. A job $j \in F_\ell$ chooses to be assigned to the a machine with the minimum ℓ-load. This can easily be done online.

By definition, the jobs assigned using randomized rounding contribute $O((\log\log m)T)$ to the makespan. Thus it suffices to bound the contribution of the jobs assigned using greedy.

Let G_ℓ be the bipartite graph consisting of nodes for each job in F_ℓ that rejected their random assignment. The set of machines are on the other side. A job $i \in F_\ell$ is connected to a machine j with an edge if and only if $x_{i,j} \geq 0$. The proof will show that every connected component of G_ℓ has size $O(\text{poly}(\log m))$ with high probability. It then follows that the jobs assigned by greedy contribute $O(\log\log m)T$ to the makespan. This is because the deterministic greedy algorithm [4] is known to achieve a $O(\log \hat{m})$ approximation for any instance of restricted assignment on $\hat{m}$ machines and each connected component is an instance of size $O(\text{poly}(\log m))$ with high probability. Thus, these are "instances" of size $O(\text{poly}(\log m))$ and the makespan of the greedy algorithm as compare to optimal can be at most a $O(\log\log m)T$ factor larger.

In order to show that G_ℓ has small connected components we apply a technique similar to shattering in the distributed computing literature. We will define a special substructure and show that if a connected component of G_ℓ is large, then one of these substructures exist. We will then show that the probability of one of these substructures existing is small; after carefully counting the number of possible substructures and applying a union bound we can conclude that every connected component of G_ℓ is small with high probability.

We start by defining the substructure.

DEFINITION 5.3. *Two jobs j and j' are machine disjoint if $N(j) \cap N(j') = \emptyset$.*

DEFINITION 5.4. *A sequence $j_1, j_2, \ldots, j_\beta$ of jobs is* ***special*** *if all the jobs are pairwise machine disjoint* *and for each k, j_k is within 4 hops of at least one of $j_1, \ldots, j_{k-1}$ in G.*

Later on, machine disjointness will allow us to show that certain events are statistically independent. Note that by definition, all of the jobs in a special sequence must belong to the same connected component. Equipped with these definitions we prove the following lemma:

LEMMA 5.5. *Let C be a connected component of G_ℓ of size at least $\log^c m$ with $c > 7$, then there is a special job sequence of size $\beta = \log m$.*

Proof. We prove the lemma by induction on the size of the sequence. Every large connected component has at least one job, so the base case is trivial. Now for the induction step. Let C be a connected component with size at least $\log^c(m)$. Suppose there is a special job sequence of $\beta - 1$ jobs. We combine these jobs into a single node and start a breadth first search in C. Since the underlying graph is bipartite, the first level of this search consists of machines, the second jobs, and the third machines. If there is any job in the fourth level of this search, then it must be machine disjoint from the first $\beta - 1$. Suppose that there is no such job. Then 3 levels of this search suffices to explore all nodes of C. Thus since the maximum degree of a job or machine is bounded by λ, the size of C is at most:

$$|C| \le (\beta - 1)\lambda^3 \le (\beta - 1)\log^6 m.$$

This leads to a contradiction when $\beta = \log m$ and $c > 7$. Thus such a job in the fourth level exists, yielding a special job sequence of size $\beta = O(\log m)$. □

The proof above gave a way to construct the sequence of jobs given the graph. This also gives us a way to upper bound the number of such special sequences.

LEMMA 5.6. *Let C be a connected component of G_ℓ. There are at most $m(\beta\lambda)^{4\beta}$ special job sequences for $\beta = \log m$. So for $\lambda \le \log^2 n$, this is upper bounded by $m(\log^{12} m)^{\log m} \le m^{1+12\log\log m}$.*

Proof. Let us count special job sequences by following the construction given in the proof of Lemma 5.5. There are at most m jobs we can start with to construct a special sequence. At the i'th step of the construction there are at most $(i\lambda)^4$ possible jobs we can choose to append onto the sequence by traversing out 4 hops in the graph from the currently chosen jobs. Since $i \le \beta$, this is at most $(\beta\lambda)^4$. Thus there are at most $m(\beta\lambda)^{4\beta}$ such special job sequences. □

The previous lemma bounds the number of possible special job sequences. Using this, we can bound the probability that all jobs in a fixed special job sequence fail to be assigned by randomized rounding, and then union bound over all possible special job sequences.

LEMMA 5.7. *Fix a special job sequence σ of length $\beta = \log m$. The probability that all jobs in σ fail to be assigned by randomized rounding is at most $m^{-100\log\log m}$.*

Proof. Recall that a job is failed to be assigned by randomized rounding if it chose a machine with load $> (100\log\log m + 1)T$. Let L_i be the load of jobs that sample machine i in randomized rounding and let X_{ij} be the random variable indicating whether or not job j sampled machine i in randomized rounding. Then we have $L_i = \sum_{j\in N(i)} p'_j X_{ij}$ and in expectation:

$$\mathbb{E}[L_i] \le \sum_{j\in N(i)} p'_j \frac{1}{\lambda} \le T$$

Applying Theorem A.2, we have that the probability that a machine becomes overloaded is:

$$\begin{aligned}\Pr[L_i > T(1 + 100\log\log m)]& \\ &\le \exp\left(-\frac{(100\log\log m)^2}{2 + \log\log m}\right)\\ &\le \exp(-100\log\log m)\end{aligned}$$

Now let σ be a special job sequence. For $j \in \sigma$, we have that the probability that j fails to be assigned by randomized rounding is at most the probability that any machine becomes overloaded. Since $N(j) \cap N(j') = \emptyset$ for all $j, j' \in \sigma$, we have that the jobs in σ fail to be assigned by randomized rounding independently. Thus the probability that all jobs in σ fail to be assigned is at most

$$\begin{aligned}\exp(-100\log\log m)^\beta &= \exp(-100\beta\log\log m)\\ &= m^{-100\log\log m},\end{aligned}$$

proving the lemma. □

We are now ready to show that every connected component in G_ℓ is small with high probability.

LEMMA 5.8. *With high probability, every connected component of G_ℓ has size $O(\log^c m)$.*

Proof. By Lemma 5.5, it suffices to show that no special sequence of jobs of length $\beta = \log m$ exists in G_ℓ with high probability. For a fixed special sequence of jobs, Lemma 5.7 states the probability that it is in G_ℓ is at most $m^{-100\log\log m}$. Now taking a union bound over all special sequences, the probability that there exists a special sequence of jobs in G_ℓ of length $\beta = \log m$ is at most

$$\begin{aligned}\left(m^{-100\log\log m}\right)&\left(m(\log^{12} m)^{\log m}\right)\\ &= \left(m^{-100\log\log m}\right)\left(m^{1+12\log\log m}\right)\\ &\le m^{-5},\end{aligned}$$

where we use the bound on the number of special job sequences from Lemma 5.6. Thus no special sequence of length $\log m$ exists in G_ℓ with high probability. ∎

LEMMA 5.9. *Fix a class ℓ of large support jobs. We can round the jobs in this class with makespan at most $O(\log\log m)T$ with high probability.*

Proof. Each job in this class is assigned by randomized rounding or by a separate greedy assignment. The jobs assigned by randomized rounding contribute $O(\log\log m)T$ to the makespan by definition of our algorithm. Looking at the instance after removing all jobs assigned by randomized rounding, by Lemma 5.8 every connected component in the underlying graph of this instance has size at most $O(\log^c m)$ for some constant c with high probability. Thus running Greedy on this remaining instance contributes an extra $O(\log\log m)T$ to the makespan. In aggregate, the contribution to the makespan from this class of large support jobs is $O(\log\log m)T$. ∎

5.2.3 Rounding Jobs with Small Support

In this section we consider the case where for jobs j that have small support. Recall that this means that $\sum_{i\in S'_j} x_{ij} \ge 1/2$ where S'_j is the set of machines i where $x_{ij} \le \frac{1}{\log^2 m}$. In this case, we set $x_{ij} = 0$ for $i \notin S'_j$. Then we renormalize the remaining fractional assignment to ensure $\sum_i x_{ij} = 1$ for all j, increasing each assignment by at most a factor of 2. Note that since we are rounding jobs of a single class, we may assume all jobs are unit sized and the makespan is T is bounded by $O(\log m)$ by rescaling.

The algorithm that we use to handle this type of jobs, Iterated Randomized Rounding, works in several phases. Each phase k maintains a fractional load T_k and integer load $L(i,k)$ for each machine. The load $L(i,k)$ counts the total size of jobs assigned using this procedure in phase k. In each phase we attempt to randomly assign a job to several machines, however this fails if $L(i,k)$ is too large for the sampled machines. In the case of failure, the job goes on to the next phase. Our analysis will show that after $O(\log\log m)$ phases only few jobs will be left and we will handle them separately.

Interestingly the procedure can be done for each job individually, where $L(i,k)$ is the load assigned to the machine so far among jobs in phase k. Thus the procedure can be done online.

Algorithm 2 Iterated Randomized Rounding

```
1: for each job $j$ do
2:     for each phase $k = 0, 1, 2, \dots$ do
3:         For each $i \in N(j)$ assign $j$ to $i$ independently with probability $x_{ij}$
4:         If $L(i,k) > 10T$ for all sampled machines, then $j$ goes to the next phase
5:         Otherwise assign $j$ to $i$ such that $L(i,k) < 10T$ and increase the load of all sampled machines $i$ with $L(i,k) < 10T$
```

In the first phase of Iterated Randomized Rounding, the fractional load of each machine is at most T. The intuition behind this algorithm is that as the algorithm goes to higher and higher phases, then this fractional load should decrease quickly. Let T_k be the bound on the fractional load in phase k of Iterated Random Rounding. We will to show that $T_{k+1} \le \rho T_k$ for some constant $\rho \in (0,1)$ with high probability. We can continue running Iterated Random Rounding while this bound is relatively large. When we reach a phase where the fractional load becomes too small, any job that is still unassigned becomes a "leftover" job and we assign it using a different technique. The number of phases of Iterated Randomized Rounding we need will be $O(\log\log m)$, implying that the contribution to the makespan of jobs with small support will be $O(\log\log m)T$ plus the contribution of "leftover" jobs. We start the analysis with the following lemma.

LEMMA 5.10. *Let T_k be an upper bound on the fractional load of all jobs that make it to phase k in Iterated Randomized Rounding, with $T_0 = T$. For all $k = 0, 1, 2, \dots$ we have that $T_{k+1} \le \rho T_k$ for some constant $\rho \in (0,1)$ with high probability.*

Proof. Consider a phase k and let T_k be as in the lemma statement. We need to upper bound the fractional load of jobs that fail to be assigned in phase k, and hence contribute to the fractional load in phase $k+1$. Let $L(i,k)$ be the load of machine i in phase k and let $N(i,k)$ be the set of jobs that can be assigned to machine i in phase k. We have that

$$\mathbb{E}[L(i,k)] = \sum_{j\in N(i,k)} p_j x_{ij} \le T_k$$

The probability that any job fails to be assigned in phase k is at most the probability that machine i has load more than $10T$ in phase k. By Markov's inequality this is at most

$$\Pr[L(i,k) > 10T] \le \frac{\mathbb{E}[L(i,k)]}{10T} \le \frac{T_k}{10T} \le \frac{1}{10}$$

since $T_k \le T$ for all k. Now fix a machine i^*. We are interested in how much fractional load i^* may potentially contribute to the next phase. Let $Z(k,i^*,i) = \sum_{j \in N(i,k)} x_{i^*j} I(j \text{ picks } i)$. Intuitively, if j picks machine i and j ends up going to phase $k+1$, then j will contribute x_{i^*j} to i^*'s fractional load. First we bound $Z(k,i^*,i)$ with high probability.

Claim 5.11. *For each i, $Z(k,i^*,i) \le \left(\frac{d+1}{\log^2 m}\right) T_k$ with high probability for some constant $d > 0$ when $T_k = \Omega(\frac{1}{\log m})$.*

Proof. In expectation we have

$$\begin{aligned}\mathbb{E}[Z(k,i^*,i)] &= \sum_{j \in N(i,k)} x_{i^*j} \mathbb{E}[I(j \text{ picks } i)] \\ &= \sum_{j \in N(i,k)} x_{i^*j} x_{ij} \\ &\le \frac{2}{\log^2 m} \sum_{j \in N(i,k)} x_{ij} \le \frac{2T_k}{\log^2 m}\end{aligned}$$

Since $Z(k,i^*,i)$ is a sum of independent random variables in the form needed for Theorem A.2, we apply this theorem with $a \le \frac{2}{\log^2 m}$ and $v \le \frac{2}{\log^2 m}\mathbb{E}[Z(k,i^*,i)]$. Thus taking $\lambda = dT_k/\log^2 m$, we have

$$\begin{aligned}\Pr[Z(k,i^*,i) > &\frac{2T_k}{\log^2 m} + \lambda] \\ &\le \exp\left(\frac{-\lambda^2}{\frac{4\mathbb{E}[Z(k,i^*,i)]}{\log^2 m} + \frac{4\lambda}{3\log^2 m}}\right) \\ &\le \exp\left(\frac{-d^2 T_k^2}{\frac{4T_k}{\log^2 m} + \frac{4dT_k}{3\log^2 m}}\right) \\ &= \exp\left(-\left(\frac{d^2}{4+4d/3}\right) T_k \log^2 m\right) \\ &= \exp(-c' \log m) = m^{-c},\end{aligned}$$

for some constant c' depending on d. Note from the second to last line to the last line we used the fact that $T_k = \Omega(1/\log m)$. Now choosing c', d large enough and taking a union bound over all machines we see that $Z(k,i^*,i) \le \frac{d+1}{\log^2 m} T_k$ for all i with probability at least $1 - 1/m^{c'-1}$. ∎

To bound the fractional load in the next phase define $Z(k,i^*) = \sum_i Z_{k,i^*,i} \mathbb{I}(i \text{ overloaded in phase } k)$. Note that this is a bound on the fractional load that survives phase k and hence goes to phase $k+1$. Thus any bound on this random variable that holds for all i^* yields a bound on T_{k+1}.

Claim 5.12. *For each i^*, $Z(k,i^*) \le \rho T_k$ for some $\rho \in (0,1)$ with high probability when $T_k = \Omega(1/\log m)$.*

Proof. For each i^*, in expectation we have

$$\begin{aligned}&\mathbb{E}[Z(k,i^*)] \\ &\quad= \sum_i \mathbb{E}[Z(k,i^*,i)] \Pr[i \text{ overloaded in phase } k] \\ &\quad\le \frac{1}{10} \sum_i \sum_{j \in N(i,k)} x_{i^*j} x_{ij} \\ &\quad= \frac{1}{10} \sum_{j \in N(i,k)} x_{i^*j} \sum_i x_{ij} \\ &\quad= \frac{1}{10} \sum_{j \in N(i,k)} x_{i^*j} \le \frac{1}{10} T_k.\end{aligned}$$

By Claim 5.11, we have that $Z(k,i^*,i) \le \frac{d+1}{\log^2 m} T_k$ for all i with high probability. Now since $Z(k,i^*)$ is defined as a sum of independent random variables[5], we can again apply Theorem A.2 to this random variable with $a \le \frac{d+1}{\log^2 m} T_k$ and $v \le \frac{d+1}{\log^2 m} T_k \mathbb{E}[Z(k,i^*)]$. Taking $\lambda = qT_k$ we have

$$\begin{aligned}&\Pr[Z(k,i^*) > \frac{1}{10}T + \lambda] \\ &\quad\le \exp\left(\frac{-\lambda^2}{\frac{2(d+1)T_k\mathbb{E}[Z(k,i^*)]}{\log^2 m} + \frac{2(d+1)T_k\lambda}{3\log^2 m}}\right) \\ &\quad= \exp\left(\frac{-q^2 T_k^2}{\frac{2(d+1)T_k^2}{10\log^2 m} + \frac{2(d+1)qT_k^2}{3\log^2 m}}\right) \\ &\quad= \exp\left(-\left(\frac{q^2}{(d+1)/5 + 2q(d+1)/3}\right)\log^2(m)\right)\end{aligned}$$

Now we choose q such that $\rho = \frac{1}{10} + q \in (0,1)$ and the above expression becomes sufficiently small, i.e. $\le m^{-c}$ for some constant $c > 1$. Then taking a union bound over all machines i^*, we have that $Z(k,i^*) \le (\frac{1}{10}+q)T_k = \rho T_k$ for all i^* with high probability. ∎

By Claim 5.12, we can take $T_{k+1} = \max_{i^*} Z(k,i^*)$, which is at most ρT_k with high probability, proving the lemma. ∎

[5]Machines in the same phase become overloaded independently of one another because the machines are selected independently by each job and a job can increase the load of multiple machines in the same phase (even if it is assigned to a single one).

Now we have a sequence of bounds $T_0, T_1, \dots, T_k, \dots$ that hold with high probability. Note that in the proof of Claim 5.11, we required that $T_k = \Omega(1/\log m)$. Thus it only makes sense to consider this sequence while this bound is true. We assume that $T_0 = T = \Omega(1)$ and that $T = O(\log m)$. Thus there are $O(\log\log m)$ phases before T_k becomes $O(1/\log m)$. Jobs that make it this far without being assigned become "leftover" jobs and we assign them using a different technique.

5.2.4 Rounding the "Leftover" Large Jobs with Small Support

The "leftover" jobs are small support jobs that survived too many phases of random assignments. The setup of this case is the following. Each job has $x_{i,j} \le \frac{1}{\log^2 m}$. Let $T(i) = \sum_j x_{i,j}$ be the fractional load of machine i. It is the case that $T(i) \le \frac{1}{64\log m}$ for each machine $i \in m$.

Consider the following algorithm. Each job j independently samples a set $M(j)$ of machines from $N(j)$ where machine $i \in N(j)$ is in the set with probability $32\log m \cdot x_{i,j}$. Each job j is assigned to the machine which has the smallest load in this phase. Now we show that with high probability that for each job j it is always the case that $M(j)$ contains a machine i such that $i \notin M(j')$ for all other jobs j. Thus, with high probability each machine is assigned at most one job.

Lemma 5.13. *With probability at least $1 - \frac{1}{m}$ it is the case that for all jobs j the set $M(j)$ contains a machine i not in $M(j')$ for all $j' \neq j$.*

Proof. Fix any job j. First we show that $|M(j)| \ge 5\log m$ with probability at least $1 - \frac{1}{m^4}$. Indeed, let X_i be 1 if job j samples machine i and 0 otherwise. By definition of the algorithm $\mathbb{E}[X_i] = 32\log m \cdot x_{i,j}$ and $\mathbb{E}[\sum_{i=1}^m X_i] = 32\log m$. Using TheoremA.1 we have that the probability $|M(j)| = \sum_{i=1}^m X_i$ is smaller than $5\log m$ is at most $\exp(-\frac{32\log m}{8}) = \frac{1}{m^4}$. Thus $|M(j)| \ge 5\log m$ with probability at least $1 - \frac{1}{m^4}$.

Consider any machine i. Let G_i be the random variable with value 1 if there is no job $j' \neq j$ such that $i \in M(j')$. Otherwise G_i has value 0. By definition of the algorithm we have the following.

$$\begin{aligned}
\Pr[G_i = 1] &= \Pr[i \notin M(j') \ \ \forall j \neq j'] \\
&= \prod_{j' \neq j} \Pr[i \notin M(j')] \\
&= \prod_{j' \neq j} (1 - 32\log m \cdot x_{i,j'}) \\
&\ge \exp(-32\log m \sum_{j' \neq j} x_{i,j'}) \\
&\ge \exp(-32\log m T(i)) \\
&\ge \frac{1}{e^{1/2}} \quad [T(i) \le \tfrac{1}{64\log m} \text{ by assumption}]
\end{aligned}$$

Let E_j denote the event that $|M(j)| \ge 5\log m$. Consider the probability that $G := \sum_{i\in M(j)}(1 - G_i)$ given that E_j occurs. This is the probability of the bad event where no machine in $M(j)$ is selected only by j given E_j. Given a set $M(j)$, we know $\mathbb{E}[G \mid M(j)] = \mathbb{E}[\sum_{i\in M(j)} G_i \mid M(j)] = \sum_{i\in M(j)} \mathbb{E}[G_i] = \sum_{i\in M(j)} \frac{1}{e^{1/2}} = \frac{|M(j)|}{e^{1/2}}$. This holds for all sets $M(j)$. Thus, $\mathbb{E}[G \mid E_j] \ge \frac{5\log m}{e^{1/2}}$. Using Theorem A.1 the probability that $G := \sum_{i\in M(j)}(1 - G_i)$ given E_j is at most $\exp(-\frac{5\log m}{2e^{1/2}}) \ge \frac{1}{m^{3/2}}$.

We now put the above facts together. The probability E_j does not occur is at most $\frac{1}{m^4}$. The probability that $G = 0$ given E_j occurs is at most $\frac{1}{m^{3/2}}$. One of these events must occur for G to be 0. Thus, a union bound says that the probability $G = 0$ is at most $\frac{1}{m^4} + \frac{1}{m^{3/2}} \le \frac{1}{m}$. Therefore, the probability $G \ge 1$ happens with probability at least $1 - \frac{1}{m}$, implying that there is a machine i in $M(j)$ such that no other job j' has $i \in M(j')$. □

5.2.5 Assigning the Small Jobs

In this section we show how the small jobs can be assigned. For each small job, we preprocess its fractional assignment as follows. First we set $x'_{ij} = 0$ for $i \notin S_j$, then set $x'_{ij} = x_{ij}/\sum_{i'\in S_j} x_{i'j}$ for $i \in S_j$. It is easy to verify that this transformation has the following properties.

- $x'_{ij} \ge 0$ and $\sum_{i\in N(j)} x'_{ij} = 1$
- $x'_{ij} \le 2x_{ij}$
- If $x'_{ij} > 0$ then $p_{ij} \le T/\log(m)$

This is all the preprocessing we need to do for small jobs. Afterwards all small jobs are assigned using randomized rounding.

Note that the preprocessing and the randomized rounding can be executed online.

Algorithm 3 Randomized Rounding

```
1: for each job $j$ do
2:     Sample $i \in N(j)$ according to the distribution $\{x_{ij}\}_{i=1}^m$
3:     Assign job $j$ to machine $i$
```

LEMMA 5.14. *Let S be the set of all small jobs. Applying randomized rounding with the preprocessed fractional assignments x'_{ij} yields a makespan of $O(T)$ for just the jobs in S with high probability.*

Proof. Let X_{ij} be the indicator random variable for the event that $j \in S$ is assigned to machine i. By definition of randomized rounding, we have that the random variables X_{ij} are independent for varying j. Let $L_i^S = \sum_{j \in S \cap N(i)} p_{ij} X_{ij}$ be the load of the small jobs on machine i. Computing expectations we have

$$\mathbb{E}[L_i^S] = \sum_{j \in S \cap N(i)} p_{ij} x'_{ij} \le 2 \sum_{j \in S \cap N(i)} p_{ij} x_{ij} \le 2T.$$

Applying Theorem A.2 with $v = \sum_{j \in S \cap N(i)} p_{ij}^2 x'_{ij} \le 2\frac{T}{\log m}\mathbb{E}[L_i^S] \le \frac{2T^2}{\log m}$, $a \le \frac{T}{\log m}$, and $\lambda = cT$ for some large enough constant c we have

$$\begin{aligned}
\Pr[L_i^S > 2T + \lambda] &\le \Pr[L_i^S > \mathbb{E}[L_i^S] + \lambda] \\
&\le \exp\left(\frac{-\lambda^2}{2v + a\lambda/3}\right) \\
&= \exp\left(\frac{-c^2T^2}{\frac{4T^2}{\log m} + \frac{cT^2/3}{\log m}}\right) \\
&= \exp\left(-d \log m)\right) = m^{-d},
\end{aligned}$$

where $d = \frac{3c^2}{12+c}$ is a constant. Now taking a union bound over all machines, we have $L_i^S \le (c+2)T$ for all i with probability at least $1 - m^{d-1}$. Since c is some constant, this proves the lemma. ∎

6 Lower Bounds for Online Rounding

In this section we aim to prove the following result, as stated in Section 1.

THEOREM 6.1. *Let x be a fractional assignment of restricted assignment jobs that is received online and let $T := \max\{\max_i \sum_{j \in N(i)} p_j x_{ij}, \max_j p_j\}$ be the adjusted fractional makespan. No deterministic algorithm for converting x to an integer assignment can be $o(\log m / \log\log m)$-competitive with respect to T. Further, no randomized algorithm for the same task can be $o(\log\log m / \log\log\log m)$-competitive with respect to T.*

6.1 Deterministic Lower Bound

In this section we give a bad instance for deterministic online rounding algorithms for makespan. A rounding algorithm converts a fractional solution $x_{ij} \ge 0$ in which $\sum_{i \in N(j)} x_{ij} = 1$ for each job j into an assignment of job j on some machine $i \in N(j)$. For a sequence of n jobs with fractional solutions, the fractional makespan is $\max_i \sum_{j \in N(i)} p_j x_{ij}$. Due to bad integrality gaps for some instances, we compare our algorithms to $T := \max\{\max_i \sum_{j \in N(i)} p_j x_{ij}, \max_j p_j\}$, which we refer to as the adjusted fractional makespan. We show that for any deterministic online rounding algorithm there is an instance for which it incurs a large makespan when compared to T.

LEMMA 6.1. *For any deterministic online rounding algorithm A there exists a sequence of unit size jobs such that A has makespan* $\log m / \log\log m$ *while the fractional makespan is* $1/\log\log m$ *and the optimal solution has makespan* 1.

Proof. Fix the deterministic rounding algorithm A. Let λ, p and m be integers such that $\lambda^p = m$. The exact value of λ will be chosen later. We consider an instance with m machines. Each job in our sequence will have a size of 1 and a neighborhood of cardinality λ and fractional solution $x_{ij} = \frac{1}{\lambda}$ for each $i \in N(j)$. The bad sequence of jobs will consist of p phases. In the first phase, we release m/λ jobs, each with *disjoint* neighborhoods of size λ. We observe where A assigns these jobs. Since these jobs had disjoint neighborhoods they get assigned to different machines. Let M' be the set of machines where a job got assigned and recurse on this set of machines, starting a new phase. Note that $|M'| = m/\lambda$. This recursion continues until we run out of machines. By our choice of λ, p, m, there are p phases since $\lambda^p = m$.

Letting $\lambda = \log(m)$, we observe that the rounding algorithm's makespan is $p = \log(m)/\log\log(m)$, while the optimal solution in hindsight has makespan 1. It is also easy to verify that the fractional makespan is $p/\lambda = 1/\log\log(m)$. ∎

For the above sequence $T = \max\{1/\log\log m, 1\}$, and so this implies the $\Omega(\log m / \log\log m)$ lower bound for deterministic algorithms. Note that using a uniform fractional assignment on other sequences of jobs such as the one described in [4] does not suffice. For an analysis of a deterministic algorithm on these sequences of jobs we would have $T = \Theta(\log m)$, while the algorithm's makespan would be $\Omega(\log m)$. Thus the resulting ratio would be constant.

6.2 Randomized Lower Bound

Applying Yao's principle [31], we aim to give a distribution over instances such that any deterministic algorithm A has a large makespan in expectation when compared to the corresponding fractional makespan. Lemma 6.1 implies that for each algorithm A, there exists an instance I_A for which A has makespan at least $\Omega(\log m/\log\log m)$ factor bigger than the corresponding value of T for the instance. We start by describing a distribution for instances on m machines. Afterwards we boost this to a distribution for instances on $M := mk$ machines for some parameter k. We conclude the lower bound by analyzing the resulting makespan in terms of M.

6.2.1 Distribution for instances on m machines

As hinted above, our distribution over instances on m machines will be uniform over all possible instances described in Lemma 6.1. Fix integers λ, p and m such that $\lambda^p = m$. In particular we use $\lambda = \log m$ and $p = \log m/\log\log m$ as in Lemma 6.1. Let $\mathcal{I}$ be the set of all instances of the form given by Lemma 6.1 with parameters λ, p and m. Then our distribution over instances is uniform over $\mathcal{I}$, i.e. for any instance I we set

$$\Pr[\text{send } I] = \begin{cases} 1/|\mathcal{I}| & \text{if } I \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

We now analyze this distribution and state some key properties it has.

Proposition 6.1. *The set of instances $\mathcal{I}$ has the following properties:*

1. $|\mathcal{I}| \le O\left(\lambda^{O(p^2\lambda^p)}\right)$
2. $|\mathcal{I}| \ge \Omega(\lambda^{\Omega(p\lambda^p)})$
3. *For every $I \in \mathcal{I}$, the corresponding fractional makespan is $1/\log\log m$*
4. *For every deterministic algorithm A, there exists $I_A \in \mathcal{I}$ such that A has makespan at least $\log m/\log\log m$.*

Proof. The last two points follow from Lemma 6.1, so we prove the first two points. To bound the number of such instances we describe a process to generate an instance of $\mathcal{I}$. We start by choosing a set of λ machines from the set of m machines, then λ from the remaining $m-\lambda$ machines, and so on. Each set corresponds to unit size job with the set of machines as its neighborhood. Afterwards, we choose of m/λ machines, one from each set, and recurse on these machines. We count the number of ways to pick the initial set of jobs as

$$\binom{m}{\lambda}\binom{m-\lambda}{\lambda}\cdots\binom{m-(m/\lambda)\lambda}{\lambda} = \frac{m!}{(\lambda!)^{m/\lambda}}$$

To see this equality note that corresponding terms in the numerators and denominators cancel out, leaving just the first $m!$ and a $\lambda!$ for each binomial term. After picking these jobs, there are $\lambda^{m/\lambda}$ ways to choose the set of machines to recurse on since there are m/λ sets of size λ and we choose one machine from each. Applying this idea recursively, we get that

$$|\mathcal{I}| = \prod_{\ell=0}^{p} \frac{(m/\lambda^\ell)!}{(\lambda!)^{m/\lambda^\ell}}\lambda^{m/\lambda^\ell}$$

At some loss, we upper bound this by taking the first term (since it's the largest) in the product above to the p'th power.

$$|\mathcal{I}| \le \left(\frac{m!}{(\lambda!)^{m/\lambda}}\lambda^{m/\lambda}\right)^p$$

We now use the fact that $m = \lambda^p$ to express everything in terms of only λ and p.

$$|\mathcal{I}| \le \left(\frac{(\lambda^p)!}{(\lambda!)^{\lambda^{p-1}}}\lambda^{\lambda^{p-1}}\right)^p$$

Using Sterling's approximation for factorial we have that $(\lambda^p)! = O(\lambda^{p(\lambda^p+1)})$ and $\lambda! \ge \Omega(\lambda^\lambda)$. Now combining these two inequalities we have that

$$|\mathcal{I}| \le O\left(\lambda^{O(p^2\lambda^p)}\right)$$

Now to get the lower bound we look at the first term in the product above and substitute $m = \lambda^p$.

$$|\mathcal{I}| \ge \frac{m!}{(\lambda!)^{m/\lambda}}\lambda^{m/\lambda} = \frac{(\lambda^p)!}{(\lambda!)^{\lambda^{p-1}}}\lambda^{\lambda^{p-1}}$$

Again using Sterling's approximation for factorial we have $(\lambda^p)! = \Omega(\lambda^{p\lambda^p})$ and $\lambda! = O(\lambda^{\lambda+1})$. Combining these yields

$$|\mathcal{I}| \ge \Omega(\lambda^{\Omega(p\lambda^p)})$$

completing the proof. □

The above proposition implies that for any deterministic algorithm A, the probability that A incurs makespan at least $\Omega(\log m/\log\log m)$ is $1/|\mathcal{I}| \ge \Omega(1/\lambda^{O(p^2\lambda^p)})$

6.2.2 Boosting the Distribution

Since the above distribution on m machines has a low probability of incurring a high makespan on some deterministic algorithm A, we need to boost this in order to conclude our lower bound. Let $k := |\mathcal{I}|$ and set $M := mk$. We construct a distribution for instances on M machines as follows. Partition the set of M machines into k groups of m machines and on each group independently sample an instance from $\mathcal{I}$.

Let $I_1, I_2, \ldots, I_k$ be the sampled instances on each group of machines. For any deterministic algorithm A we have that group s has makespan $\log m / \log \log m$ if $I_s = I_A$, which happens with probability at least $1/k$. Using this we can show the following lower bound on the expected makespan of algorithm A.

LEMMA 6.2. *The expected makespan of any deterministic algorithm A on the above distribution over instances on M machines is at least $\Omega(\log m / \log \log m)$.*

Proof. Algorithm A has makespan $\Omega(\log m / \log \log m)$ if any group's sampled instance is equal to A's bad instance, I_A. This occurs with the following probability:

$$\begin{aligned}\Pr[\vee_{s=1}^{k}(I_s = I_A)] &= 1 - \Pr[\wedge_{s=1}^{k}(I_s \neq I_A)] \\ &= 1 - \prod_{s=1}^{k}(1 - \Pr[I_s = I_k]) \\ &\geq 1 - (1 - 1/k)^k \\ &\geq 1 - 1/e = \Omega(1)\end{aligned}$$

So A's expected makespan is $\Omega(\log m / \log \log m)$. □

Finally, we just need to conclude that $\log \log M = O(\log m)$ and $\log \log \log M = \Omega(\log \log m)$ to finish the proof of the lower bound.

LEMMA 6.3. *For $M := mk$, we have $\log \log(M) = O(\log m)$ and $\log \log \log M = \Omega(\log \log m)$*

Proof. Since $k = O\left(\lambda^{O(p^2 \lambda^p)}\right)$ and $m = \lambda^p$, we have $M = O\left(\lambda^{O(p^2 \lambda^p)}\right)$ as well. Thus we have $\log M = O\left(p^2 \lambda^p \log \lambda\right)$ and

$$\log \log M = O(\log p + p \log \lambda + \log \log \lambda) = O(p \log \lambda).$$

Now using that $p = \log m / \log \log m$ and $\lambda = \log m$ we have that $\log \log M = O(\log m)$.

Similarly, since $k = \Omega(\lambda^{\Omega(p\lambda^p)})$ and $m = \lambda^p$, we have $M = mk = \Omega(\lambda^{\Omega(p\lambda^p)})$. Thus we have $\log M = \Omega(p \lambda^p \log \lambda)$ and $\log \log M = \Omega(\log p + p \log \lambda + \log \log \lambda)$. Thus we have

$$\log \log \log M = \Omega(\log p) = \Omega(\log \log m)$$

since $p = \log m / \log \log m$. □

Finally, we see that Lemmas 6.1, 6.2, and 6.3 imply Theorem 6.1.

7 Conclusion

We study the use of predictions for scheduling algorithms, in particular demonstrating how to use machine learned predictions to improve the online competitive ratio for minimizing the makespan in the restricted assignment setting. After proposing the concept of predicting machine weights, we show how to use these weights to construct fractional assignments online that are robust to possible errors in the predictions. We then give a new randomized algorithm for rounding the fractional assignments online while incurring only $O((\log \log m)^3)$ loss in the competitive ratio.

Many interesting open problems remain. An immediate avenue is generalizing our work to other scheduling settings. For instance, it is not clear what kinds of predictions can be used to effectively recover a near optimal fractional assignment for the unrelated machines setting. The ML advice paradigm also led us to investigate the problem of rounding fractional assignments online. We showed a lower bound of $\Omega\left(\frac{\log \log m}{\log \log \log m}\right)$, tightening the gap between the bounds remains a challenging open problem. More generally, understanding how to effectively use predictions to formally improve competitive ratios is an interesting area for further research.

Acknowledgement

We thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

References

[1] S. Agrawal, M. Zadimoghaddam, and V. Mirrokni. Proportional allocation: Simple, distributed, and diverse matching with high entropy. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 99–108, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[2] N. Ailon, B. Chazelle, K. L. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. *SIAM J. Comput.*, 40(2):350–375, 2011.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, May 1997.

[4] Y. Azar, J. S. Naor, and R. Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, Mar. 1995.

[5] E. Balkanski, A. Rubinstein, and Y. Singer. The power of optimization from samples. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4017–4025, 2016.

[6] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3):20:1–20:45, 2016.

[7] J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen. Online algorithms with advice: A survey. *SIGACT News*, 47(3):93–129, Aug. 2016.

[8] S. Bubeck and A. Slivkins. The best of both worlds: Stochastic and adversarial bandits. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pages 42.1–42.23, 2012.

[9] D. Chakrabarty, S. Khanna, and S. Li. On $(1,\epsilon)$)-restricted assignment makespan minimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1087–1101, 2015.

[10] R. Cole and T. Roughgarden. The sample complexity of revenue maximization. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 243–252, New York, NY, USA, 2014. ACM.

[11] N. R. Devanur and T. P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 71–78, 2009.

[12] P. Dütting, Z. Feng, H. Narasimhan, and D. C. Parkes. Optimal auctions through deep learning. *CoRR*, abs/1706.03459, 2017.

[13] M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 270–277, 2016.

[14] C.-Y. Hsu, P. Indyk, D. Katabi, and A. Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

[15] K. Jansen and L. Rohwedder. On the configuration-lp of the restricted assignment problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2670–2678, 2017.

[16] W. Kong, C. Liaw, A. Mehta, and D. Sivakumar. A new dog learns old tricks: RL finds classic optimization algorithms. In *International Conference on Learning Representations*, 2019.

[17] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 489–504, New York, NY, USA, 2018. ACM.

[18] R. Kumar, M. Purohit, A. Schild, Z. Svitkina, and E. Vee. Semi-online bipartite matching. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 50:1–50:20, 2019.

[19] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1):259–271, Jan 1990.

[20] T. Lykouris and S. Vassilvtiskii. Competitive caching with machine learned advice. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[21] M. Mahdian, H. Nazerzadeh, and A. Saberi. Online optimization with uncertain information. *ACM Trans. Algorithms*, 8(1):2:1–2:29, 2012.

[22] A. M. Medina and S. Vassilvitskii. Revenue optimization with approximate bid predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1856–1864, 2017.

[23] V. S. Mirrokni, S. O. Gharan, and M. Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1690–1701, 2012.

[24] M. Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 462–471, 2018.

[25] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning.* The MIT Press, 2012.

[26] M. Purohit, Z. Svitkina, and R. Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 9684–9693, 2018.

[27] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993.

[28] O. Svensson. Santa claus schedules jobs on unrelated machines. *SIAM J. Comput.*, 41(5):1318–1341, 2012.

[29] E. Vee, S. Vassilvitskii, and J. Shanmugasundaram. Optimal online assignment with forecasts. In *Proceedings of the 11th ACM Conference on Electronic Commerce*, EC '10, pages 109–118, New York, NY, USA, 2010. ACM.

[30] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.

[31] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer*

Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, pages 222–227. IEEE Computer Society, 1977.

A Concentration Inequalities

THEOREM A.1. *Let $X_1, X_2, \dots, X_t$ be a collection of independent Bernoulli RV's with* $\Pr[X_i = 1] = p_i$. *Let* $X = \sum_i X_i$ *and* $\mu = \sum_i \mathbb{E}[X_i]$. *Then for all* $\delta \in (0, 1)$

$$\Pr[X < (1-\delta)\mu] \leq \exp\left(\frac{-\delta^2\mu}{2}\right).$$

THEOREM A.2. (BERNSTEIN) *Let $X_1, \dots, X_t$ be independent Bernoulli RV's with* $\Pr[X_i = 1] = p_i$ *and let $a_1, \dots, a_t$ be non-negative scalars. Let* $X = \sum_i a_i X_i$, $v = \sum_i a_i^2 p_i$, *and* $a = \max_i a_i$. *Then for all* $\lambda > 0$

$$\Pr[X > \mathbb{E}[X] + \lambda] \leq \exp\left(\frac{-\lambda^2}{2v + 2a\lambda/3}\right).$$

THEOREM A.3. (UNION BOUND) *Let $A_1, \dots, A_t$ be a collection of events, then*

$$\Pr[A_1 \cup \ldots \cup A_t] \leq \sum_{i=1}^{t} \Pr[A_i]$$

B Learning Weights

In Section 3.2 we described a two step procedure—given an instance of the problem, we generate feature and weight pairs, (f, w), and then use any off the shelf algorithm to find the best h that maps features to weights.

Instead, we can consider directly building a neural network that finds a low makespan solution. Using neural networks for optimization problems is a nascent area, see for instance recent work by [16, 12]. The key is to find a good representation of the solution, and then allow the network to find the best fit function from features to the representation. In our case, it will be helpful to look at the logarithmic transform of the weights, let $z_i = \log w_i$. Recall the softmax function $\sigma : \mathbb{R}^m \to \mathcal{S}^{m-1}$, where $\mathcal{S}^d$ is the d-dimensional simplex. Given a vector $z \in \mathbb{R}^m$, $\sigma(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^m \exp(z_j)}$. It is clear that the fractional allocation $x_{ij}(w)$ is simply the softmax of the corresponding z vector, with coordinates representing infeasible assignments zeroed out. Let δ_j be the binary vector representing feasible machines for job j, and $\circ$ be the pointwise (Haddamard) product.

Then, the goal of the network is to find a transformation from f to z that minimizes

$$\phi(z) = \max_{i \in [m]} \sum_j \sigma(\delta_j \circ z)_i.$$

While the softmax function is not convex, it is nonetheless frequently used in neural networks, with good results.

C Proof of Rounding Theorem

In this section we combine the results of the analysis in Section 5 to conclude Theorem 5.1. We recall the statement of this result here.

THEOREM C.1. (THEOREM 5.1 RESTATED) *Let x be a fractional assignment of restricted assignment jobs that is received online and let T be the fractional makespan of x, i.e.* $T := \max_i \sum_{j \in N(i)} p_j x_{ij}$. *There exists a randomized online algorithm that rounds a fractional assignment to an integer assignment such that the resulting makespan is at most* $O((\log\log m)^3 T)$ *with high probability.*

Proof. The result mostly follows from the lemmas in Section 5. The worst case for our algorithm is due to the large jobs with large support. In our algorithm there are $O(\log\log m)$ classes of large jobs by Lemma 5.1. Within a fixed class of large jobs there are $O(\log\log m)$ classes of large support jobs by Lemma 5.2. Finally, rounding a fixed class of large support jobs loses can be done with makespan $O((\log\log m)T)$ due to Lemma 5.9. The other cases of our algorithm lose fewer factors of $\log\log m$, and all cases of our algorithm succeed with high probability. Combining these losses we see that the makespan is at most $O((\log\log m)^3 T)$ with high probability. ∎

D The Doubling Lemma

Throughout the body of the paper, we described our algorithms and results as if we knew the optimal makespan T^*. We now show how to remove this assumption via a standard online doubling analysis.

LEMMA D.1. *There exists an online algorithm using predictions yielding the same competitiveness as the algorithm guaranteed by Theorems 3.1 and 5.1 that does not need to know the optimal makespan T^* and succeeds with high probability.*

The proof of this lemma has been omitted from this version of the paper.