Algorithms with Prediction Portfolios

Michael Dinitz
Johns Hopkins University
mdinitz@cs.jhu.edu

Sungjin Im
UC Merced
sim3@ucmerced.edu

Thomas Lavastida*
University of Texas at Dallas
thomas.lavastida@utdallas.edu

Benjamin Moseley
Carnegie Mellon University
moseleyb@andrew.cmu.edu

Sergei Vassilvitskii
Google Research
sergeiv@google.com

Abstract

The research area of algorithms with predictions has seen recent success showing how to incorporate machine learning into algorithm design to improve performance when the predictions are correct, while retaining worst-case guarantees when they are not. Most previous work has assumed that the algorithm has access to a single predictor. However, in practice, there are many machine learning methods available, often with incomparable generalization guarantees, making it hard to pick the best method a priori. In this work we consider scenarios where multiple predictors are available to the algorithm and the question is how to best utilize them.

Ideally, we would like the algorithm's performance to depend on the quality of the *best* predictor. However, utilizing more predictions comes with a cost, since we now have to identify which prediction is the best. We study the use of multiple predictors for a number of fundamental problems, including matching, load balancing, and non-clairvoyant scheduling, which have been well-studied in the single predictor setting. For each of these problems we introduce new algorithms that take advantage of multiple predictors, and prove bounds on the resulting performance.

1 Introduction

An exciting recent line of research attempts to go beyond traditional worst-case analysis of algorithms by equipping algorithms with *machine-learned predictions*. The hope is that these predictions allow the algorithm to circumvent worst case lower bounds when the predictions are good, and approximately match them otherwise. The precise definitions and guarantees vary with different settings, but there have been significant successes in applying this framework for many different algorithmic problems, ranging from general online problems to classical graph algorithms (see Section 1.2 for a more detailed discussion of related work, and [35] for a survey). In all of these settings it turns out to be possible to define a "prediction" where the "quality" of the algorithm (competitive ratio, running time, etc.) depends the "error" of the prediction. Moreover, in at least some of these settings, it has been further shown that this prediction is actually learnable with a small number of samples, usually via standard ERM methods [18].

Previous work has shown the power of accurate predictions, and there are numerous examples showing improved performance in both theory and practice. However, developing accurate predictors remains an art, and a single predictor may not capture all of the subtleties of the instance space. Recently, researchers have turned to working with *portfolios of predictors*: instead of training a single model, train multiple models, with the hope that one of them will give good guarantees.

*Work was done while the author was at Carnegie Mellon University.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

It is easy to see why the best predictor in a portfolio may be *significantly* better than a one-size fits all predictor. First, many of the modern machine learning methods come with a slew of hyperparameters that require tuning. Learning rate, mini-batch size, optimizer choice, all of these have significant impact on the quality of the final solution. Instead of commiting to a single setting, one can instead try to cover the parameter space, with the hope that some of the predictors will generalize better than others. Second, problem instances themselves may come from complex distributions, consisting of many latent groups or clusters. A single predictor is forced to perform well on average, whereas multiple predictors can be made to "specialize" to each cluster.

In order to take advantage of the increased accuracy provided by the portfolio approach, we must adapt algorithms with predictions to take advantage of multiple predictions. To capture the gains in performance, the algorithm must perform as if equipped with the best predictor, auto-tuning to use the best one available in the portfolio. However, it is easy to see that there should be a cost as the size of the portfolio grows. In the extreme, one can add every possible prediction to the portfolio, providing no additional information, yet now requiring high performance from the algorithm. Therefore, we must aim to minimize the dependence on the number of predictions in the portfolio.

We remark that the high level set up may be reminiscent of expert- or bandit-learning literature. However, there is a critical distinction. In expert and bandit learning, we are given a sequence of problem instances, and the goal is to compete (minimize regret) with respect to the best prediction *averaged* over the whole sequence. On the other hand, in our setup, we aim to compete with the best predictor on a *per-instance* basis.

Previous work on multiple predictions. Bhaskara et al. studied an online linear optimization problem where the learner seeks to minimize the regret, provided access to multiple hints [15]. Inspired by the work, Anand et al. recently studied algorithms with multiple learned predictions in [7], proving strong bounds for important online covering problems including online set cover, weighted caching, and online facility location. It was a significant extension of the work [23] which studied the rent-or-buy problem with access to two predictions. However, their techniques and results are limited to online covering problems. Moreover, they do not discuss the learning aspects at all: they simply assume that they are given k predictions, and their goal is to have competitive ratios that are based on the minimum error of any of the k predictions. (They actually compete against a stronger dynamic benchmark, but for our purposes this distinction is not important.)

On the other hand Balcan et al. [13] look at this problem through a data driven algorithm lens and study the sample complexity and generalization error of working with k (as opposed to 1) parameter settings. The main difference from our work is that they also aim learn a selector, which selects one of the k parameters *prior* to beginning to solve the problem instance. In contrast, in this work we make the selection during the course of the algorithm, and sometimes switch back and forth while honing in on the best predictor.

1.1 Our Results and Contributions

In this paper we study three fundamental problems, min-cost perfect matching, online load balancing, and non-clairvoyant scheduling for total completion time, in this new setting. Each of these has seen significant success in the single-prediction model but is not covered by previous multiple-prediction frameworks. Our results are primarily theoretical, however we have included a preliminary empirical validation of our algorithm for min-cost perfect matching in Appendix A.

For each of these we develop algorithms whose performance depends on the error of the *best* prediction, and explore the effect of the number of predictions, k. Surprisingly, in the case of matching and scheduling we show that using a limited number of predictions is essentially free, and has *no* asymptotic impact on the algorithm's performance. For load balancing, on the other hand, we show that the cost of multiple predictions grows *logarithmically* with k, again implying a tangible benefit of using multiple predictions. We now describe these in more detail.

Min-Cost Perfect Matching. We begin by showcasing our approach with the classical min-cost perfect matching problem in Section 3. This problem was recently studied by [17, 18] to show that it is possible to use learned predictions to improve *running times* of classical optimization problems. In particular, [18] showed it is possible to speed up the classical Hungarian algorithm by predicting dual values, and moreover that it is possible to efficiently (PAC-)learn the best duals. We show that

simple modifications of their ideas lead to similar results for multiple predictions. Interestingly, we show that as long as $k \leq O(\sqrt{n})$, the extra "cost" (running time) of using k predictions is negligible compared to the cost of using a single prediction, so we can use up to $\sqrt{n}$ predictions "for free" while still getting running time depending on the best of these predictions. Moreover, since in this setting running time is paramount, we go beyond sample complexity to show that it is also computationally efficient to learn the best k predictions.

Online Load Balancing with Restricted Assignments. We continue in Section 4 with the fundamental load balancing problem. In this problem there are m machines, and n jobs which appear in online fashion. Each job has a size, and a subset of machines that it can be assigned to. The goal is to minimize the maximum machine load (i.e., the makespan). This problem has been studied extensively in the traditional scheduling and online algorithms literature, and recently it has also been the subject of significant study given a single prediction [27, 28, 30]. In particular, Lattanzi, Lavastida, Moseley, and Vassilvitskii [27] showed that there exist per machine "weights" and an allocation function so that the competitive ratio of the algorithm depends logarithmically on the maximum error of the predictions. We show that one can use k predictions and incur an additional $O(\log k)$ factor in the competitive ratio, while being competitive with the error of the *best* prediction. Additionally, we show that learning the best k predicted weights (in a PAC sense) can be done efficiently.

Non Clairvoyant Scheduling Finally, in Section 5 we move to the most technically complex part of this paper. We study the problem of scheduling n jobs on a single machine, where all jobs are released at time 0, but where we do not learn the length of a job until it actually completes (the *non-clairvoyant* model). Our objective is to minimize the sum of completion times. This problem has been studied extensively, both with and without predictions [25, 32, 37, 39]. Most recently, Lindermayr and Megow [32] suggested that we use an *ordering* as the prediction (as opposed to the more obvious prediction of job sizes), and use the difference between the cost induced by the predicted ordering and the cost induced by the instance-optimal ordering as the notion of "error". In this case, simply following the predicted ordering yields an algorithm with error equal to the prediction error.

We extend this to the multiple prediction setting, which turns out to be surprisingly challenging. The algorithm of [32] is quite simple: follow the ordering given by the prediction (and run a 2-competitive algorithm in parallel to obtain a worst-case backstop). But we obviously cannot do this when we are given multiple orderings! So we must design an algorithm which considers all k predictions to build a schedule that has error comparable to the error of the *best* one. Slightly more formally, we prove that we can bound the sum of completion times by $(1+\epsilon)\mathrm{OPT}$ plus $\mathrm{poly}(1/\epsilon)$ times the error of the best prediction, under the mild assumption that no set of at most $\log \log n$ jobs has a large contribution to OPT.

To do this, we first use sampling techniques similar to those of [25] to estimate the size of the approximately ϵn'th smallest job without incurring much cost. We then use even more sampling and partial processing to determine for each prediction whether its ϵn prefix has many jobs that should appear later (a bad sequence) or has very few jobs that should not be in the prefix (a good sequence). If all sequences are bad then every prediction has large error, so we can use a round robin schedule and charge the cost to the prediction error. Otherwise, we choose one of the good orderings and follow it for its ϵn prefix (being careful to handle outliers). We then recurse on the remaining jobs.

1.2 Related Work

As discussed, the most directly related papers are Anand et al. [7] and Balcan, Sandholm, and Vitercik [13]; these give the two approaches (multiple predictions and portfolio-based algorithm selection) that are most similar to our setting. The single prediction version of min-cost bipartite matching was studied in [17, 18], the single prediction version of our load balancing problem was considered by [27, 28, 30] (and a different though related load balancing problem was considered by [4]), and the single prediction version of our scheduling problem was considered by [32] with the same prediction that we use (an ordering) and earlier with different predictions by [25, 39, 41]. Online scheduling with estimates of the true processing times was considered in [11, 12].

More generally, there has been an enormous amount of recent progress on algorithms with predictions. This is particularly true for online algorithms, where the basic setup was formalized by [33] in the

context of caching. For example, the problems considered include caching [26, 33, 40], secretary problems [9, 21], ski rental [5, 39, 41], and set cover [14]. There has also been recent work on going beyond traditional online algorithms, including work on running times [17, 18], algorithmic game theory [2, 22, 34], and streaming algorithms [1, 19, 24]. The learnability of predictions for online algorithms with predictions was considered by [6]. They give a novel loss function tailored to their specific online algorithm and prediction, and study the sample complexity of learning a mapping from problem features to a prediction. While they are only concerned with the sample complexity of the learning problem, we also consider the computational complexity, giving polynomial time $O(1)$-approximate algorithms for the learning problems associated with min-cost matching and online load balancing.

The above is only a small sample of the work on algorithms with predictions. We refer the interested reader to a recent survey [35], as well as a recently set up website which maintains a list of papers in the area [31].

2 Learnability of k Predictions and Clustering

In this section we discuss the learnability of k predictions and its connection to k-median clustering, which we apply to specific problems in later sections.

Learnability and Pseudo-dimension: Consider a problem $\mathcal{I}$ and let $\mathcal{D}$ be an unknown distribution over instances of $\mathcal{I}$. We are interested in the learnability of predictions for such problems with respect to an error function. Suppose that the predictions come from some space Θ and for a given instance $I \in \mathcal{I}$ and prediction $\theta \in \Theta$ the error is $\eta(I, \theta)$. Given S independent samples $\{I_s\}_{s=1}^{S}$ from $\mathcal{D}$, we would like to compute $\hat{\theta} \in \Theta$ such that with probability at least $1 - \delta$, we have that

$$\mathbb{E}_{I \sim \mathcal{D}}[\eta(I, \hat{\theta})] \leq \min_{\theta \in \Theta} \mathbb{E}_{I \sim \mathcal{D}}[\eta(I, \theta)] + \epsilon. \tag{1}$$

We would like S (the sample complexity) to be polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$ and other parameters of the problem $\mathcal{I}$ (e.g. the number of vertices in matching, or the number of jobs and machines in scheduling and load balancing).

The natural algorithm for solving this problem is empirical risk minimization (ERM): take $\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{S} \sum_{s=1}^{S} \eta(I_s, \theta)$. The sample complexity of ERM, i.e. how large S should be so that (1) holds, can be understood in terms of the pseudo-dimension of the class of functions $\{\eta(\cdot, \theta) \mid \theta \in \Theta\}$. More generally, the pseudo-dimension can be defined for any class of real valued function on some space $\mathcal{X}$.

Definition 2.1. *[8, 36, 38] Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$. Let $C = \{x_1, x_2, \ldots, x_S\} \subset \mathcal{X}$. We say that that C is* shattered *by $\mathcal{F}$ if there exist real numbers $r_1, \ldots, r_S$ so that for all $C' \subseteq C$, there is a function $f \in \mathcal{F}$ such that $f(x_i) \leq r_i \iff x_i \in C'$ for all $i \in [S]$. The* pseudo-dimension *of $\mathcal{F}$ is the largest S such that there exists an $C \subseteq X$ with $|C| = S$ that is shattered by $\mathcal{F}$.*

The pseudo-dimension allows us to give a bound on the number of samples required for uniform convergence, which in turn can be used to show that ERM is sufficient for achieving (1).

Theorem 2.2. *[8, 36, 38] Let $\mathcal{D}$ be a distribution over a domain X and $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to [0, H]$ with pseudo-dimension $d_{\mathcal{F}}$. Consider S independent samples $x_1, x_2, \ldots, x_S$ from $\mathcal{D}$. There is a universal constant c_0, such that for any $\epsilon > 0$ and $\delta \in (0, 1)$, if $S \geq c_0 \left(\frac{H}{\epsilon}\right)^2 (d_{\mathcal{F}} + \ln(1/\delta))$ then we have*

$$\left| \frac{1}{s} \sum_{s=1}^{S} f(x_i) - \mathbb{E}_{x \sim \mathcal{D}}[f(x)] \right| \leq \epsilon$$

for all $f \in \mathcal{F}$ with probability at least $1 - \delta$.

We can extend this learning problem to the setting of multiple predictions. The setup is the same as above, except that now we are interested in outputting k predictions $\hat{\theta}^1, \hat{\theta}^2, \ldots, \hat{\theta}^k$ such that with probability at least $1 - \delta$:

$$\mathbb{E}_{I \sim \mathcal{D}} \left[\min_{\ell \in [k]} \eta(I, \hat{\theta}^\ell) \right] \leq \min_{\theta^1, \theta^2, \ldots, \theta^k \in \Theta} \mathbb{E}_{I \sim \mathcal{D}} \left[\min_{\ell \in [k]} \eta(I, \theta^\ell) \right] + \epsilon. \tag{2}$$

We can again consider ERM algorithms for this task, and we would like to bound the sample complexity. We do this by showing that if the pseudo-dimension of the class of functions associated with one prediction is bounded, then it is also bounded for k predictions. More formally, we want to bound the pseudo-dimension of the class of functions $\{\min_{\ell\in[k]} \eta(\cdot,\theta^\ell) \mid \theta^1,\theta^2,\ldots,\theta^k \in \Theta\}$. This can be done via the following result combined with Theorem 2.2 (assuming that the pseudo-dimension of $\{\eta(\cdot,\theta) \mid \theta\in\Theta\}$ is bounded).

Theorem 2.3. *Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$ with pseudo-dimension d and let $\mathcal{F}^k := \{F(x) = \min_{\ell\in[k]} f^\ell(x) \mid f^1, f^2, \ldots, f^k \in \mathcal{F}\}$. Then the pseudo-dimension of $\mathcal{F}^k$ is at most $\tilde{O}(dk)$.*

Proof. To show this we first relate things back to VC-dimension.

Proposition 2.4. *Let $\mathcal{F}$ be a class of real valued functions on $\mathcal{X}$. Define $\mathcal{H}$ as a class of binary functions on $\mathcal{X}\times\mathbb{R}$ as $\mathcal{H} = \{h(x,r) = \operatorname{sgn}(f(x)-r) \mid f\in\mathcal{F}\}$. Then the pseudo-dimension of $\mathcal{F}$ equals the VC dimension of $\mathcal{H}$.*

The above proposition follows directly from the definition of pseudo- and VC-dimensions. The next lemma we need is well known.

Proposition 2.5 (Sauer-Shelah Lemma)**.** *If $\mathcal{H}$ has VC-dimension d and $x_1,\ldots,x_m$ is a sample of size m, then the number of sets shattered by $\mathcal{H}$ is at most $O(m^d)$.*

Let $x_1,\ldots,x_m \in \mathcal{X}$ and $r_1,\ldots,r_m\in\mathbb{R}$ be given. We upper bound the number of possible labelings induced by $\mathcal{F}_k$ on this set. Note that on this sample the Sauer-Shelah Lemma implies that the number of labelings induced by $\mathcal{F}$ on this sample is at most $O(m^d)$. $\mathcal{F}_k$ allows us to choose k functions from $\mathcal{F}$ so this increases the number of possible labelings to at most $O(m^{dk})$. We shatter this set if this bound is greater than 2^m. Reorganizing these bounds implies that $m = \tilde{O}(dk)$, which implies the upper bound on the pseudo-dimension of $\mathcal{F}_k$. □

The associated ERM problem for computing k predictions becomes more interesting. Recall that in this problem we are given a sample of S instances $I_1, I_2, \ldots, I_S \sim \mathcal{D}$ and we want to compute $\hat{\theta}^1,\hat{\theta}^2,\ldots,\hat{\theta}^k\in\Theta$ in order to minimize $\frac{1}{S}\sum_{s=1}^S \min_{\ell\in[k]}\eta(I_s,\hat{\theta}^\ell)$. This can be seen as an instance of the k-median clustering problem where we want to cluster the "points" $\{I_s\}_{s=1}^S$ by opening k "facilities" from the set Θ and the cost of assigning I_s to θ is $\eta(I_s,\theta)$. In general, this problem may be hard to solve or even approximate. In the case that the costs have some metric structure, then it is known how to compute $O(1)$-approximate solutions [29]. For minimum cost matching (Section 3 and load balancing (Section 4), we will show that the ERM problem can be seen as a k-median problem on an appropriate (pseudo-)metric space.

Metrics and Clustering: Recall that $(\mathcal{X},d)$ is a metric space if the distance function $d : \mathcal{X}\times\mathcal{X}\to\mathbb{R}_+$ satisfies the following properties:

1. For all $x,y\in\mathcal{X}$, $d(x,y) = 0 \iff x = y$
2. For all $x,y\in\mathcal{X}$, $d(x,y) = d(y,x)$
3. For all $x,y,z\in\mathcal{X}$, $d(x,z)\le d(x,y)+d(y,z)$

If we replace the first property with the weaker property that for all $x\in\mathcal{X}, d(x,x)=0$, then we call $(\mathcal{X},d)$ a pseudo-metric space.

Given a finite set of points $X\subseteq\mathcal{X}$, the k-median clustering problem is to choose a subset $C\subseteq\mathcal{X}$ of k centers to minimize the total distance of each point in X to its closest center in C. In notation, the goal of k-median clustering is to solve

$$\min_{C\subseteq\mathcal{X},|C|=k}\sum_{x\in X}\min_{c\in C} d(x,c) \tag{3}$$

In our settings it will often be challenging to optimize C over all of $\mathcal{X}$, so at an $O(1)$-factor loss to the objective we can instead optimize C over X. Formally, we have the following standard lemma.

Lemma 2.6. *Let $(\mathcal{X}, d)$ be a pseudo-metric space and let X be a finite subset of $\mathcal{X}$. Then for all $k > 0$ we have*

$$\min_{C \subseteq X, |C|=k} \sum_{x \in X} \min_{c \in C} d(x, c) \leq 2 \cdot \min_{C \subseteq \mathcal{X}, |C|=k} \sum_{x \in X} \min_{c \in C} d(x, c).$$

Proof. Let $C^* \subseteq \mathcal{X}$ be an optimal solution to the problem on the right hand side of the inequality, and let its cost be OPT. We consider a mapping $\phi : C^* \to X$, which gives us a solution to the problem on the left hand side by taking $C = \{\phi(c) \mid c \in C^*\}$[2]. For $c \in C^*$, define $\phi(c) = \arg\min_{x \in X} d(x, c)$ We will argue that the cost of C is at most 2OPT. Let $x \in X$ and let $c^* = \arg\min_{c \in C^*} d(x, c)$ be its closest center in C^*, then we have

$$\min_{c \in C} d(x, c) \leq d(x, \phi(c^*)) \leq d(x, c^*) + d(c^*, \phi(c^*)) \leq 2d(x, c^*) = 2 \min_{c \in C^*} d(x, c)$$

The second to last inequality follows from the triangle inequality and the last follows from the definition of ϕ. Now summing over all $x \in X$ yields that the cost of using C is at most 2OPT. □

3 Minimum Cost Bipartite Matching with Predicted Duals

In this section we study the minimum cost bipartite matching problem with multiple predictions. The case of a single prediction has been considered recently [17, 18], where they used dual values as a prediction and showed that the classical Hungarian algorithm could be sped up by using appropriately learned dual values. Our goal in this section is to extend these results to multiple predictions, i.e., multiple duals. In particular, in Section 3.2 we show that we can use k duals and get running time comparable to the time we would have spent if we used the single best of them in the algorithm of [18], with no asymptotic loss if k is at most $O(\sqrt{n})$. Then in Section 3.3 we show that k predictions can be learned with not too many more samples (or running time) than learning a single prediction.

3.1 Problem Definition and Predicted Dual Variables

In the minimum cost bipartite matching problem we are given a bipartite graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, with edge costs $c \in \mathbb{Z}^E$. The objective is to output a perfect matching $M \subseteq E$ which minimizes the cost $c(M) := \sum_{e \in E} c_e$. This problem is exactly captured by the following primal and dual linear programming formulations.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e & & \\ & \sum_{e \in N(i)} x_e = 1 & \forall i \in V & \quad \text{(MWPM-P)} \\ & x_e \geq 0 & \forall e \in E & \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{i \in V} y_i & & \\ & y_i + y_j \leq c_e & \forall e = ij \in E & \quad \text{(MWPM-D)} \end{aligned}$$

Dinitz et al. [18] studied initializing the Hungarian algorithm with a prediction $\hat{y}$ of the optimal dual solution y^*. They propose an algorithm which operates in two steps (see Algorithm 1 for pseudo-code). First, the predicted dual solution $\hat{y}$ may not be feasible, so they give an $O(n+m)$ time algorithm which recovers feasibility (which we refer to as Make-Feasible). Second, the now-feasible dual solution is used in a primal-dual algorithm such as the Hungarian algorithm (which we refer to as Primal-Dual) and they show that the running time depends on the ℓ_1 error in the predicted solution. In addition to this they show that learning a good initial dual solution is computationally efficient with low sample complexity. More formally, they proved the following theorems.

Theorem 3.1 (Dinitz et al. [18])**.** *Let (G, c) be an instance of minimum cost bipartite matching and $\hat{y}$ be a prediction of an optimal dual solution y^*. Algorithm 1 returns an optimal solution and runs in time $O(m\sqrt{n} \cdot \|y^* - \hat{y}\|_1)$. Moreover, the Make-Feasible step of Algorithm 1 runs in $O(n+m)$ time.*

[2] In the case that $|C| < k$, adding arbitrary points from X to C so that $|C| = k$ can only decrease the cost from just using C.

Theorem 3.2 (Dinitz et al. [18]). *Let $\mathcal{D}$ be an unknown distribution over instances (G, c) on n vertices and let $y^*(G, c)$ be an optimal dual solution for the given instance. Given S independent samples from $\mathcal{D}$, there is a polynomial time algorithm that outputs a solution $\hat{y}$ such that*

$$\mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\|y^*(G,c) - \hat{y}\|_1\right] \leq \min_y \mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\|y^*(G,c) - y\|_1\right] + \epsilon$$

with probability $1 - \delta$ where $S = \text{poly}(n, \frac{1}{\epsilon}, \frac{1}{\delta})$.

Algorithm 1 Minimum cost matching with a predicted dual solution

1: **procedure** PREDICTEDPRIMAL-DUAL($G, c, \hat{y}$)
2: $y \leftarrow$MakeFeasible$(G, c, \hat{y})$
3: $M \leftarrow$Primal-Dual(G, c, y)
4: Return M
5: **end procedure**

3.2 Using k Predicted Dual Solutions Efficiently

Given k predicted dual solutions $\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^k$, we would like to efficiently determine which solution has the minimum error for the given problem instance. Note that the predicted solutions may still be infeasible and that we do not know the target optimal dual solution y^*. We propose the following simple algorithm which takes as input k predicted solutions and whose running time depends only on the ℓ_1 error of the *best* predicted solution. First, we make each predicted solution feasible, just as before. Next, we select the (now-feasible) dual solution with highest dual objective value and proceed running the primal-dual algorithm with only that solution. See Algorithm 2 for pseudo-code.

Algorithm 2 Minimum cost matching with k predicted dual solutions

1: **procedure** k-PREDICTEDPRIMAL-DUAL($G, c, \hat{y}^1, \hat{y}^2, \ldots, \hat{y}^k$)
2: **for** $\ell \in [k]$ **do**
3: $y^\ell \leftarrow$MakeFeasible$(G, c, \hat{y}^\ell)$
4: **end for**
5: $\ell' \leftarrow \arg\max_{\ell\in[k]} \sum_{i\in V} y_i^\ell$
6: $M \leftarrow$Primal-Dual$(G, c, y^{\ell'})$
7: Return M
8: **end procedure**

We have the following result concerning Algorithm 2. To interpret this result, note that the cost for increasing the number of predictions is $O(k(n + m))$, which will be dominated by the $m\sqrt{n}$ term we pay for running the Hungarian algorithm unless k is extremely large (certainly larger than $\sqrt{n}$) or there is a prediction with 0 error (which is highly unlikely). Hence we can reap the benefit of a large number of predictions "for free".

Theorem 3.3. *Let (G, c) be a minimum cost bipartite matching instance and let $\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^k$ be predicted dual solutions. Algorithm 2 returns an optimal solution and runs in time $O(k(n + m) + m\sqrt{n} \cdot \min_{\ell\in[k]} \|y^* - \hat{y}^\ell\|_1)$.*

Proof. The correctness of the algorithm (i.e., returning an optimal solution) follows from the correctness of Algorithm 1. For the running time, we clearly spend $O(k(n + m))$ time making each predicted solution feasible, thus we just need to show the validity of latter term in the running time. Let $\ell' = \arg\max_{\ell\in[k]} \sum_{i\in V} y_i^\ell$ be the solution chosen in line 5 of Algorithm 2 and let $\ell^* = \arg\min_{\ell\in[k]} \|y^* - \hat{y}^\ell\|_1$ be the solution with minimum error. Recall that for each $\ell \in [k]$, y^ℓ is the resulting *feasible* dual solution from calling Make-Feasible on $\hat{y}^\ell$. By the analysis from [18], we have that $\|y^* - y^{\ell^*}\|_1 \leq 3\|y^* - \hat{y}^{\ell^*}\|_1$, so it suffices to show that the number of primal-dual iterations will be bounded by $\|y^* - y^{\ell^*}\|_1$. By our choice of ℓ', we have $\sum_i y_i^{\ell'} \geq \sum_i y_i^{\ell^*}$, therefore we have that $\sum_i (y_i^* - y^{\ell'}) \leq \sum_i (y_i^* - y_i^{\ell^*}) \leq \|y^* - y^{\ell^*}\|_1$. From the analysis in [18], we have that the number of primal-dual iterations will be at most $\sum_i (y_i^* - y^{\ell'})$, completing the proof. □

3.3 Learning k Predicted Dual Solutions

Next we extend Theorem 3.2 to the setting where we output k predictions. Let $\mathcal{D}$ be a distribution over problem instances (G, c) on n vertices. We show that we can find the best set of k predictions. More formally, we prove the following theorem.

Theorem 3.4. *Let $\mathcal{D}$ be an unknown distribution over instances (G, c) on n vertices and let $y^*(G, c)$ be an optimal dual solution for the given instance. Given S independent samples from $\mathcal{D}$, there is a polynomial time algorithm that outputs k solutions $\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^k$ such that*

$$\mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\min_{\ell\in[k]} \|y^*(G,c) - \hat{y}^\ell\|_1\right] \leq O(1) \cdot \min_{y^1,y^2,\ldots,y^k} \mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\min_{\ell\in[k]} \|y^*(G,c) - y^\ell\|_1\right] + \epsilon$$

with probability $1-\delta$ where $S = \text{poly}(n, k, \frac{1}{\epsilon}, \frac{1}{\delta})$.

Proof. By Theorem 7 in [18] and Theorems 2.2 and 2.3, we get the polynomial sample complexity. Thus we just need to give a polynomial time ERM algorithm to complete the proof. Let $\{(G^s, c^s\}_{s=1}^S$ be the set of sampled instances. We start by computing $z^s = y^*(G^s, c^s) \in \mathbb{Z}^V$ for each $s \in [S]$. Consider the ERM problem where we replace the expectation by a sample average:

$$\min_{y^1,y^2,\ldots,y^k} \frac{1}{S}\sum_{s=1}^{S} \min_{\ell\in[k]} \|z^s - y^\ell\|_1$$

This can be seen as a k-median clustering problem where each predicted solution y^ℓ is a cluster center and distances are given by the ℓ_1 norm. Thus we can find an $O(1)$-approximate solution $\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^k$ to this problem which is of polynomial size in polynomial time (for example by applying the algorithm due to [29]). □

4 Online Load Balancing with Predicted Machine Weights

We now apply our framework to online load balancing with restricted assignments. In particular, we consider proportional weights, which have been considered in prior work [27, 28, 30]. Informally, we show in Section 4.2 that if β is the cost of the *best* of the k predictions, then even without knowing a priori which prediction is best, we get cost of $O(\beta \log k)$. Then in Section 4.3 we show that it does not take many samples to actually learn the best k predictions.

4.1 Problem Definition and Proportional Weights

In online load balancing with restricted assignments there is a sequence of n jobs which must be assigned to m machines in an online fashion. Upon seeing job j, the online algorithm observes its size $p_j > 0$ and a neighborhood $N(j) \subseteq [m]$ of *feasible* machines. The algorithm must then choose some feasible machine $i \in N(j)$ to irrevocably assign the job to before seeing any more jobs in the sequence. We also consider fractional assignments, i.e. vectors belonging to the set $X = \{x \in \mathbb{R}_+^{m\times n} \mid \forall j \in [n], \sum_i x_{ij} = 1, \text{ and } x_{ij} = 0 \iff i \notin N(j)\}$.

Prior work studied the application of proportional weights[3, 27, 28, 30]. Intuitively, a prediction in this setting is a weighting of *machines*, which then implies an online assignment, which is shown to be near-optimal. Slightly more formally, suppose that we are given weights w_i for each machine i. Then each job j is *fractionally* assigned to machine i to a fractional amount of $\frac{w_i}{\sum_{i'\in N(j)} w_{i'}}$. Notice that given weights, this also gives an online assignment. It is known that there exist weights for any instance where the fractional solution has a near optimal makespan, even though there are only m "degree of freedom" in the weights compared to mn in an assignment. That is, for all machines i, $\sum_{j\in[n]} p_j \cdot \frac{w_i}{\sum_{i'\in N(j)} w_{i'}}$ is at most a $(1+\epsilon)$ factor larger than the optimal makespan for any constant $\epsilon > 0$ [3, 27].

Let w^* be a set of near optimal weights for a given instance. Lattanzi et al. [27] showed the following theorem:

Theorem 4.1. *Given predicted weights $\hat{w}$, there is an online fractional algorithm which has makespan $O(\log(\eta(\hat{w}, w^*)\text{OPT})$, where $\eta(\hat{w}, w^*) := \max_{i\in[m]} \max(\frac{\hat{w}_i}{w_i^*}, \frac{w_i^*}{\hat{w}_i})$ to be the error in the prediction.*

Moreover, this fractional assignment can be converted online to an integral assignment while losing only an $O(\log \log m)$ factor in the makespan [27, 30]. Thus, we focus on constructing fractional assignments that are competitive with the best prediction in hindsight.

4.2 Combining Fractional Solutions Online

Given k different predicted weight vectors $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$, we want to give an algorithm which is competitive against the *minimum* error among the predicted weights, i.e. we want the competitiveness to depend upon $\eta_{\min} := \min_{\ell \in [k]} \eta(\hat{w}^\ell, w^*)$.

The challenge is that we do not know up front which $\ell \in [k]$ will yield the smallest error, but instead learn this in hindsight. For each $\ell \in [k]$, let x^ℓ, be the resulting fractional assignment from applying the fractional online algorithm due to [27] with weights $\hat{w}^\ell$. This fractional assignment is revealed one job at a time.

We give an algorithm which is $O(\log k)$-competitive against any collection of k fractional assignments which are revealed online. Moreover, our result applies to the unrelated machines setting, in which each job has a collection of machine-dependent sizes $\{p_{ij}\}_{i \in [m]}$. The algorithm is based on the doubling trick and is similar to results in [10] which apply to metrical task systems. Let $\beta := \min_{\ell \in [k]} \max_i \sum_j p_{ij} x_{ij}^\ell$ be the best fractional makespan in hindsight. As in previous work, our algorithm is assumed to know β, an assumption that can be removed [27]. At a high level, our algorithm maintains a set $A \subseteq [k]$ of solutions which are good with respect to the current value of β, averaging among these. See Algorithm 3 for a detailed description. We have the following theorem.

Theorem 4.2. *Let $x^1, x^2, \ldots, x^k$ be fractional assignments which are revealed online. If Algorithm 3 is run with $\beta := \min_{\ell \in [k]} \max_i \sum_j p_{ij} x_{ij}^\ell$, then it yields a solution of cost $O(\log k) \cdot \beta$ and never reaches the fail state (line 7 in Algorithm 3).*

Let $\beta_\ell = \max_i \sum_j p_{ij} x_{ij}^\ell$ and OPT be the optimal makespan. Theorem 4.1 shows that $\beta_\ell \leq O(\log \eta_\ell)\mathrm{OPT}$. The following corollary is then immediate:

Corollary 4.3. *Let $w^1, w^2, \ldots, w^k$ be the predicted weights with errors $\eta^1, \eta^2, \ldots, \eta^k$. Then Algorithm 3 returns a fractional assignment with makespan at most $\mathrm{OPT} \cdot O(\log k) \cdot \min_{\ell \in [k]} \log(\eta^\ell)$.*

Algorithm 3 Algorithm for combining fractional solutions online for load balancing.

```
1: procedure COMBINE-LOADBALANCING(β)
2:     A ← [k]                                          ▷ Initially all solutions are good
3:     for each job j do
4:         Receive the assignments x^1, x^2, ..., x^k
5:         A(j, β) ← {ℓ ∈ A | ∀i ∈ [m], x_ij^ℓ > 0 ⟹ p_ij x_ij^ℓ ≤ β}
6:         if A = ∅ or A(j, β) = ∅ then
7:             Return "Fail"
8:         end if
9:         ∀i ∈ [m], x_ij ← (1/|A(j,β)|) Σ_{ℓ∈A(j,β)} x_ij^ℓ
10:        B ← {ℓ ∈ A | max_{i∈[m]} Σ_{j'≤j} p_ij' x_ij'^ℓ > β}        ▷ Bad solutions w.r.t. β
11:        A ← A \ B
12:    end for
13: end procedure
```

The analysis relies on the following decomposition of machine i's load. Note that in our algorithm we assign a weight $\alpha_j^\ell \in [0, 1]$ to solution ℓ when computing the fractional assignment for job j. That is, we take $x_{ij} = \sum_{\ell \in A} \alpha_j^\ell x_{ij}^\ell$ where $\alpha_j^\ell = 1/|A(j, \beta)|$ if $\ell \in A(j, \beta)$ and 0 otherwise. Then the decomposition of machine i's load L_i is

$$L_i = \sum_j p_{ij} x_{ij} = \sum_j p_{ij} \sum_{\ell \in [k]} \alpha_j^\ell x_{ij}^\ell = \sum_{\ell \in [k]} \left(\sum_j \alpha_j^\ell p_{ij} x_{ij}^\ell \right). \tag{4}$$

Now we define $C_i^\ell := \sum_j \alpha_j^\ell p_{ij} x_{ij}^\ell$ to be the contribution of solution ℓ to machine i's load. Without loss of generality suppose that the order in which solutions are removed from S in Algorithm 3 is $1, 2, \ldots, k$.

Lemma 4.4. *For all $i \in [m]$ and each $\ell \in [k]$, we have that $C_i^\ell \leq \frac{2\beta}{k-\ell+1}$.*

Proof. Let j be the job in which ℓ is removed from A by our algorithm. Before this, for all $j' < j$ we had that ℓ's fractional makespan was at most β. Thus we can write C_i^ℓ as:

$$C_i^\ell = \sum_j \alpha_j^\ell p_{ij} x_{ij}^\ell = \sum_{j'<j} \alpha_{j'}^\ell p_{ij'} x_{ij'}^\ell + \alpha_j^\ell p_{ij} x_{ij}^\ell + \sum_{j'>j} \alpha_{j'}^\ell p_{ij'} x_{ij'}^\ell$$

For the first set of terms, we have that $\alpha_{j'}^\ell \leq \frac{1}{k-\ell+1}$ since ℓ has not yet been removed from A. Thus these terms can be bounded above by $\frac{\beta}{k-\ell+1}$, since ℓ's fractional makespan was bounded above by β before job j. For the middle term, we use a similar observation for α_j^ℓ but also apply the definition of $S(j, \beta)$ to conclude that $\alpha_j^\ell > 0 \implies p_{ij} x_{ij}^\ell \leq \beta$. Thus we get a bound of $\frac{\beta}{k-\ell+1}$ for the middle term. For the final set of terms, we have $\alpha_{j'}^\ell = 0$ for all $j' > j$ since we have removed ℓ from A at this point, and so these contribute nothing. Combining these bounds gives the lemma. □

Proof of Theorem 4.2. By (4), we have that the load of machine i is at most $\sum_{\ell \in [k]} C_i^\ell$. Applying Lemma 4.4 we have $C_i^\ell \leq \frac{2\beta}{k-\ell+1}$. Thus machine i's load is at most $\sum_{\ell \in [k]} \frac{2\beta}{k-\ell+1} = 2H_k\beta = O(\log k) \cdot \beta$, where $H_k = \sum_{\ell=1}^k \frac{1}{k}$ is the k'th harmonic number. Now if we run the algorithm with $\beta = \min_{\ell \in [k]} \max_i \sum_j p_{ij} x_{ij}^\ell$, then there is some solution $\ell^* \in [k]$ which has a fractional makespan of β, so it never gets removed from A or $A(j, \beta)$ in Algorithm 3. In this case Algorithm 3 never fails, completing the proof of the theorem. □

4.3 Learning k Predicted Weight Vectors

We now turn to the question of showing how to learn k different predicted weight vectors $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$. Recall that there is an unknown distribution $\mathcal{D}$ over sets of n jobs from which we receive independent samples $J_1, J_2, \ldots, J_S$. Our goal is to show that we can efficiently learn (in terms of sample complexity) k predicted weight vectors to minimize the expected minimum error. Let $w^*(J)$ be the correct weight vector for instance J and let $\eta(w, w') = \max_{i \in [m]} \max(\frac{w_i}{w_i'}, \frac{w_i'}{w_i})$ be the error between a pair of weight vectors. We have the following result.

Theorem 4.5. *Let $\mathcal{D}$ be an unknown distribution over restricted assignment instances on n jobs and let $w^*(J)$ be a set of good weights for instance J. Given S independent samples from $\mathcal{D}$, there is a polynomial time algorithm that outputs k weight vectors $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$ such that $\mathbb{E}_{J \sim \mathcal{D}} \left[\min_{\ell \in [k]} \log(\eta(\hat{w}^\ell, w^*(J))\right] \leq O(1) \cdot \min_{w^1, w^2, \ldots, w^k} \mathbb{E}\left[\min_{\ell \in [k]} \log(\eta(w^\ell, w^*(J))\right] + \epsilon$ with probability $1 - \delta$, where $S = \text{poly}(m, k, \frac{1}{\epsilon}, \frac{1}{\delta})$*

Prior work [28] has observed that we can take the weights to be from the set $\mathcal{W}(R) = \{w \in \mathbb{R}_+^m \mid \forall i \in [m], \exists \alpha \in [R], \text{ s.t. } w_i = (1 + \epsilon/m)^\alpha\}$. Moreover, it suffices to take $R = \Theta(m^2 \log(m))$ in order to guarantee that for any instance there exists some set of weights in $\mathcal{W}(R)$ such that the associated fractional assignment yields an $O(1)$-approximate solution. Since this set is finite, with only $m^{O(m^2 \log m)}$ members, it follows that the pseudo-dimension of any class of functions parameterized by the weights is bounded by $\log(|\mathcal{W}(R)|) = O(m^2 \log m)$, and so we get polynomial sample complexity. Thus in order to prove Theorem 4.5, it suffices to give a polynomial time ERM algorithm for this problem. As hinted at in the statement of Theorem 4.5, we will be working with the logarithms of the errors. Working in this space allows us to carry out a reduction to k-median clustering.

For any pair of weight vectors $w, w' \in \mathbb{R}_+^m$, recall that we define the error between them to be $\eta(w, w') = \max_{i \in [m]} \max(\frac{w_i}{w_i'}, \frac{w_i'}{w_i})$. Note that $\eta(w, w') \geq 1$ with equality if and only if $w = w'$ and that $\eta(w, w') = \eta(w', w)$. The main lemma is that defining $d(w, w') := \log(\eta(w, w'))$ satisfies the triangle inequality, and thus forms a metric on $\mathbb{R}_+^m$ due to the aforementioned observations.

Lemma 4.6. *Let $\eta : \mathbb{R}_+^m \times \mathbb{R}_+^m$ and $d : \mathbb{R}_+^m \times \mathbb{R}_+^m$ be defined as above. Then $(\mathbb{R}_+^m, d)$ forms a metric space.*

Proof. It is easy to see that $d(w, w') \geq 0$ with equality if and only if $w = w'$ and that $d(w, w') = d(w', w)$. Thus we just need to show that the triangle inequality holds, i.e. that for all w, w', w'' we have $d(w, w'') \leq d(w, w') + d(w', w'')$. This will hold as a result of the following claim. For all w, w', w''. we have:

$$\eta(w, w'') \leq \eta(w, w') \cdot \eta(w', w''). \tag{5}$$

Now the triangle inequality follows since

$$d(w, w') = \log(\eta(w, w'')) \leq \log(\eta(w, w')) + \log(\eta(w', w'')) = d(w, w') + d(w', w'').$$

To prove the claim we have the following:

$$\begin{aligned}
\eta(w, w'') &= \max_i \left\{ \max\left(\frac{w_i}{w_i''}, \frac{w_i''}{w_i} \right) \right\} \\
&= \max_i \left\{ \max\left(\frac{w_i}{w_i'} \frac{w_i'}{w_i''}, \frac{w_i''}{w_i'} \frac{w_i'}{w_i} \right) \right\} \\
&\leq \max_i \left\{ \max\left(\frac{w_i}{w_i'}, \frac{w_i'}{w_i''} \right) \cdot \max\left(\frac{w_i''}{w_i'}, \frac{w_i'}{w_i} \right) \right\} \\
&\leq \max_i \left\{ \max\left(\frac{w_i}{w_i'}, \frac{w_i'}{w_i''} \right) \right\} \cdot \max_i \left\{ \max\left(\frac{w_i''}{w_i'}, \frac{w_i'}{w_i} \right) \right\} \\
&= \eta(w, w') \cdot \eta(w', w'').
\end{aligned}$$

The two inequalities above follow from the next two claims below. □

Claim 4.7. *For all $a, b, c > 0$ we have $\max(\frac{a}{c}, \frac{c}{a}) \leq \max(\frac{a}{b}, \frac{b}{a}) \cdot \max(\frac{b}{c}, \frac{c}{b})$*

Proof. Without loss of generality, we may assume that $a \geq c$. Now we have several cases depending on the value of b. For the first case, lets consider when $a \geq b \geq c > 0$. In this case, the left hand side evaluates to $\frac{a}{c}$ while the right hand side also evaluates to $\frac{a}{c}$, so the inequality is valid in this case.

For the next case, consider when $b \geq a \geq c > 0$. In this case, the left hand side is still $\frac{a}{c}$, while the right hand side evaluates to $\frac{b^2}{ac} \geq \frac{ab}{ac} \geq \frac{a}{c}$. Thus the inequality is valid.

In the final case, we have $a \geq c \geq b > 0$. The left hand side is $\frac{a}{c}$, while the right hand side evaluates to $\frac{ac}{b^2} \geq \frac{ac}{c^2} = \frac{a}{c}$, completing the proof. □

Claim 4.8. *Let $u, v \in \mathbb{R}_+^m$, then we have $\max_i(u_i v_i) \leq (\max_i u_i)(\max_i v_i)$*

Proof. Suppose for contradiction that this isn't the case, i.e. $\max_i(u_i v_i) > (\max_i u_i)(\max_i v_i)$. Now let $i^* = \arg\max_i(u_i v_i)$. Then we have

$$u_{i^*} v_{i^*} > (\max_i u_i)(\max_i v_i) \geq u_{i^*} v_{i^*}$$

which is the desired contradiction. □

Proof of Theorem 4.5. The sample complexity follows from the discussion above and Theorem 2.3 which shows that the pseudo-dimension of the error function is at most $O(m^2 \log(m))$. Given S samples $J_1, J_2, \ldots, J_S$ from $\mathcal{D}$, we want to solve the corresponding ERM instance. To do this we set up the following k-median instance to compute the predicted weights $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$. First we compute $w^s = w^*(J_s)$ for each $s \in [S]$, then we set the distance between w^s and $w^{s'}$ to be the distance function $d(w^s, w^{s'})$ defined above. At a loss of a factor of 2, we can take each $\hat{w}^\ell$ to be in $\{w^s\}_{s=1}^S$ by Lemma 2.6. Thus we can apply an $O(1)$-approximate k-median algorithm (e.g. the one due to [29]) to get the predicted weight vectors $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$ in polynomial time. □

5 Scheduling with Predicted Permutations

In this problem there are n jobs, indexed by $1, 2, \ldots, n$, to be scheduled on a single machine. We assume that they are all available at time 0. Job j has size p_j and needs to get processed for p_j time units to complete. If all job sizes are known a priori, Shortest Job First (or equivalently Shortest Remaining Time First), which processes jobs in non-decreasing order of their size, is known to be optimal for minimizing total completion time. We assume that the true value of p_j is unknown and is revealed only when the job completes, i.e. the *non-clairvoyant* setting. In the non-clairvoyant setting, it is known that Round-Robin (which processes all alive jobs equally) is 2-competitive and that this is the best competitive ratio one can hope for [37].

We study this basic scheduling problem assuming certain predictions are available for use. Following the recent work by Lindermayr and Megow [32], we will assume that we are given k orderings/sequences as prediction, $\{\sigma_\ell\}_{\ell \in [k]}$. Each σ_ℓ is a permutation of $J := [n]$. Intuitively, it suggests an ordering in which jobs should be processed. This prediction is inspired by the aforementioned Shortest Job First (SJF) as an optimal schedule can be described as an ordering of jobs, specifically increasing order of job sizes.

For each σ_ℓ, its error is measured as $\eta(J, \sigma_\ell) := \textsc{cost}(J, \sigma_\ell) - \mathrm{OPT}(J)$, where $\textsc{cost}(J, \sigma_\ell)$ denotes the objective of the schedule where jobs are processed in the order of σ_ℓ and $\mathrm{OPT}(J)$ denotes the optimal objective value. We may drop J from notation when it is clear from the context.

As observed in [32], the error can be expressed as $\eta(J, \sigma_\ell) = \sum_{i<j \in J} I^\ell_{i,j} \cdot |p_i - p_j|$, where $I^\ell_{i,j}$ is an indicator variable for 'inversion' that has value 1 if and only if σ_ℓ predicts the pairwise ordering of i and j incorrectly. That is, if $p_i < p_j$, then the optimal schedule would process i before j; here $I^\ell_{i,j} = 1$ iff $i \succ_{\sigma_\ell} j$.

As discussed in [32], this error measure satisfies two desired properties, monotonicity and Lipschitzness, which were formalized in [25].

Our main result is the following.

Theorem 5.1. *Consider a constant $\epsilon > 0$. Suppose that for any $S \subseteq J$ with $|S| = \Theta(\frac{1}{\epsilon^4}(\log\log n + \log k + \log(1/\epsilon)))$, we have $\mathrm{OPT}(S) \leq c\epsilon \cdot \mathrm{OPT}(J)$ for some small absolute constant c. Then, there exists a randomized algorithm that yields a schedule whose expected total completion time is at most $(1+\epsilon)\mathrm{OPT} + (1+\epsilon)\frac{1}{\epsilon^5}\eta(J, \sigma_\ell)$ for all $\ell \in [k]$.*

As a corollary, by running our algorithm with $1-\epsilon$ processing speed and simultaneously running Round-Robin with the remaining ϵ of the speed, the cost increases by a factor of at most $\frac{1}{1-\epsilon}$ while the resulting hybrid algorithm is $2/\epsilon$-competitive.[3]

5.1 Algorithm

To make our presentation more transparent we will first round job sizes. Formally, we choose ρ uniformly at random from $[0, 1)$. Then, round up each job j's size to the closest number of the form $(1+\epsilon)^{\rho+t}$ for some integer t. Then, we scale down all job sizes by $(1+\epsilon)^\rho$ factor. We will present our algorithm and analysis assuming that every job has a size equal to a power of $(1+\epsilon)$. Later we will show how to remove this assumption without increasing our algorithm's objective by more than $1+\epsilon$ factor in expectation (Section 5.4).

We first present the following algorithm that achieves Theorem 5.1 with $|S| = \Theta(\frac{1}{\epsilon^4}(\log n + \log k))$. The improved bound claimed in the theorem needs minor tweaks of the algorithm and analysis and they are deferred to the supplementary material.

Our algorithm runs in rounds. Let J_r be the jobs that complete in round $r \geq 1$. For any subset S of rounds, $J_S := \cup_{r \in S} J_r$. For example, $J_{\leq r} := J_1 \cup \ldots \cup J_r$. Let $n_r := |J_{\geq r}| = n - |J_{<r}|$ denote the number of alive jobs at the beginning of round r.

[3] This hybrid algorithm is essentially the preferential time sharing [25, 32, 39]. Formally, we run our algorithm ignoring RR's processing and also run RR ignoring our algorithm; this can be done by a simple simulation. Thus, we construct two schedules concurrently and each job completes at the time when it does in either schedule. This type of algorithms was first used in [39].

Fix the beginning of round r. The algorithm processes the job in the following way for this round. If $n_r \leq \frac{1}{\epsilon^4}(\log n + \log k)$, we run Round-Robin to complete all the remaining jobs, $J_{\geq r}$. This is the last round and it is denoted as round $L+1$. Otherwise, we do the following Steps 1-4:

Step 1. Estimating ϵ-percentile. Roughly speaking, the goal is to estimate the ϵ-percentile of job sizes among the remaining jobs. For a job $j \in J_{\geq r}$, define its rank among $J_{\geq r}$ as the number of jobs no smaller than j in $J_{\geq r}$ breaking ties in an arbitrary yet fixed way. Ideally, we would like to estimate the size of job of rank ϵn_r, but do so only approximately.

The algorithm will find $\tilde{q}_r$ that is the size of a job whose rank lies in $[\epsilon(1-\epsilon)n_r, \epsilon(1+\epsilon)n_r]$. To handle the case that there are many jobs of the same size $\tilde{q}_r$, we estimate y_r the number of jobs no bigger than $\tilde{q}_r$; let $\tilde{y}_r$ denote our estimate of y_r. We will show how we can do these estimations without spending much time by sampling some jobs and partially processing them in Round-Robin manner (the proof of the following lemma can be found in Section 5.1.1.)

Lemma 5.2. *W.h.p. the algorithm can construct estimates $\tilde{q}_r$ and $\tilde{y}_r$ in time at most $O(\tilde{q}_r \frac{1}{\epsilon^2} \log n)$ such that there is a job of size $\tilde{q}_r$ whose rank lies in $[\epsilon(1-\epsilon)n_r, \epsilon(1+\epsilon)n_r]$ and $|\tilde{y}_r - y_r| \leq \epsilon^2 n_r$.*

Step 2. Determining Good and Bad Sequences. Let σ_ℓ^r denote σ_ℓ with all jobs completed in the previous rounds removed and with the relative ordering of the remaining jobs fixed. Let $\sigma_{\ell,\epsilon}^r$ denote the first $\tilde{y}_r$ jobs in the ordering. We say a job j is big if $p_j > \tilde{q}_r$; middle if $p_j = \tilde{q}_r$; small otherwise. Using sampling and partial processing we will approximately distinguish good and bad sequences. Informally σ_ℓ^r is good if $\sigma_{\ell,\epsilon}^r$ has few big jobs and bad if it does many big jobs. The proof of the following lemma can be found in Section 5.1.2.

Lemma 5.3. *For all $\ell \in [k]$, we can label sequence σ_ℓ^r either good or bad in time at most $O(\tilde{q}_r \frac{1}{\epsilon^2}(\log n + \log k))$ that satisfies the following with high probability: If it is good, $\sigma_{\ell,\epsilon}^r$ has at most $3\epsilon^2 n_r$ big jobs; otherwise $\sigma_{\ell,\epsilon}^r$ has at least $\epsilon^2 n_r$ big jobs.*

Step 3. Job Processing. If all sequences are bad, then we process all jobs, each up to $\tilde{q}_r$ units in an arbitrary order. Otherwise, we process the first $\tilde{y}_r$ jobs in an arbitrary good sequence, in an arbitrary order, each up to $\tilde{q}_r$ units.

Step 4. Updating Sequences. The jobs completed in this round drop from the sequences but the remaining jobs' relative ordering remains fixed in each (sub-)sequence. For simplicity, we assume that partially processed jobs were never processed—this is without loss of generality as this assumption only increases our schedule's objective.

5.1.1 Subprocedure: Step 1

We detail the first step of the algorithm. Our goal is to prove Lemma 5.2. To obtain the desired estimates, we take a sample S_r of size $\frac{1}{\epsilon^2}\log n$ from $J_{\geq r}$ with replacement. The algorithm processes the sampled jobs using Round-Robin until we complete $\epsilon|S_r|$ jobs.[4] Note that there could be multiple jobs that complete at the same time. This is particularly possible because we assumed that jobs have sizes equal to a power of $1+\epsilon$. In this case, we break ties in an arbitrary but fixed order. Let $\tilde{q}_r$ be the size of the job that completes $\epsilon|S_r|$th. If m_r is the number of jobs in S_r we completed, we estimate $\tilde{y}_r = \frac{m_r}{|S_r|} n_r$.

Let B_1 is the bad event that $\tilde{q}_r$ has a rank that doesn't belong to $[a_1 := \epsilon(1-\epsilon)n_r, a_2 := \epsilon(1+\epsilon)n_r]$. Let X_1 be the number of jobs sampled that have rank at most a_1. Similarly let X_2 be the number of jobs sampled that have rank at most a_2. Note that $\neg B_1$ occurs if $X_1 \leq \epsilon|S_r| \leq X_2$. Thus, we have $\Pr[B_1] \leq \Pr[X_1 > \epsilon|S_r|] + \Pr[X_2 < \epsilon|S_r|]$. Note that $X_1 = Binomial(a_1/n_r, |S_r|)$.

We use the following well-known Hoeffding's Inequality.

Theorem 5.4. *Let $Z_1, \cdots, Z_T$ be independent random variables such that $Z_i \in [0,1]$ for all $i \in [T]$. Let $\bar{Z} = \frac{1}{T}(Z_1 + \cdots + Z_T)$. We have $P(|\bar{Z} - E[\bar{Z}]| \geq \delta) \leq 2e^{-2T\delta^2}$ where $\delta \geq 0$.*

[4] If a job is sampled more than once, we can pretend that multiples copies of the same job are distinct and simulate round robin [25].

By Theorem 5.4 we have,

$$\Pr[X_1 < \epsilon|S_r|] = \Pr[X_1/|S_r| - a_1/n_r < \epsilon - a_1/n_r = \epsilon^2] \leq 2\exp(-2|S_r|\epsilon^2) = 2/n^2.$$

Similarly, we can show that

$$\Pr[X_2 > \epsilon|S_r|] = 2/n^2$$

Thus, we conclude that $\Pr[B_1] \leq 4/n^2$.

We consider the second bad event $|\tilde{y}_r - y_r| > \epsilon^2 n_r$, which we call B_2. Assume that $\tilde{q}_r$ is the size of a fixed job of rank in $[a_1, a_2]$. If m' is the actual number of jobs of size $\tilde{q}_r$ in $J_{\geq r}$, $m_\ell = Binomial(m'/n_r, |S_r|)$. As before, using Theorem 5.4 we can show that $\Pr[B_2] \leq 4/n^2$. Thus, we can avoid both bad events simultaneously with probability $1 - 8/n^2$.

Since we process each job in S_r up to at most $\tilde{q}_r$ units, the total time we spend for estimation in Step 1 is at most $\tilde{q}_r \frac{1}{\epsilon^2} \log n$. This completes the proof.

5.1.2 Subprocedure: Step 2

We now elaborate on the second step. Our goal is to prove Lemma 5.3. To decide whether each sequence is good or not, as before we take a uniform sample from S'_r of size $\frac{1}{\epsilon^2}(\log n + \log k)$ with replacement. By processing the jobs each up to $\tilde{q}_r$ units, we can decide the number of jobs in S'_r of size no bigger than $\tilde{q}_r$. If the number is c, we estimate $\tilde{y}_r = \frac{c}{|S'_r|} n_r$. The proof follows the same lines as Lemma 5.2 and is omitted. The only minor difference is that we need to test if each of the k sequences is good or not, and to avoid the bad events for all k sequences simultaneously, we ensure the probability that the bad events occur for a sequence is at most $O(\frac{1}{kn^2})$. This is why we use a bigger sample size than in Step 1.

5.2 Analysis of the Algorithm's Performance

Let σ^* be an arbitrary sequence against which we want to be competitive. The analysis proceeds in rounds. Let b_r be the start time of round r; by definition $b_1 = 0$. For the sake of analysis we decompose our algorithm's cost as follows:

$$\sum_{r\in[L]} \left(\sum_{j\in J_r} (C_j - b_r) + T_r \cdot |J_{>r}| \right) + 2\mathrm{OPT}(J_{L+1}),$$

where T_r is the total time spent in round r. To see this, observe that $\sum_{j\in J_r}(C_j - b_r)$ is the total completion time of jobs that complete in round r, ignoring their waiting time before b_r. Each job $j \in J_r$'s waiting time in the previous rounds is exactly $\sum_{r'\in[L-1]} T_{r'}$. The total waiting time over all of the jobs during rounds where they are not completed is $\sum_{r'\in[L-1]} T_{r'} \cdot |J_{>r'}|$. The last term $2\mathrm{OPT}(J_{L+1})$ follows from the fact that we use Round-Robin to finish the last few jobs in the final round $L+1$.

To upper bound $A_r := \left(\sum_{j\in J_r}(C_j - b_r) + T_r \cdot |J_{>r}|\right)$ by $\textsc{cost}(\sigma^*)$ we also decompose $\textsc{cost}(\sigma^*)$ as

$$\sum_{r\in[L]} \left(\textsc{cost}(\sigma^{*r+1}) - \textsc{cost}(\sigma^{*r})\right) + \textsc{cost}(\sigma^{*L+1})$$

Recall that we complete jobs J_r in round r and σ evolves from σ^r to σ^{r+1} by dropping J_r from the sequence σ^r. Thus, it decreases the cost of following σ.

Since we use round-robin to finish the remaining jobs J_{L+1} in the final round and round-robin is 2-competitive, our algorithm incurs cost at most $2\mathrm{OPT}(J_{L+1})$. If the assumption in Theorem 5.1 holds, this is at most $2\epsilon\mathrm{OPT}$.

Let's take a close look at $\left(\textsc{cost}(\sigma^{*r+1}) - \textsc{cost}(\sigma^{*r})\right)$. This is equivalent to the following: For each pair $i \in J_r$ and $j \in J_{>r}$, σ^* has an error $|p_i - p_j|$ if and only if it creates an inversion. This quantity can be charged generously. Let's call this aggregate error η_r. But $\mathrm{OPT}(J_r) + \sum_{i\in J_r, j\in J_{>r}} \min\{p_i, p_j\}$ should be charged sparingly. Let's call this part of optimum quantity OPT_r. Note that we are using the following:

Lemma 5.5. $\mathrm{OPT} = \sum_{r\in[L]} \left(\mathrm{OPT}(J_r) + \sum_{i\in J_r, j\in J_{>r}} \min\{p_i, p_j\}\right) + \mathrm{OPT}(J^{L+1})$

Proof. We know that the optimal schedule is Shortest-Job-First. Thus we have the following. Here we assume jobs are indexed in an arbitrary but fixed manner.

$$\begin{aligned}
\mathrm{OPT} &= \sum_{j\in J} p_j + \sum_{i\in J, p_i<p_j} p_i \\
&= \sum_{i\in J, j\in J, i\geq j} \min\{p_i, p_j\} \\
&= \sum_{r\in[L+1]} \left(\sum_{i\in J_r, j\in J_r\, i\geq j} \min\{p_i, p_j\} + \sum_{i\in J_r, j\in J_{>r}} \min\{p_i, p_j\} \right) \\
&= \sum_{r\in[L+1]} \left(\mathrm{OPT}(J_r) + \sum_{i\in J_r, j\in J_{>r}} \min\{p_i, p_j\} \right) \\
&= \sum_{r\in[L]} \left(\mathrm{OPT}(J_r) + \sum_{i\in J_r, j\in J_{>r}} \min\{p_i, p_j\} \right) + \mathrm{OPT}(J^{L+1}) \quad \square
\end{aligned}$$

Using the previous lemma, if we bound A_r by $(1+O(\epsilon))\mathrm{OPT}_r + O(\frac{1}{\epsilon^5})\eta_r$ we will have Theorem 5.1 by scaling ϵ appropriately. Proving this for all $r \in [L]$ is the remaining goal.

We consider two cases.

All Sequences Are Bad. In this case we complete all jobs of size at most $\tilde{q}_r$. Further $\sigma^*_{r,\epsilon}$ has at least $\epsilon^2 n_r$ big jobs. This implies that there are at least $\epsilon^2 n_r$ big jobs that do not appear in $\sigma^*_{r,\epsilon}$. So, for each pair of such a big job and a non-big job, σ^*_r creates an inversion and has an error of at least $\epsilon\tilde{q}_r$, which contributes to η_r. Thus, $\eta_r \geq \epsilon\tilde{q}_r\epsilon^4 n_r^2$.

Now we want to upper bound A_r. Since the delay due to estimation is at most $\tilde{q}_r \frac{2}{\epsilon^2}(\log n + \log k)$ and all jobs are processed up to $\tilde{q}_r$ units, we have $A_r \leq (\tilde{q}_r \frac{2}{\epsilon^2}(\log n + \log k)) * n_r + \tilde{q}_r (n_r)^2 \leq (2\epsilon + 1)\tilde{q}_r (n_r)^2$. Thus, $A_r \leq O(1)\frac{1}{\epsilon^5}\eta_r$.

Some Sequences Are Good. We will bound the expected cost of the algorithm for round r when there is a good sequence. The bad event B_1 occurs with very small probability as shown in the analysis of Step 1. The contribution to the expected cost is negligible if the bad event occurs. Due to this, we may assume that the event $\neg B_1$ occurs.

Say the algorithm processes the first $\tilde{y}_r$ jobs in a good sequence σ_r. By Lemmas 5.2 and 5.3, J_r processes all small and middle jobs in σ^r_ϵ. Additionally, the algorithm may process up to $3\epsilon^2 n_r$ big jobs without completing them.

The total time it takes to process the big jobs is $4\epsilon^2 n_r \cdot \tilde{q}_r$ and up to n_r jobs wait on them. The contribution to the objective of all jobs waiting while these are processed is at most $4\epsilon^2 n_r^2 \cdot \tilde{q}_r$. We call this the wasteful delay due to processing big jobs. We show that this delay is only $O(\epsilon)$ fraction of $A_r + A_{r+1}$.

Consider $A_r + A_{r+1}$. By definition of the algorithm, at least $\frac{1}{2}\epsilon n_r$ jobs of size at least $\tilde{q}_r$ are completed during rounds r and $r+1$. If less than $\frac{1}{2}\epsilon n_r$ middle or big jobs complete in round r, we can show that $n_{r+1} \geq (1-2\epsilon)n_r$. Then, we observe that $J_{\geq r+1}$ must have at most $4\epsilon^2 n_r$ small jobs. This is because σ^r_ϵ includes at most $3\epsilon^2 n_r$ big jobs and it includes $\tilde{y}_r$ jobs. Since y_r is the number of small and middle jobs and $|y_r - \tilde{y}_r| \leq \epsilon^2 n_r$, σ^r_ϵ must include all non-big jobs, except up to $4\epsilon^2 n_r$. Thus, most jobs completing in round $r+1$ are middle or big and we can show that the number of such jobs completing in round $r+1$ is at least $\frac{1}{2}\epsilon n_r$. Therefore, at least $\frac{1}{2}n_r$ jobs will wait on the first $\frac{1}{2}\epsilon n_r$ jobs of size at least $\tilde{q}_r$ completed. This implies, $A_r + A_{r+1} \geq \frac{1}{4}\tilde{q}_r \epsilon n_r^2$. This is at least a $\Theta(\frac{1}{\epsilon})$ factor larger than the wasteful delay.

Note that the delay due to processing big jobs and as well as the time spent computing the estimates (used in Lemma 5.2 and 5.3) is at most $n_r \cdot 6\epsilon^2 n_r \tilde{q}_r \leq O(\epsilon) \cdot (A_r + A_{r+1})$. In the following, we will bound the cost without incorporating these costs. Factoring in these two costs will increase the bound by at most $1/(1 - O(\epsilon))$ factor.

Thus, we only need to consider small and middle jobs that complete. For the sake of analysis, we can consider the worst case scenario where we first process middle jobs and then small jobs. We will bound A_r by $\mathrm{OPT}(J_r)$ (i.e. without charging to η_r). In this case, A_r/OPT_r is maximized when when all small jobs have size 0. We will assume this in the following. Let m be the number of mid sized jobs we complete. For brevity, assume that the number of small jobs is at most ϵn_r although the actual bound is $\epsilon(1+\epsilon)n_r$. Further, we assume that $m(m+1) \approx m^2$ as we are willing to lose $(1+\epsilon)$ factor in our bound. Let $n = n_r$ for notational convenience. Then, we have $A_r = m^2\tilde{q}_r/2 + m\tilde{q}_r\epsilon n + (n(1-\epsilon) - m)m\tilde{q}_r = m^2\tilde{q}_r/2 + (n-m)m\tilde{q}_r$. In contrast, $\mathrm{OPT}_r = m^2\tilde{q}_r/2 + ((1-\epsilon)n - m)m\tilde{q}_r$. The ratio of the two quantities is $\frac{n-m/2}{(1-\epsilon)n-m/2}$ where $m \leq n$. Therefore, in the worst case $\frac{1/2}{1/2-\epsilon} \leq (1+3\epsilon)$. Thus, A_r can be bounded by $(1+3\epsilon)\mathrm{OPT}_r$.

5.3 Improved Guarantees

One can show that at least $(\epsilon - 4\epsilon^2)$ fraction of jobs complete in every round assuming no bad events occur: if all sequences are bad then all non-big jobs complete, which means at least $(\epsilon - \epsilon^2)n_r$ jobs complete due to Lemma 5.2. Otherwise, any good sequence σ_r^ℓ has at least $(\epsilon - \epsilon^2 - 3\epsilon^2)n_r$ non-big jobs in $\sigma_{r,\epsilon}^\ell$ due to Lemmas 5.2 and 5.3 and they all complete if σ_r^ℓ is chosen. Thus there are at most $O(\frac{1}{\epsilon}\log n)$ rounds and there are at most $O(\frac{k}{\epsilon}\log n)$ bad events, as described in Lemmas 5.2 and 5.3, to be avoided.

Suppose we run round robin all the time using ϵ of the speed, so even in the worst case we have a schedule that is $2/\epsilon$-competitive against the optimum and therefore against the best prediction as well. This will only increase our upper bound by $1/(1-\epsilon)$ factor. Then, we can afford to have bad events with higher probabilities.

Specifically, we can reduce the sample size in Steps 1 and 2 to $s := \Theta(\frac{1}{\epsilon^2}\log(k(1/\epsilon^3)\log n))$ to avoid all bad events with probability at least $1 - \epsilon^2$; so if any bad events occur, at most an extra $(2/\epsilon) \cdot \epsilon^2\mathrm{OPT}$ cost occurs in expectation. Then, we can show that we can do steps 1-4 as long as $n_r = \Omega(\frac{1}{\epsilon^4}(\log k + \log\log n + \log\frac{1}{\epsilon}))$.

Then the delay due to estimation is at most $2s\tilde{q}_r n_r$ in Steps 1 and 2. We want to ensure that this is at most $O(\epsilon)$ fraction of $A_r + A_{r+1}$. Recall that we showed $A_r + A_{r+1} \geq \frac{1}{4}\tilde{q}_r\epsilon n_r^2$. Thus, if we have $\frac{1}{4}\epsilon^2\tilde{q}_r n_r^2 \geq 2s\tilde{q}_r n_r$, we will have the desired goal. This implies that all the delay due to estimation can be charged to $O(\epsilon)$ of the algorithm's objective as long as $n_r \geq 8\frac{1}{\epsilon^2}s$. This is why we switch to round-robin if $n_r \leq 8\frac{1}{\epsilon^2}s$.

5.4 Removing the Simplifying Assumption on Job Sizes

Recall that we chose ρ uniformly at random from $[0,1)$ and rounded up each j's size to the closest number of the form $(1+\epsilon)^{\rho+t}$ for some integer t. Although we then scaled down all job sizes by $(1+\epsilon)^\rho$, we assume that we didn't do it. This assumption is wlog as all the bounds remain the same regardless of uniform scaling of job sizes.

Let η^ℓ be the prediction error of σ^ℓ. Let $\bar{\eta}^\ell$ be the error after rounding up job sizes. Let $\bar{p}_j$ be j's size after rounding. Note that i) $p_j \leq \bar{p}_j \leq (1+\epsilon)p_j$; ii) if $p_i \leq p_j$, then $\bar{p}_i \leq \bar{p}_j$. The second property implies jobs relative ordering is preserved.

In the following we drop k for notational convenience. We have

$$\eta := \sum_{i \neq j \in J} \mathbf{1}(p_i < p_j) \cdot \mathbf{1}(i \succ_{\sigma^*} j) \cdot |p_i - p_j|$$

$$\bar{\eta} := \sum_{i \neq j \in J} \mathbf{1}(p_i < p_j) \cdot \mathbf{1}(i \succ_{\sigma^*} j) \cdot |\bar{p}_i - \bar{p}_j|$$

Lemma 5.6. $\mathbb{E}\bar{\eta} = \Theta(1) \cdot \eta$.

Proof. Thanks to linearity of expectation it suffices to show that $\mathbb{E}|\bar{p}_i - \bar{p}_j| = \Theta(1) \cdot |p_i - p_j|$ for every pair of jobs i and j.

Assume $p_i > p_j$ wlog. Scale both job sizes for convenience, so $p_j = 1$. Let $p_i/p_j = (1+\epsilon)^\delta$.

Case 1. $\delta \geq 1$. In this case we have $\bar{p}_i > \bar{p}_j$ almost surely. Using the fact that rounding up increases job sizes by at most $1+\epsilon$ factor, we can show that $(1/3)|p_i - p_j| \leq |\bar{p}_i - \bar{p}_j| \leq (1+\epsilon)^2|p_i - p_j|$.

Case 2. $\delta \in (1/2, 1]$. In this case we can show that $|p_i - p_j| = \Theta(\epsilon)$ and $\mathbb{E}|\bar{p}_i - \bar{p}_j| = \Theta(\epsilon)$.

Case 3. $\delta \in (0, 1/2)$. We have $|p_i - p_j| = (1+\epsilon)^\delta - 1$ and $\mathbb{E}|\bar{p}_i - \bar{p}_j| = \int_{\rho=0}^{\delta} \epsilon(1+\epsilon)^\rho d\rho = \frac{\epsilon}{\ln(1+\epsilon)}((1+\epsilon)^\delta - 1)$. Thus, the ratio of the two is $\frac{\epsilon}{\ln(1+\epsilon)} = \Theta(1)$ for small $\epsilon > 0$. □

What we showed was the following. For any $\bar{\eta}^\ell$, $\mathbb{E}A \leq (1+\epsilon)(\mathrm{OPT} + O(1)\frac{1}{\epsilon^5}\bar{\eta}^\ell) + 2(1+\epsilon)\mathrm{OPT}(J_{K+1})$. By taking expectation over randomized rounding, we have $\mathbb{E}A \leq (1+\epsilon)^2(\mathrm{OPT} + O(1)\frac{1}{\epsilon^5}\eta^\ell) + 2(1+\epsilon)^2\mathrm{OPT}(J_{K+1})$. By scaling ϵ appropriately, we obtain the same bound claimed in Theorem 5.1.

5.5 Learning k Predicted Permutations

Now we show that learning the best k permutations has polynomial sample complexity.

Theorem 5.7. *Let $\mathcal{D}$ be an unknown distribution of instances on n jobs. Given S independent samples from $\mathcal{D}$, there is an algorithm that outputs k permutations $\hat{\sigma}_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_k$ such that $\mathbb{E}_{J\sim\mathcal{D}}\left[\min_{\ell\in[k]} \eta(J, \hat{\sigma}_\ell)\right] \leq \min_{\sigma_1,\sigma_2,\ldots,\sigma_k} \mathbb{E}_{J\sim\mathcal{D}}\left[\min_{\ell\in[k]} \eta(J, \sigma_\ell)\right] + \epsilon$ with probability $1-\delta$, where $S = \mathrm{poly}(n, k, \frac{1}{\epsilon}, \frac{1}{\delta})$.*

Proof. The algorithm is basic ERM, and the polynomial sample complexity follows from Theorem 2.3 and Theorem 20 in Lindermayr and Megow [32]. □

6 Conclusion

Despite the explosive recent work in algorithms with predictions, almost all of this work has assumed only a single prediction. In this paper we study algorithms with *multiple* machine-learned predictions, rather than just one. We study three different problems that have been well-studied in the single prediction setting but not with multiple predictions: faster algorithms for min-cost bipartite matching using learned duals, online load balancing with learned machine weights, and non-clairvoyant scheduling with order predictions. For all of the problems we design algorithms that can utilize multiple predictions, and show sample complexity bounds for learning the best set of k predictions. Demonstrating the effectiveness of our algorithms (and the broader use of multiple predictions) empirically is an interesting direction for further work.

Surprisingly, we have shown that in some cases, using multiple predictions is essentially "free." For instance, in the case of min-cost perfect matching examining $k = O(\sqrt{n})$ predictions takes the same amount of time as one round of the Hungarian algorithm, but the number of rounds is determined by the quality of the *best* prediction. In contrast, for load balancing, using k predictions always incurs an $O(\log k)$ cost, so using a constant number of predictions may be best. More generally, studying this trade-off between the cost and the benefit of multiple predictions for other problems remains an interesting and challenging open problem.

Acknowledgments and Disclosure of Funding

Michael Dinitz was supported in part by NSF grant CCF-1909111. Sungjin Im was supported in part by NSF grants CCF-1617653, CCF-1844939 and CCF-2121745. Thomas Lavastida and Benjamin Moseley were supported in part by NSF grants CCF-1824303, CCF-1845146, CCF-2121744 and CMMI-1938909. Benjamin Moseley was additionally supported in part by a Google Research Award, an Infor Research Award, and a Carnegie Bosch Junior Faculty Chair.

References

[1] Anders Aamand, Piotr Indyk, and Ali Vakilian. (learned) frequency estimation algorithms under zipfian distribution. *arXiv preprint arXiv:1908.05198*, 2019.

[2] Priyank Agrawal, Eric Balkanski, Vasilis Gkatzelis, Tingting Ou, and Xizhi Tan. Learning-augmented mechanism design: Leveraging predictions for facility location, 2022. URL https://arxiv.org/abs/2204.01120.

[3] Shipra Agrawal, Morteza Zadimoghaddam, and Vahab S. Mirrokni. Proportional allocation: Simple, distributed, and diverse matching with high entropy. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 99–108. PMLR, 2018. URL http://proceedings.mlr.press/v80/agrawal18b.html.

[4] Sara Ahmadian, Hossein Esfandiari, Vahab Mirrokni, and Binghui Peng. *Robust Load Balancing with Machine Learned Advice*, pages 20–34. 2022. doi: 10.1137/1.9781611977073.2. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977073.2.

[5] Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ML predictions for online algorithms. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 303–313. PMLR, 2020. URL http://proceedings.mlr.press/v119/anand20a.html.

[6] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. A regression approach to learning-augmented online algorithms. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 30504–30517, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/ffeed84c7cb1ae7bf4ec4bd78275bb98-Abstract.html.

[7] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. Online algorithms with multiple predictions. In *Proceedings of the 39th International Conference on Machine Learning, ICML 2022*, 2022.

[8] Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

[9] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/5a378f8490c8d6af8647a753812f6e31-Abstract.html.

[10] Yossi Azar, Andrei Z. Broder, and Mark S. Manasse. On-line choice of on-line algorithms. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 432–440. ACM/SIAM, 1993. URL http://dl.acm.org/citation.cfm?id=313559.313847.

[11] Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1070–1080. ACM, 2021. doi: 10.1145/3406325.3451023. URL https://doi.org/10.1145/3406325.3451023.

[12] Yossi Azar, Stefano Leonardi, and Noam Touitou. Distortion-oblivious algorithms for minimizing flow time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 252–274. SIAM, 2022. doi: 10.1137/1.9781611977073.13. URL https://doi.org/10.1137/1.9781611977073.13.

[13] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Generalization in portfolio-based algorithm selection. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12225–12232. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17451.

[14] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e834cb114d33f729dbc9c7fb0c6bb607-Abstract.html.

[15] Aditya Bhaskara, Ashok Cutkosky, Ravi Kumar, and Manish Purohit. Online linear optimization with many hints. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9530–9539. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/6c250b592dc94d4de38a79db4d2b18f2-Paper.pdf.

[16] Rajen Bhatt and Abhinav Dhall. Skin segmentation dataset, uci machine learning repository, 2012.

[17] Justin Y. Chen, Sandeep Silwal, Ali Vakilian, and Fred Zhang. Faster fundamental graph algorithms via learned predictions. *CoRR*, abs/2204.12055, 2022. doi: 10.48550/arXiv.2204.12055. URL https://doi.org/10.48550/arXiv.2204.12055.

[18] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10393–10406, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5616060fb8ae85d93f334e7267307664-Abstract.html.

[19] Elbert Du, Franklyn Wang, and Michael Mitzenmacher. Putting the "learning" into learning-augmented algorithms for frequency estimation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2860–2869. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/du21d.html.

[20] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[21] Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. *CoRR*, abs/2011.06726, 2020. URL https://arxiv.org/abs/2011.06726.

[22] Vasilis Gkatzelis, Kostas Kollias, Alkmini Sgouritsa, and Xizhi Tan. Improved price of anarchy via predictions, 2022. URL https://arxiv.org/abs/2205.04252.

[23] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019. URL http://proceedings.mlr.press/v97/gollapudi19a.html.

[24] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *7th International Conference on Learning Representations*, 2019.

[25] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 285–294. ACM, 2021. doi: 10.1145/3409964.3461790. URL https://doi.org/10.1145/3409964.3461790.

[26] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.ICALP.2020.69. URL https://doi.org/10.4230/LIPIcs.ICALP.2020.69.

[27] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020. doi: 10.1137/1.9781611975994.114. URL https://doi.org/10.1137/1.9781611975994.114.

[28] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.ESA.2021.59. URL https://doi.org/10.4230/LIPIcs.ESA.2021.59.

[29] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016. doi: 10.1137/130938645. URL https://doi.org/10.1137/130938645.

[30] Shi Li and Jiayi Xian. Online unrelated machine load balancing with predictions revisited. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 6523–6532. PMLR, 2021. URL http://proceedings.mlr.press/v139/li21w.html.

[31] Alexander Lindermayr and Nicole Megow. Algorithms with predictions. https://algorithms-with-predictions.github.io/, 2022.

[32] Alexander Lindermayr and Nicole Megow. Non-clairvoyant scheduling with predictions revisited. *CoRR*, abs/2202.10199, 2022. URL https://arxiv.org/abs/2202.10199.

[33] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3302–3311, 2018.

[34] Andrés Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1856–1864, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[35] Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with Predictions*, page 646–662. Cambridge University Press, 2021. doi: 10.1017/9781108637435.037.

[36] Jamie H Morgenstern and Tim Roughgarden. On the pseudo-dimension of nearly optimal auctions. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 136–144. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5766-on-the-pseudo-dimension-of-nearly-optimal-auctions.pdf.

[37] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

[38] David Pollard. *Convergence of stochastic processes*. Springer Science & Business Media, 2012.

[39] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[40] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Symposium on Discrete Algorithms (SODA)*, 2020.

[41] Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/5bd844f11fa520d54fa5edec06ea2507-Abstract.html`.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Experiments

We now consider a preliminary empirical investigation of the algorithm proposed in Section 3 for min-cost perfect matching with a portfolio of predicted dual solutions. For this, we modify the experimental setup utilized by Dinitz et al. [18] to evaluate the case of single prediction for this problem. We use a training set which is a mixture between three distinct types of instances, and show that by using more than one prediction we see an overall improvement.

Dataset	Shuttle	KDD	Skin [16]
# of Points (n)	43500	98,942	100,000
# of Features (d)	10	38	4

Table 1: Datasets used in preliminary experiments.

Experiment Setup and Datasets: These experiments were performed on a machine with a 6 core, 3.7 GHz AMD Ryzen 5 5600x CPU and 16 GB of RAM. The algorithms were implemented in Python and the code is available at `https://github.com/tlavastida/PredictionPortfolios`.

To construct bipartite matching instances we adopt the same technique as Dinitz et al. [18] for Euclidean data sets. At a high level, their technique takes in a dataset X of points in $\mathbb{R}^d$ and a parameter $n \in \mathbb{N}$, and outputs a distribution $\mathcal{D}(X, n)$ over dense instances of min-cost perfect matching with n nodes on each side. Sampling from the distribution $\mathcal{D}(X, n)$ can be done efficiently. We consider three datasets (Shuttle, KDD, and Skin) from the UCI Machine Learning Repository [20] that were also considered in [18], see Table 1 for details.

Instead of considering each dataset (and its corresponding distribution) separately, we consider them together as a mixture model. For each dataset, we consider a sub-sample X of 20,000 points and we set $n = 150$ (so $2n = 300$ nodes per instance) in order to construct each distribution $\mathcal{D}(X, n)$. We then sample 20 instances from each distribution and consider these 60 instances together as our training set (i.e. our learning algorithm doesn't know which dataset each instance was derived from). For testing, we sample an additional 10 instances from each dataset.

Results: To evaluate our approach, we vary the number of predicted dual solutions (k) learned from one (baseline) to five and also compare to the standard Hungarian method (referred to as "No Learning"). Following [18] our evaluation metrics are running time and the number of iterations of the Hungarian algorithm.

Given that the dataset has three distinct clusters by construction, we expect the average number of iterations of the Hungarian algorithm to decrease as k grows from 1 to 3 and then stay stable. We also expect the running time to decrease as k grows from 1 to 3, and then increase as k grows from 3 to 5, as the cost of the projection step grows linearly with k.

In Table2 we have divided our results on the test instances by dataset, so that different scales don't obscure the results.

Dataset	Shuttle		KDD		Skin [16]	
k	# Iterations	Time (s)	# Iterations	Time (s)	# Iterations	Time (s)
No Learning	149.0	0.929	304.4	1.97	63.1	0.396
1	77.9	0.601	149.4	1.09	68.7	0.472
2	59.0	0.433	144.0	1.08	306.7	2.690
3	42.4	0.350	144.0	1.09	38.9	0.318
4	42.4	0.373	130.3	1.05	38.9	0.342
5	42.4	0.394	136.1	1.15	38.9	0.357

Table 2: Average iteration count and average running time across each data set and value of k.

Observe that for the Shuffle dataset the process proceeds exactly as we had predicted – the number of iterations drops as k grows from 0 to 3 and then stays constant, the wall clock time drops as well and then starts increasing.

For KDD and Shuffle the situation is more interesting. For KDD we don't see the nice inflection point at $k = 3$. We conjecture that the KDD dataset itself is diverse and induces many subclusters

for its family of matching instances, thus increasing the number of clusters keeps on improving the performance.

For the Skin dataset, we observe an anomaly at $k = 2$. We conjecture that this is due to the clustering in the learning stage allocating both predictions to the KDD and Shuffle datasets, essentially ignoring the Skin dataset and thus giving extremely poor performance. In other words, the gain from allocating the "extra" prediction to the other datasets was enough to outweigh the cost to the Skin data. This aligns well with our conjecture that the KDD data itself has many subclusters.

Overall, however, we see that the empirical evaluation supports our conclusion that there are performance gains to be had when judiciously using multiple advice models.