Non-Clairvoyant Scheduling with Predictions

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

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

Mahshid Montazer Qaem
University of California
Merced, CA
mmontazerqaem@ucmerced.edu

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

ABSTRACT

In the single-machine *non-clairvoyant* scheduling problem, the goal is to minimize the total completion time of jobs whose processing times are *unknown* a priori. We revisit this well-studied problem and consider the question of how to effectively use (possibly erroneous) predictions of the processing times. We study this question from ground zero by first asking what constitutes a good prediction; we then propose a new measure to gauge prediction quality and design scheduling algorithms with strong guarantees under this measure. Our approach to derive a prediction error measure based on natural desiderata could find applications for other online problems.

CCS CONCEPTS

• **Theory of computation → Approximation algorithms analysis**; **Online algorithms**.

KEYWORDS

Scheduling; non-clairvoyance; competitive ratio; prediction

ACM Reference Format:
Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. 2021. Non-Clairvoyant Scheduling with Predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3409964.3461790

1 INTRODUCTION

Non-clairvoyance, where the scheduler is not aware of the exact processing times of a job a priori, is a highly desired property in the design of scheduling algorithms. Due to its myriad practical applications, non-clairvoyant scheduling has been extensively studied in various settings in the scheduling literature [12, 14, 25]. With no access to the processing times (i.e., job sizes), non-clairvoyant algorithms inherently suffer from worse performance guarantees than the corresponding clairvoyant algorithms. For example, in the most basic version of non-clairvoyant scheduling, we have a set of jobs that need to be scheduled on a single machine with the goal of minimizing the total completion time of all jobs. The job sizes are unknown to the algorithm and only become known *after* the job has completed. In this setting, the *Round-Robin* algorithm [23] that divides the machine equally among all incomplete jobs is 2-competitive, and this is known to be optimal. In contrast, in the clairvoyant setting where job sizes are known a priori, the *Shortest Job First* (SJF) algorithm that schedules jobs in non-decreasing order of their sizes is known to be optimal.

Practitioners often face scheduling problems that lie somewhere in between clairvoyant and non-clairvoyant settings. While it is almost impossible to know the exact job sizes, rather than assuming non-clairvoyance, it is possible to estimate job sizes based on their features using a predictor [2, 20, 24]; such an estimation can be error-prone. Can one use the (possibly erroneous) predicted job sizes to improve the performance of scheduling algorithms?

Augmenting traditional algorithms with machine-learned predictions is a fascinating and newly emerging line of work. In particular, this paradigm is applicable to online algorithms, which typically focus on obtaining worst-case guarantees against uncertain future inputs and thus settle for pessimistic bounds. Recent works have shown that, using predictions (that may be incorrect), one can provably improve the guarantees of traditional online algorithms for caching [13, 19, 26], ski-rental [3, 10, 16], scheduling [7, 16, 22], load balancing [17], secretary problem [6], metrical task systems [5], set cover [8], flow and matching [18], and bin packing [4], etc.

In this paper we continue the study of learning-augmented algorithms for single-machine non-clairvoyant scheduling. This problem, where an algorithm has access to predictions of each job size, was first investigated in [16]. Without making any assumptions on the prediction quality, they design a non-clairvoyant algorithm that satisfies two important properties, namely, consistency and robustness. *Consistency* means that the guarantees of the algorithm improve with good predictions; in particular, the algorithm obtains a competitive ratio better than 2 if the predictions are good. *Robustness* ensures that the algorithm gracefully handles bad predictions, i.e., even if the predictions are adversarially bad, the competitive ratio stays bounded. For any $\lambda \in (0, 1)$, they design an algorithm that guarantees robustness of $\frac{2}{1-\lambda}$ and consistency of $\frac{1}{\lambda}$. [1]

SPAA '21, July 6–8, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8070-6/21/07…$15.00
https://doi.org/10.1145/3409964.3461790

[1] Here, α-robustness and β-consistency mean that the algorithm's cost is at most α times the optimum for all inputs but improves to at most β factor when the prediction coincides with the actual input. See Definition 2.

1.1 The Need for a New Error Measure

Although [16] demonstrates an appealing trade-off between consistency and robustness for non-clairvoyant scheduling, a closer look reveals some brittleness of the result. Here, we discuss the issue at a high-level and delve in more detail in the next section when we formally define the problem and the old/new error notions.

The main issue stems from the total completion time objective. Since this objective measures the total waiting time of all jobs, a shorter job could delay more jobs. In fact, different jobs can have different effect on how much they delay other jobs. The objective is thus *neither linear nor quadratic* in the job sizes.[2]

In [16], it is assumed that the algorithm has a prediction $\hat{p}_j$ of each job size p_j. The quality of the prediction is the sum of the prediction errors of individual jobs, i.e., $\ell_1(p, \hat{p}) = \sum_j |\hat{p}_j - p_j|$. Intuitively, such a linear error measure is incompatible with the completion time objective and may not distinguish good predictions vs poor predictions; in fact, small perturbations in the predictions can result in large changes to the optimal solution. Consequently, the results in [16] are forced to be pessimistic and have a weak dependence on the error term. In particular, they show that scheduling the jobs in non-decreasing order of their predicted sizes (SPJF) yields a competitive ratio of at most $\textsc{opt} + (n-1) \cdot \ell_1(p, \hat{p})$ and is tight, where $\textsc{opt}$ is the optimum solution and n the number of jobs.

We examine the $\ell_1(\cdot, \cdot)$ error measure and show that it violates a natural and desirable Lipschitz-like property for the total completion time objective. This prompts the search for a new error measure based on two desiderata (see Section 2.2). Our new error measure better captures the sensitive nature of the objective and allows us to obtain an algorithm with competitive ratio at most $(1+\epsilon)\textsc{opt} + O_\epsilon(1) \cdot \nu(p, \hat{p})$ where $\nu(\cdot, \cdot)$ is the measure we propose.

In practice, job sizes are predicted using black-box machine-learned models that utilize various features of the jobs (e.g., history) and may be expensive to train. While it is impossible to precisely define the goodness of a prediction, intuitively, an effective error measure should neither tag bad predictions as good nor miss out on predictions that could improve the objective.

1.2 Our Contributions

Under the new notion of error (denoted ν), we give the following results, stated informally below. We assume all jobs are available for scheduling from time 0. Let $\textsc{opt}$ be the optimum objective.

(1) We obtain a non-clairvoyant algorithm that is $O(1)$-robust (with no dependency on ϵ) and $(1+\epsilon)$-consistent for any $\epsilon > 0$ w.h.p., if no subset of $O(\frac{1}{\epsilon^3} \log n)$ jobs dominates the objective. (Theorem 32 and Corollary 33)

(2) We obtain a non-clairvoyant algorithm that is $O(\frac{1}{\epsilon})$-robust and $(1+\epsilon)$-consistent in expectation for any sufficiently small $\epsilon > 0$. More precisely, the cost of the algorithm is at most $(1+\epsilon)\textsc{opt} + O(\frac{1}{\epsilon^3} \log \frac{1}{\epsilon})\nu$. (Theorem 34)

In contrast, [16] obtains an algorithm that is $O(\frac{1}{\epsilon})$-robust and whose cost is at most $(1+\epsilon)\textsc{opt} + (1+\epsilon)(n-1) \cdot \ell_1(p, \hat{p})$. Since our error measure satisfies $\ell_1(p, \hat{p}) \le \nu \le n \cdot \ell_1(p, \hat{p})$, our algorithm never has an asymptotically worse dependence on the prediction quality and is often sharper.

(3) We show that for any sufficiently small $\epsilon, \gamma > 0$, no deterministic algorithm can have a smaller objective than $(1+\epsilon)\textsc{opt} + O(1/\epsilon^{1-\gamma})\nu$. (Theorem 36)

We now discuss the high-level ideas. The main challenge is how to determine if a prediction is reliable or not *before* completing all jobs. If the predictions are somewhat reliable, we can more or less follow them; otherwise, we will essentially have to rely on non-clairvoyant algorithms such as Round-Robin. Therefore, we repeatedly take a small sample of jobs over the course of the algorithm and partially process them. Informally, we estimate the median remaining size of jobs, and estimate the prediction error considering job sizes up to the estimated median. Unfortunately, this estimation is not free since we have to partially process the sampled jobs and it can delay all the existing jobs. Therefore, we are forced to stop sampling once there are too few jobs left. Depending on how long we sample, we obtain the first and second results.

Due to the dynamic nature of our algorithm, the analysis turns out to be considerably non-trivial. In a nutshell, we never see the true error until we finish a job. Nevertheless, we still have to decide whether to follow the predictions. The mismatch between partial errors we perceive and the actual errors makes it challenging to charge our algorithm's cost to the optimum and the error; special care is needed throughout the analysis to avoid overcharging. We note that unlike our algorithm, [16] uses a static algorithm that linearly combines following the predictions and Round-Robin.

To summarize, our work demonstrates that it is possible to find quality solutions for a bigger class of predictions by using a more refined measure and it could lead to new algorithmic techniques.

1.3 Other Related Work

Designing learning-augmented algorithms falls into the new beyond-worst-case algorithm design paradigm [27]. Starting with the work of Kraska et al. [15] on using ML predictions to speed up indexing, there have been many efforts to leverage ML predictions to better handle common instances that are found in practice. In addition to the aforementioned works, there also exist works on frequency counting [1, 9, 11] and membership testing [21, 28].

For single machine scheduling in the clairvoyant setting, Shortest Remaining Processing Time (SRPT) is known to be optimal for minimizing the total completion time; it is in fact optimal for minimizing the total flow/response time[3]. If all jobs arrive at time 0, SJF coincides with SRPT. In the non-clairvoyant setting, when jobs have different arrival times, no algorithm is $O(1)$-competitive for minimizing the total flow time, but Round-Robin is known to be $O(1)$-competitive when compared to the optimum schedule running on a machine with speed less than $1/2 - \epsilon$, for any $\epsilon > 0$. For a survey on online scheduling algorithms, see [25].

[2] For a concrete example, consider n jobs that have unit sizes with sufficiently small perturbations. The derivative of the objective is n with respect to the smallest job; yet it is 1 with respect to the largest job.

[3] In the setting where job j has a release time r_j, the flow time of a job is defined as $C_j - r_j$ where C_j is the completion time of job j in the schedule. If all jobs are available at time 0, then the flow time coincides with completion time.

1.4 Roadmap

In Section 2 we formally define our non-clairvoyant scheduling problem. In the same section we continue to discuss what desiderata constitute a good measure of prediction error and propose a new measure meeting the desiderata. We also discuss other—both existing and candidate—measures and show that they fail to satisfy the desiderata. We present our algorithm in Section 3 and its analysis in Section 4. The lower bounds are presented in Section 5. All missing proofs and analysis are in the full version.

2 FORMULATION AND BASIC PROPERTIES

2.1 Non-Clairvoyant Scheduling

Let J denote a set of n jobs. In the classical single-machine *non-clairvoyant scheduling* setting, each job $j \in J$ has an unknown *size* or *processing time* p_j. The processing time is known only after the job is complete. A job j *completes* when it has received p_j amount of processing time, and we denote j's *completion time* as C_j. A job may be preempted at any time and resumed at a later time without any cost. Our goal is to find a schedule that completes all jobs and minimizes the *total completion time* of all jobs, i.e., $\sum_{j\in J} C_j$. In the *clairvoyant* case, an algorithm knows the p_j's in advance.

Definition 1 (Competitive Ratio)**.** *Let $\mathcal{I}$ denote the set of all instances of the non-clairvoyant scheduling problem. Let $\textsc{cost}_{\mathcal{A}}(I)$ be the total completion time of the schedule obtained by a non-clairvoyant algorithm $\mathcal{A}$ and $\textsc{opt}(I)$ be the cost of the optimum (clairvoyant) algorithm on instance I. $\mathcal{A}$ is said to be* c-competitive *if*

$$\max_{I\in\mathcal{I}} \frac{\textsc{cost}_{\mathcal{A}}(I)}{\textsc{opt}(I)} \le c.$$

In the clairvoyant case, it is well-known that the *Shortest Job First (SJF)*[4] scheduling algorithm minimizes the total completion time. In the non-clairvoyant case, the *Round-Robin*[5] algorithm achieves a competitive ratio of 2, which is known to be optimal [23].

For any subset $Z \subseteq J$ of jobs, we let $\textsc{opt}(\{x_j\}_{j\in Z})$ denote the minimum objective to complete all jobs in Z when each job $j \in Z$ has size x_j and is known to the algorithm, i.e., $\textsc{opt}$ is the completion time of SJF using x_j as the size of job j. Here, we can think of $\textsc{opt}$ as a function that takes as input a multiset of non-negative job sizes and returns the minimum objective to complete all jobs with the job sizes in the set. (Note that this is well-defined as SJF is oblivious to job identities.) If x_j is j's true size, i.e., p_j, for notational convenience, we use $\textsc{opt}(Z) := \textsc{opt}(\{p_j\}_{j\in Z})$; in particular, $\textsc{opt} := \textsc{opt}(J)$.

We consider the *learning-augmented scheduling* problem where the algorithm has access to *predictions* for each job size; let $\hat{p}_j$ denote the predicted size of job j. We emphasize that we make no assumptions regarding the validity of the predictions and they may even be adversarial. As in the usual non-clairvoyant scheduling setup, the true processing size p_j of job j is revealed only *after* the job has received p_j amount of processing time. In the learning-augmented setting, the competitive ratio of an algorithm $\mathcal{A}$ is a function of the prediction error. Our goal is to design an algorithm that satisfies the dual notions of robustness and consistency.

[4]Schedule the jobs in non-decreasing order of job sizes.
[5]Process all incomplete jobs equally at each time.

Definition 2 (Robustness and Consistency)**.** *Let $\mathcal{I}$ be the set of all instances of the learning-augmented non-clairvoyant scheduling problem*[6]*. The* robustness *of an algorithm $\mathcal{A}$ is the worst-case ratio of the algorithm's cost to the cost of the optimal solution independent of the quality of the predictions. On the other hand, the* consistency *of an algorithm $\mathcal{A}$ is the worst-case ratio when restricted to instances where the predictions are all correct, i.e., $\hat{p}_j = p_j, \forall j \in J$.*

$$\textit{Robustness}(\mathcal{A}) = \max_{I\in\mathcal{I}} \frac{\textsc{cost}_{\mathcal{A}}(I)}{\textsc{opt}(I)},$$

$$\textit{Consistency}(\mathcal{A}) = \max_{\substack{I\in\mathcal{I}\\ \hat{p}_j=p_j,\forall j}} \frac{\textsc{cost}_{\mathcal{A}}(I)}{\textsc{opt}(I)}.$$

2.1.1 Properties of $\textsc{opt}$*.* The following fact is well-known and follows from the definition of $\textsc{opt}$, i.e., SJF.

Proposition 3 ([23])**.** $\textsc{opt}(\{x_j\}_{j\in J}) = \sum_{j\in J} x_j + \sum_{i\neq j\in J} \min\{x_i, x_j\} \le \sum_{(i,j)\in J\times J} \min\{x_i, x_j\}$.

The following properties are simple consequences of SJF.

Proposition 4. *Let J denote an arbitrary set of jobs and $\{x_j\}_{j\in J}$ and $\{y_j\}_{j\in J}$ be two sets of non-negative job sizes. Then,*

1. *If $x_j \ge y_j$ for all $j \in J$, then $\textsc{opt}(\{x_j\}_{j\in J}) \ge \textsc{opt}(\{y_j\}_{j\in J})$.*
2. *For any subset $Z \subseteq J$, $\textsc{opt}(\{x_j\}_{j\in J}) \ge \textsc{opt}(\{x_j\}_{j\in Z})$.*
3. *$\textsc{opt}(\{x_j + y_j\}_{j\in J}) \ge \textsc{opt}(\{x_j\}_{j\in J}) + \textsc{opt}(\{y_j\}_{j\in J})$.*
4. *Let $X_1, \ldots, X_L$ be a partition of J, i.e., $J = \bigcup_{\ell\in[L]} X_\ell$ and $X_\ell \cap X_{\ell'} = \emptyset$ for $\ell \neq \ell'$, then we have*

$$\sum_{\ell\in[L]} \textsc{opt}(\{x_j\}_{j\in X_l}) \le \textsc{opt}(\{x_j\}_{j\in J}) \le L \cdot \sum_{\ell\in[L]} \textsc{opt}(\{x_j\}_{j\in X_l}).$$

2.2 Prediction Error

A key question in the design of algorithms with predictions is how to define the prediction error, i.e., how to quantify the quality of predictions. While this definition can be problem-dependent, it must be *algorithm-independent*. For the non-clairvoyant scheduling problem, before we dive into a definition, we identify two desirable necessary properties that we want of any such definition. Let $\textsc{err}(\{p_j\}_{j\in J}, \{\hat{p}_j\}_{j\in J})$ denote the prediction error for an instance with true sizes $\{p_j\}$ and predicted job sizes $\{\hat{p}_j\}$; note that an algorithm knows the $\hat{p}_j$'s but not the p_j's.

The first property is monotonicity, i.e., if more job sizes predictions are correct, then the error must decrease. Monotonicity is natural as better predictions are expected to decrease the error.

Property 5 (Monotonicity)**.** *For any $I \subseteq J$,* $\textsc{err}(\{p_j\}_{j\in J}, \{\hat{p}_j\}_{j\in J\setminus I} \cup \{p_i\}_{i\in I}) \le \textsc{err}(\{p_j\}_{j\in J}, \{\hat{p}_j\}_{j\in J})$.

The second property is a Lipschitz-like condition that states that a prediction $\{\hat{p}_j\}_{j\in J}$ is said to be good (as measured by $\textsc{err}(\cdot,\cdot)$) only if the optimal solution of the predicted instance is close to the true optimal solution. Indeed if the optimal solution of a predicted instance differs significantly from true optimal solution, i.e., $|\textsc{opt}(\{\hat{p}_j\}_{j\in J}) - \textsc{opt}(\{p_j\}_{j\in J})|$ is large, then the property requires that a good error measure assigns a large error to such predictions. Intuitively, this property allows us to effectively distinguish between good and bad predictions.

[6]An instance here is specified by both the predicted job sizes and the true job sizes.

Property 6 (Lipschitzness). $|\textsc{opt}(\{\hat{p}_j\}_{j\in J}) - \textsc{opt}(\{p_j\}_{j\in J})| \le \textsc{err}(\{p_j\}_{j\in J}, \{\hat{p}_j\}_{j\in J})$.

A natural way to define the prediction error is to define it as the ℓ_1 norm between the predicted and the true job sizes, i.e., $\ell_1(p,\hat{p}) = \textsc{err}(\{p_j\}_{j\in J}, \{\hat{p}_j\}_{j\in J}) = \sum_{j\in J} |p_j - \hat{p}_j|$, as was done in [16]. While this error definition satisfies monotonicity, it is not Lipschitz. Indeed, consider the following simple problem instance. Let $\epsilon > 0$ be a constant. The true job sizes are given by $p_1 = 1+\epsilon$ and $p_j = 1, \forall j \in J \setminus \{1\}$. Let $\hat{p}$ be a set of predicted job sizes given by $\hat{p}_1 = 1 + 3\epsilon$ and $\hat{p}_j = 1, \forall j \in J \setminus \{1\}$. Similarly, let $\hat{q}$ be another set of predicted job sizes given by $\hat{q}_1 = 1 - \epsilon$ and $\hat{q}_j = 1, \forall j \in J \setminus \{1\}$. By construction, $\ell_1(p,\hat{p}) = 2\epsilon = \ell_1(p,\hat{q})$. However, by the nature of the total completion time objective, there is a significant difference in the quality of the predictions in these two instances. Formally, $\textsc{opt}(\{\hat{p}_j\}_{j\in J}) - \textsc{opt}(\{p_j\}_{j\in J}) = 2\epsilon$ whereas $\textsc{opt}(\{p_j\}_{j\in J}) - \textsc{opt}(\{\hat{q}_j\}_{j\in J}) = (n+1)\cdot\epsilon \gg \ell_1(p,\hat{q})$. Intuitively, the lack of Lipschitzness causes the $\ell_1(\cdot,\cdot)$ error metric to not be able to distinguish between $\{\hat{p}\}$ and $\{\hat{q}\}$ predictions although $\{\hat{p}\}$ is arguably a much better prediction for this instance.

On the other hand, to satisfy the Lipschitz property, one can consider simply defining the prediction error as $\textsc{err}(\{p_j\}_{j\in J}, \{\hat{p}_j\}_{j\in J}) = |\textsc{opt}(\{\hat{p}_j\}_{j\in J}) - \textsc{opt}(\{p_j\}_{j\in J})|$. Unfortunately, this may not be monotone. Indeed, consider a simple instance where the predictions are a reassignment of the true job sizes to the jobs, i.e., the job sizes are predicted correctly but the job identities are permuted. In this case, we have $|\textsc{opt}(\{\hat{p}_j\}_{j\in J}) - \textsc{opt}(\{p_j\}_{j\in J})| = 0$. However, an improvement to any of the predictions will only result in a different optimum, and hence a non-zero error. In other words, this definition does not satisfy monotonicity.

These examples motivate a new definition of prediction error.

Definition 7 (Prediction Error). *For any instance of the non-clairvoyant scheduling problem with predictions where each job $j \in J$ has a true size p_j and a predicted size $\hat{p}_j$, the* prediction error *is defined as:*

$$\nu(J;\{p_j\},\{\hat{p}_j\}) := \textsc{err}(\{p_j\}_{j\in J}, \{\hat{p}_j\}_{j\in J}) = \textsc{opt}\Big(\{\hat{p}_j\}_{j\in J_o} \cup \{p_j\}_{j\in J_u}\Big) - \textsc{opt}\Big(\{p_j\}_{j\in J_o} \cup \{\hat{p}_j\}_{j\in J_u}\Big),$$

where $J_o = \{j \in J \mid \hat{p}_j > p_j\}$, $J_u = \{j \in J \mid \hat{p}_j \le p_j\}$ denote the set of jobs whose sizes are overestimated and underestimated respectively.

Intuitively, the above definition follows ensures

$$\nu \ge \textsc{opt}\Big(\{\hat{p}_j\}_{j\in J_o} \cup \{p_j\}_{j\in J_u}\Big) - \textsc{opt}\Big(\{p_j\}_{j\in J_o} \cup \{p_j\}_{j\in J_u}\Big),$$

by pretending that all underestimated job sizes were predicted correctly. Similarly, we also want

$$\nu \ge \textsc{opt}\Big(\{p_j\}_{j\in J_o} \cup \{p_j\}_{j\in J_u}\Big) - \textsc{opt}\Big(\{p_j\}_{j\in J_o} \cup \{\hat{p}_j\}_{j\in J_u}\Big).$$

Our error measure follows by adding the RHS of these inequalities.

It is easy to see that this definition, besides being symmetric and non-negative, also satisfies both monotonicity and the Lipschitz property. While this may not be the unique such definition, it is simple. Further, we are *not* aware of any other error measures, including those used in the previous work [8, 16], that satisfy the two desired properties. For more details, see Section 2.2.2.

Proposition 8. *The error measure given in Definition 7 satisfies both Monotonicity and Lipschitzness.*

When the scheduling instance is clear from context, we drop the arguments and let $\nu = \nu(J;\{p_j\},\{\hat{p}_j\})$. Note that in case all the predicted job sizes are overestimates (or underestimates) of the true sizes, then we have $\nu(J;\{p_j\},\{\hat{p}_j\}) = |\textsc{opt}(\{\hat{p}_j\}_{j\in J}) - \textsc{opt}(\{p_j\}_{j\in J})|$.

2.2.1 Surrogate Error. For the sake of analysis, we define a surrogate (prediction) error where we measure the error for overestimated and underestimated jobs separately. The surrogate error lower bounds the prediction error in Definition 7 and will be more convenient for our analysis. While it does not satisfy Lipschitzness, nevertheless, it will turn out to be a useful tool for analysis.

Definition 9 (Surrogate Error). *For any set $Z \subseteq J$ of jobs, where each job $j \in Z$ has a true size x_j and a predicted size $\hat{x}_j$, the* surrogate error *is defined as:*

$$\eta(Z;\{x_j\},\{\hat{x}_j\}) := \Big(\textsc{opt}(\{\hat{x}_j\}_{j\in Z_o}) - \textsc{opt}(\{x_j\}_{j\in Z_o})\Big) + \Big(\textsc{opt}(\{x_j\}_{j\in Z_u}) - \textsc{opt}(\{\hat{x}_j\}_{j\in Z_u})\Big),$$

where $Z_o = \{j \in Z \mid \hat{x}_j > x_j\}, Z_u = \{j \in Z \mid \hat{x}_j \le x_j\}$ denote the set of jobs whose sizes are overestimated and underestimated respectively.

Again, when the scheduling instance is clear from context, we drop the arguments and let $\eta = \eta(J;\{p_j\},\{\hat{p}_j\})$. We first show that the surrogate error can be used to lower bound the prediction error.

Proposition 10. *For any set $Z \subseteq J$ of jobs where each job $j \in Z$ has true size x_j and predicted size $\hat{x}_j$, $\nu(Z;\{x_j\},\{\hat{x}_j\}) \ge \eta(Z;\{x_j\},\{\hat{x}_j\})$.*

A key advantage of the surrogate error η is that it is easier to decompose as opposed to ν. As our analysis carefully charges our algorithm's cost in each round to the error and the optimum, decomposability will be very useful to avoid overcharging.

Proposition 11 (Superadditivity of Surrogate Error). *For any set $Z \subseteq J$ of jobs, any set of true and predicted job sizes $\{(x_j,\hat{x}_j)\}_{j\in Z}$ and any partition of Z into two disjoint subsets Z_1 and Z_2, we have $\eta(Z;\{x_j\},\{\hat{x}_j\}) \ge \eta(Z_1;\{x_j\},\{\hat{x}_j\}) + \eta(Z_2;\{x_j\},\{\hat{x}_j\})$.*

2.2.2 Comparisons with Other Error Measures. We compare our new error measure with others, including those in [8, 16]. First, we observe that our error measure is always lower bounded by the $\ell_1(p,\hat{p})$ error utilized by [16] but is at most a factor of n larger.

Proposition 12. *For any set $\{p_j\}_{j\in J}$ and $\{\hat{p}_j\}_{j\in J}$ of true and predicted job sizes, we have*

$$\ell_1(p,\hat{p}) \le \nu(J;\{p_j\},\{\hat{p}_j\}) \le n\cdot\ell_1(p,\hat{p})$$

Thus our error measure lends itself to asymptotically stronger algorithmic guarantees than the $\ell_1(p,\hat{p})$ measure. In [16], the cost of their algorithm is shown to be bounded by $(1+\epsilon)\textsc{opt} + (1+\epsilon)\cdot(n-1)\ell_1(p,\hat{p})$. As we show in the following sections, we obtain an algorithm whose cost is bounded by $(1+\epsilon)\textsc{opt} + O_\epsilon(1)\cdot\nu$. By Proposition 12, our bound is asymptotically never worse than that in [16], and can often be sharper.

Next, we discuss the error measure used by Bamas et al. [8] in their primal-dual framework. Their measure, which we call η_{BMS}, can be defined as the cost of SPJF[7] minus the optimum. It is easy to see that η_{BMS} is neither monotone nor Lipschitz. This is because

[7] Shortest Predicted Job First (SPJF) is the algorithm that blindly follows the predictions.

SPJF yields an optimal schedule as long as jobs have the same order both in their actual sizes and estimated sizes, i.e., $p_j \le p_i$ if and only if $\hat{p}_j \le \hat{p}_i$. Further, it is hard to compare our error measure to η_{BMS} as the latter does not directly factor in estimated job sizes but measures the cost for running an algorithm (that is based on the prediction) on the actual input. However, the following example shows that η_{BMS} can be excessively large even for estimating just one job size: The true job sizes are given by $p_j = 1\ \forall j \in J \setminus \{n\}$ and $p_n = n^2$. All jobs sizes are predicted correctly, except job n, where $\hat{p}_n = 0$. Then, $\nu = 1{+}2{+}\cdots{+}n{-}1{+}(n{+}n^2){-}(1{+}2{+}\cdots{+}n{-}1) = n{+}n^2$, whileas $\eta_{BMS} \ge n^2 \cdot n - (1 + 2 + \cdots + n - 1 + n + n^2) = \Omega(n^3)$, Here, $n^2 \cdot n$ comes from the fact that job n completes first under SPJF.

Finally, we note that the error measure cannot be oblivious to job identities. For example, consider the Earth Mover's distance between the true job sizes and the estimated job sizes. That is, find a mincost matching between two multi-sets $\{p_j\}_{j\in J}$ and $\{\hat{p}_i\}_{i\in J}$ where matching p_i to $\hat{p}_j$ incurs cost $|p_i - \hat{p}_j|$. However, it is easy see that such measures have zero error when the two multi-sets are identical yet the predictions are incorrect for individual jobs.

3 ALGORITHM

In this section we present our algorithm for scheduling with predictions. Our algorithm runs in rounds. To formalize, we need to set up some notation. We let J_k be the set of unfinished (alive) jobs at the beginning of round k, where $k \ge 1$. Let $n_k := |J_k|$. Let $q_{k,j}$ be the amount of processing done on job j in round k. We define

- $p_{k,j} = p_j - \sum_{w=1}^{k-1} q_{w,j}$: the *true* remaining size of j at the beginning of round k.
- $\hat{p}_{k,j} = \max\{\hat{p}_j - \sum_{w=1}^{k-1} q_{w,j}, 0\}$: the *predicted* remaining size of j at the beginning of round k.

Note that if a job j has been processed by more than its predicted size $\hat{p}_j$ in the previous rounds before k, we have $\hat{p}_{k,j} = 0$.

Our algorithm employs two subprocedures in each round to estimate the median m_k of the true remaining size of jobs in J_k and the magnitude of the error in the round. We first present the subprocedures and then present our main algorithm.

3.1 Median Estimation

We first present the method to estimate the median m_k of the true remaining size of jobs in J_k. To streamline the analysis, we will assume that all remaining sizes are distinct, which can be achieved almost surely by adding small random perturbations to the initial job sizes. Let $\tilde{m}_k$ denote our estimate of the true median m_k. Recall that Round-Robin processes all alive jobs equally at each time.

Algorithm 1 Median-Estimator(J_k, δ, n)

1: Let S be a uniform random sample, with replacement, of size $\frac{\ln 2n}{\delta^2}$ from J_k.
2: Run Round-Robin on S until half of the jobs in S complete; let j_k be the job that completed the last.
3: Return $\tilde{m}_k = p_{k,j_k}$.

Algorithm 1 takes a sample S of the remaining jobs and returns as $\tilde{m}_k$ the median of the jobs in S in terms of their remaining size. This can be done by completing half of the jobs in S by Round-Robin. The sampling with replacement is done as follows. When we take job j as the ith sample, we pretend to create a job i with size x_i equal to $p_{k,j}$. Thus, S could contain multiple "copies" originating from the same job in J_k. So, if S has two copies of the same job, it will get twice the processor share in Round-Robin. This is not an issue as we can simulate the execution of Round-Robin quicker.

When the condition in the following lemma holds true, we will say that $\tilde{m}_k$ is a $(1+\delta)$ *order-approximation* of m_k, or $(1+\delta)$-approximation for brevity.

Lemma 13. *The order of $\tilde{m}_k$ among $\{p_{k,j} \mid j \in J_k\}$ is in $((\frac{1}{2} - \delta)n_k, (\frac{1}{2} + \delta)n_k]$, with probability at least $1 - \frac{1}{n^2}$.*

3.2 Error Estimation

Next, we would like to see if the prediction for the remaining jobs in round k is accurate enough to follow closely. However, measuring the error of the predictions even by running all jobs in a small sample to completion could take too much time. Thus, we estimate the error of the remaining jobs by capping all remaining sizes and predicted sizes at $(1+\epsilon)\tilde{m}_k$. The error we seek to estimate is below.

Definition 14 (Error in Round k). $\eta_k := \textsc{opt}(\{d_{k,j}\}_{j\in J_k})$, *where* $d_{k,j} := |\min\{(1+\epsilon)\tilde{m}_k, p_{k,j}\} - \min\{(1+\epsilon)\tilde{m}_k, \hat{p}_{k,j}\}|$.

Algorithm 2 Error-Estimator$(J_k, \epsilon, n, \tilde{m}_k)$

1: Let P be a uniform random sample, with replacement, of size $\frac{1}{\epsilon^2}\log n$ from a family $Q := \{(j,j) \mid j \in J_k\} \cup \{(i,j) \mid i < j \in J_k\}$ of unordered pairs.
2: For every sampled job j, calculate $d_{k,j}$ by running j up to $(1+\epsilon)\tilde{m}_k$ units.
3: Return as our estimate: $\tilde{\eta}_k := |Q|\frac{1}{|P|}\sum_{(i,j)\in P} \min\{d_{k,i}, d_{k,j}\}$.

For any $k \in [K]$, we say for brevity that $\tilde{\eta}_k$ is a $(1{+}\epsilon)$-approximation of η_k if it satisfies

$$\eta_k - \epsilon\tilde{m}_k n_k^2 \le \tilde{\eta}_k \le \eta_k + \epsilon\tilde{m}_k n_k^2.$$

Lemma 15. *For each $k \in [K]$, $\tilde{\eta}_k$ is a $(1+\epsilon)$-approximation of η_k with probability at least $1 - \frac{1}{n^2}$.*

3.3 Main Algorithm

Given the methods to estimate the median size of all jobs in J_k and the remaining available error, we now describe our algorithm running in rounds $k \ge 1$.

If there are enough jobs alive for accurate sampling, we use our estimators to estimate the median and the error. If the estimated error is big, then we say that the current round is a *RR round*, and run Round-Robin to process all jobs equally up to $2\tilde{m}_k$ units.[8] This is intuitive as our estimator indicates that the prediction is unreliable. If not, we closely follow the prediction. We only consider jobs that are predicted to be small and process them in increasing order of their (remaining) predicted size. To allow for a small prediction error, we allow a job to get processed $3\epsilon\tilde{m}_k$ more units than its

[8] In fact, the jobs can be processed in an arbitrary order as long as they are processed up to $2\tilde{m}_k$ units.

Algorithm 3 Scheduling with Predictions

$k \leftarrow 1$ and $\delta \leftarrow 1/50$.
while $n_k \geq \frac{1}{\epsilon^3} \log n$ **do**
  $\tilde{m}_k \leftarrow$ Median-Estimator(J_k, δ, n).
  $\tilde{\eta}_k \leftarrow$ Error-Estimator$(J_k, \epsilon, n, \tilde{m}_k)$.
  if $\tilde{\eta}_k \geq \epsilon\delta^2 \tilde{m}_k n_k^2/16$ ▷ RR (big error) round
    Process each job in J_k up to $2\tilde{m}_k$ units with Round-Robin.
  else ▷ Non-RR (small error) round
    Process jobs j with $\hat{p}_{k,j} \leq (1+\epsilon)\tilde{m}_k$ up to $\hat{p}_{k,j} + 3\epsilon\tilde{m}_k$ units in increasing order of $\hat{p}_{k,j}$.
  $k \leftarrow k+1$.
Run Round-Robin to complete remaining jobs. ▷ Round $K+1$

remaining predicted size. In this case, we say that the current round is a *non-RR round*. Finally, we run Round-Robin to complete all remaining jobs, if any; this is the final round, indexed by $K+1$.

It is worth mentioning the following easy observations, which will be useful for our analysis later.

Observation 16. (1) *Every overestimated job remains overestimated in every round, i.e., if $\hat{p}_j \geq p_j$, then $\hat{p}_{k,j} \geq p_{k,j}$ for all k. A similar statement holds for every underestimated job.*
(2) *If a job j is processed in a non-RR round k, then its remaining predicted size is 0 for all the subsequent rounds, i.e., $\hat{p}_{k',j} = 0$ for all $k' > k$.*
(3) *For each job, there is exactly one round where the job's remaining predicted size becomes 0.*

4 ANALYSIS

To streamline the presentation of our analysis, we will do the analysis under the following simplifying assumption. We will remove this assumption in Section 4.3.

Assumption 17. *We assume that $\tilde{m}_k$ is a $(1+\delta)$-approximation of m_k ($\delta = 1/50$) and $\tilde{\eta}_k$ is a $(1+\epsilon)$-approximation of η_k. Further, we will assume that the estimation procedures incur no additional delay.*

For analysis we extend the definition of η_k.

Definition 18 (Error in round k on a subset)**.** $\eta_k(X) := \textsc{opt}(\{d_{k,j}\}_{j\in X})$ *for all $X \subseteq J_k$, where $d_{k,j} := |\min\{(1+\epsilon)\tilde{m}_k, p_{k,j}\} - \min\{(1+\epsilon)\tilde{m}_k, \hat{p}_{k,j}\}|$.*

Note that $\eta_k = \eta_k(J_k)$.

4.1 Robustness

In this section, we show that our algorithm always yields a constant approximation. This guarantee holds in all cases even if the predicted job sizes are arbitrarily bad or even adversarially chosen.

Theorem 19. *Algorithm 3 is an $O(1)$-approximation, under Assumption 17.*

Key to the analysis is to show that a constant fraction of jobs complete in each round.

Lemma 20. *For all $k \in [K]$, we have $n_{k+1} \leq (1/2 + 2\delta)n_k$.*

Proof. Suppose round k is an RR round. Since there are $(1/2 - \delta)n_k$ jobs with $p_{k,j} \leq \tilde{m}_k$ and all jobs are processed up to $2\tilde{m}_k$ units, clearly we complete at least $(1/2-\delta)n_k$ jobs in the round.

Now suppose round k is a non-RR round. We first show that there are many jobs considered by the algorithm. Let $X := \{j \in J_k \mid \hat{p}_{k,j} \leq (1+\epsilon)\tilde{m}_k\}$ and $Y := \{j \in J_k \mid p_{k,j} \leq \tilde{m}_k\}$. We claim,

$$|X| \geq (1/2 - 1.5\delta)n_k.$$

Suppose not. Knowing that $|Y| \geq (1/2-\delta)n_k$, we have $|Y \setminus X| \geq 0.5\delta n_k$. Note that for all $j \in Y \setminus X$, $d_{k,j} \geq \epsilon\tilde{m}_k$. We have a contradiction as we have $\eta_k \geq \eta_k(Y \setminus X) \geq \frac{1}{2}\epsilon\tilde{m}_k(0.5\delta n_k)^2$.

Note that all jobs in X are processed by the algorithm. Now we show most of jobs in X complete in the round. Let Z denote those in X that do not complete in the round k. Note that for every job $j \in Z$, we have $d_{k,j} \geq 3\epsilon\tilde{m}_k$. Thus, if we have $|Z| \geq 0.5\delta n_k$, as before, we will have $\eta_k \geq \eta_k(Z) \geq \frac{1}{2}(3\epsilon\tilde{m}_k)(0.5\delta n_k)^2$, another contradiction.

Thus, we have shown $|X \setminus Z| \geq (1/2 - 2\delta)n_k$, meaning the algorithm completes at least $(1/2-2\delta)n_k$ jobs in each round k. □

Intuitively, from Lemma 20, we know that a large number of jobs must complete in each round and further since we assume that $\tilde{m}_k$ approximates the true median well, many of those jobs have remaining sizes at least $\tilde{m}_k$. The following lemma shows that $\Omega(n_k)$ jobs must have remaining size at least $\tilde{m}_k$ and hence the optimal solution must incur a total cost of at least $\Omega(\sum_k \tilde{m}_k n_k^2)$.

Lemma 21. $\sum_{k=1}^{K} \tilde{m}_k n_k^2 \leq 266 \cdot \textsc{opt}(J \setminus J_{K+1})$.

Proof. By Proposition 4 (4), we know $\textsc{opt}(J \setminus J_{K+1})$ is lower bounded by $\sum_{\text{odd } k\in[K-1]} \textsc{opt}(J_k \setminus J_{k+2})$, and also by $\sum_{\text{even } k\in[K-1]} \textsc{opt}(J_k \setminus J_{k+2})$. Thus, we have

$$2\textsc{opt}(J \setminus J_{K+1}) \geq \sum_{k=1}^{K-1} \textsc{opt}(J_k \setminus J_{k+2}).$$

By Lemma 20, we know that at least $(1-(1/2+2\delta)^2)n_k$ jobs in J_k or more complete in round k or $k+1$. Further, as $\tilde{m}_k$ is a $(1+\delta)$-approximation, less than $(1/2+\delta)n_k$ jobs in J_k have $p_{k,j} \leq \tilde{m}_k$. Thus, we conclude that there are at least $(1-((1/2+2\delta)^2)-(1/2+\delta))n_k \geq (1/8)n_k$ jobs in J_k with $p_{k,j} \geq \tilde{m}_k$ that complete in round k or $k+1$; here, $\delta \leq 1/50$. Let F_k denote the set of those jobs. Note that $\textsc{opt}(J_k \setminus J_{k+2}) \geq \textsc{opt}(F_k) \geq (1/2)\tilde{m}_k((1/8)n_k)^2 = \tilde{m}_k n_k^2/128$.

Therefore, we have,

$$2\textsc{opt}(J \setminus J_{K+1}) \geq \sum_{k=1}^{K-1} \frac{1}{128}\tilde{m}_k n_k^2.$$

Further, we have,

$$\textsc{opt}(J \setminus J_{K+1}) \geq \textsc{opt}(J_K) \geq (1/2)\tilde{m}_K(0.45n_K)^2 \geq 0.1\tilde{m}_K n_K^2,$$

as there are at least $(1/2-\delta)n_K \geq 0.45n_K$ jobs of sizes $\geq \tilde{m}_K$. □

Next, we upper bound our algorithm's cost. Let A_k be the total delay incurred by our algorithm in round k. Formally, we have:

$$A_k := \sum_{j\in J_k} q_{k,j} \cdot n_{k,j},$$

where $n_{k,j}$ is the number of jobs that are still alive when job j is processed in round k.

The following notes that each job j gets processed by a maximum of $2\tilde{m}_k$ units and it can delay at most $(n_k - 1)$ jobs in round k.

Lemma 22. *For any $k \in [K]$, $A_k \leq 2\tilde{m}_k n_k^2$.*

Observe that we use Round-Robin in the final round $K+1$, which is known to be 2-competitive. Hence, to complete all the remaining jobs, by considering each job's contribution to the objective, our algorithm's cost is upper bounded as follows.

Lemma 23. *The algorithm's cost is at most* $\sum_{k=1}^{K} 2\tilde{m}_k n_k^2 + 2\textsc{opt}(J_{K+1}) + \sum_{j\in J} p_j$.

By Lemmas 21 and 23, the algorithm's cost is at most $2\cdot 266\ \textsc{opt}(J\setminus J_{K+1}) + 2\textsc{opt}(J_{K+1}) + \textsc{opt}(J) \le 535\ \textsc{opt}(J)$, where the last inequality follows from Proposition 4. This completes the proof of Theorem 19.

4.2 Consistency

In this section, we show that Algorithm 3 also utilizes good predictions to obtain improved guarantees. We analyze the delay incurred by our algorithm in RR rounds and non-RR rounds separately.

4.2.1 RR round. This section is devoted to showing the following lemma. Intuitively, using the fact that the error is huge in each RR round, we can upper bound our algorithm's total delay in all RR rounds by the error. Recall $A_k := \sum_{j\in J_k} q_{k,j}\cdot n_{k,j}$. As δ is set to an absolute constant ($\delta = 1/50$), we will hide it in asymptotic notation. (Recall the surrogate error η from Definition 9.)

Lemma 24. $\sum_{k\in RR} A_k \le O(\frac{1}{\epsilon})\eta \le O(\frac{1}{\epsilon})\nu$, *under Assumption 17.*

From Lemma 22, the total delay A_k incurred in round $k\in[K]$ in our algorithm is at most $2\tilde{m}_k n_k^2$. Thus, our goal is to carefully identify a part of the surrogate error of magnitude $\Omega(\epsilon\tilde{m}_k n_k^2)$ to charge A_k to in each RR round k.

In the following lemma, we consider three types of jobs and show that the error is big enough for at least one type of jobs. The job types are: (i) those completing in round k, (ii) whose remaining predicted sizes become 0 in the round, and (iii) whose remaining predicted sizes are 0 and that do not complete in this round. We need to be careful when extracting some error for type (iii) jobs as they may reappear as type (iii) jobs in subsequent RR rounds. This is why we measure the error by pretending their remaining sizes are $2\tilde{m}_k$, exactly the amount by which the jobs each are processed in the round.

Lemma 25. *In any RR round k, at least one of the following is $\Omega(\epsilon)\tilde{m}_k(n_k)^2$:*

1. $\eta(J_k \setminus J_{k+1})$.
2. $\eta(\hat{F}_k)$ *where* $\hat{F}_k = \{j\in J_k \mid \hat{p}_{k,j} > 0$ *and* $\hat{p}_{k+1,j} = 0\}$.
3. $\eta(\hat{Z}_k;\{2\tilde{m}_k\},\{0\}) = \textsc{opt}(\{2\tilde{m}_k\}_{j\in\hat{Z}_k})$ *where* $\hat{Z}_k := \{j\in J_k \mid \hat{p}_{k,j} = 0$ *and* $p_{k,j} > 2\tilde{m}_k\}$.

Proof. Let $S_k := \{j\in J_k \mid \hat{p}_{k,j} \le (1+\epsilon)\tilde{m}_k$ or $p_{k,j}\le(1+\epsilon)\tilde{m}_k\}$. Since $d_{k,j} = 0$ for all jobs $j\in J_k\setminus S_k$, by definition of η_k, we have $\eta_k = \eta_k(S_k)$. For notational convenience, let $X := S_k\cap(J_k\setminus J_{k+1})$, $Y := S_k\cap\hat{F}_k$ and $Z := S_k\cap\hat{Z}_k$. As each job in J_k is of one of the above three types, we have $S_k = X\cup Y\cup Z$.

Since k is an RR round, we have $\eta_k = \eta_k(S_k) = \Omega(\epsilon\tilde{m}_k n_k^2)$. Because of the monotonicity of η_k (it can only become larger when more jobs are considered) and the fact that $S_k = X\cup Y\cup Z$, by Proposition 4, we know at least one of $\eta_k(X)$, $\eta_k(Y)$, $\eta_k(Z)$ must be no smaller than $(1/3)\eta_k$. We consider each case in the following.

Case i. $\eta_k(X)\ge(1/3)\eta_k$. Let X_o, X_u denote the jobs in X that are overestimated and underestimated, respectively. By definition of η_k and Proposition 4, we have $\eta_k(X_o)+\eta_k(X_u)\ge(1/2)\eta_k(X)$, and

$$\begin{aligned}\eta_k(X_u) &= \textsc{opt}(\{\min\{(1+\epsilon)\tilde{m}_k, p_{k,j}\} - \min\{(1+\epsilon)\tilde{m}_k, \hat{p}_{k,j}\}\}_{j\in X_u})\\ &\le \textsc{opt}(\{p_{k,j}-\hat{p}_{k,j}\}_{j\in X_u})\\ &\le \textsc{opt}(\{p_{k,j}\}_{j\in X_u}) - \textsc{opt}(\{\hat{p}_{k,j}\}_{j\in X_u}) \;=\; \eta(X_u).\end{aligned}$$

Similarly, we can show $\eta_k(X_o)\le\eta(X_o)$. Thus, we have,

$$\begin{aligned}\eta(J_k\setminus J_{k+1}) &\ge \eta(X) \ge \eta(X_u)+\eta(X_o) \ge \eta_k(X_u)+\eta_k(X_o)\\ &\ge (1/2)\eta_k(X) \ge (1/6)\eta_k,\end{aligned}$$

where the first two inequalities follow from Proposition 11.

Case ii. $\eta_k(Y)\ge(1/3)\eta_k$. This case can be similarly handled as the first case, and we can show $\eta(\hat{F}_k)\ge\eta(Y)\ge(1/6)\eta_k$.

Case iii. $\eta_k(Z)\ge(1/3)\eta_k$. Note that all jobs j in Z are underestimated because of $p_{k,j} > 2\tilde{m}_k$ and $\hat{p}_{k,j} = 0$ and by Observation 16. Also, by definition of S_k, Z, $\hat{Z}_k$, we have $Z = \hat{Z}_k$. Therefore,

$$\begin{aligned}\eta_k(Z) &= \textsc{opt}(\{\min\{(1+\epsilon)m_k, p_{k,j}\} - \min\{(1+\epsilon)m_k, \hat{p}_{k,j}\}\}_{j\in Z})\\ &= \textsc{opt}(\{(1+\epsilon)\tilde{m}_k\}_{j\in Z}) \;\le\; \eta(\hat{Z}_k;\{2\tilde{m}_k\},\{0\}).\end{aligned}$$

Thus we have $\eta(\hat{Z}_k;\{2\tilde{m}_k\},\{0\})\ge(1/3)\eta_k$. □

We next show that the above errors add up to $O(\eta)$.

Lemma 26. $\sum_{k\in RR}\Big(\eta(J_k\setminus J_{k+1}) + \eta(\hat{F}_k) + \eta(\hat{Z}_k;\{2\tilde{m}_k\},\{0\})\Big) \le 3\eta$.

Proof. We start by considering the first quantity. Since the sets in $\{J_k\setminus J_{k+1}\}_{k\ge1}$ are disjoint, from Proposition 11, we know that $\sum_{k\in RR}\eta(J_k\setminus J_{k+1})\le\eta$. Similarly, we can show $\sum_{k\in RR}\eta(\hat{F}_k)\le\eta$.

It now remains to show

$$\sum_{k\in RR}\eta(\hat{Z}_k;\{2m_k\},\{0\})\le\eta.$$

Note that by definition of $\hat{Z}_k$ and Observation 16, we know that if $j\in\hat{Z}_k$, then j must be underestimated, i.e., $\hat{p}_j\le p_j$. Let J_u denote the set of all underestimated jobs. By definition of η, we know that $\eta\ge\textsc{opt}(\{p_j\}_{j\in J_u}) - \textsc{opt}(\{\hat{p}_j\}_{j\in J_u})$. Further, by Proposition 4,

$$\eta\ge\textsc{opt}(\{p_j-\hat{p}_j\}_{j\in J_u}).$$

The proof idea is to show the decrease of $\textsc{opt}(\{p_{k,j}-\hat{p}_{k,j}\}_{j\in J_u})$ in each RR round k is as big as $\eta(\hat{Z}_k;\{2\tilde{m}_k\},\{0\})$. By definition of $\hat{p}_{1,j}$ and $p_{1,j}$, we have

$$\textsc{opt}(\{p_{1,j}-\hat{p}_{1,j}\}_{j\in J_u}) = \textsc{opt}(\{p_j-\hat{p}_j\}_{j\in J_u}).$$

Further, by Observation 16, we know $\hat{Z}_k\subseteq J_u$. Note that for every job $j\in\hat{Z}_k$, $((p_{k,j}-\hat{p}_{k,j})-(p_{k+1,j}-\hat{p}_{k+1,j})) = p_{k,j}-p_{k+1,j} = 2\tilde{m}_k$. Thus, we have,

$$\begin{aligned}&\textsc{opt}(\{p_{k,j}-\hat{p}_{k,j}\}_{j\in J_u}) - \textsc{opt}(\{p_{k+1,j}-\hat{p}_{k+1,j}\}_{j\in J_u})\\ &\ge \textsc{opt}(\{(p_{k,j}-\hat{p}_{k,j})-(p_{k+1,j}-\hat{p}_{k+1,j})\}_{j\in J_u})\\ &\quad\text{[Proposition 4 and } (p_{k,j}-\hat{p}_{k,j}) \text{ is decreasing in } k\ \forall j\in J_u]\\ &\ge \textsc{opt}(\{(p_{k,j}-\hat{p}_{k,j})-(p_{k+1,j}-\hat{p}_{k+1,j})\}_{j\in\hat{Z}_k})\\ &\quad\text{[Monotonicity of opt]}\\ &= \textsc{opt}(\{2\tilde{m}_k\}_{j\in\hat{Z}_k}) = \eta(\hat{Z}_k;\{2\tilde{m}_k\},\{0\}).\end{aligned}$$ □

Lemmas 25, 26 with Lemma 22 complete the proof of Lemma 24.

4.2.2 Non-RR round. This section is devoted to proving the following lemma that bounds our algorithm's total delay in non-RR rounds (denoted NRR). As we do not have sufficiently large errors in non-RR rounds, we will have to bound it by both OPT and ν. Note that $\textsc{opt} - (\sum_{i\in J} p_i)$ is the total delay cost of the optimal schedule.

Lemma 27. *Under Assumption 17, we have* $\sum_{k\in\text{NRR}} A_k \le (1 + O(\epsilon))\textsc{opt} - (\sum_{i\in J} p_i) + O(1/\epsilon^2)\nu$.

We begin our analysis of consistency for non-RR rounds by proving the following lemma, which shows how much error we can use for each pair of jobs.

Lemma 28. *$\nu(J, \{p_j\}, \{\hat{p}_j\}) \ge \sum_{i\ne j\in J} \nu(i, j)$, where $\nu(i, j) = |\min\{p_i, p_j\} - \min\{\hat{p}_i, \hat{p}_j\}|$.*

Proof. By Proposition 3 we can decompose ν as follows.

$$\begin{aligned}
&\nu(J, \{p_j\}, \{\hat{p}_j\}) \\
&= \textsc{opt}\Big(\{\hat{p}_j\}_{j\in J_o} \cup \{p_j\}_{j\in J_u}\Big) - \textsc{opt}\Big(\{p_j\}_{j\in J_o} \cup \{\hat{p}_j\}_{j\in J_u}\Big) \\
&\ge \sum_{i,j\in J_o} (\min\{\hat{p}_i, \hat{p}_j\} - \min\{p_i, p_j\}) + \sum_{i,j\in J_u} (\min\{p_i, p_j\} - \min\{\hat{p}_i, \hat{p}_j\}) \\
&\quad + \sum_{i\in J_o, j\in J_u} (\min\{\hat{p}_i, p_j\} - \min\{p_i, \hat{p}_j\}).
\end{aligned}$$

Since every term in the summation is non-negative, to prove the lemma it suffices to show

$$\min\{\hat{p}_i, p_j\} - \min\{p_i, \hat{p}_j\} \ge |\min\{p_i, p_j\} - \min\{\hat{p}_i, \hat{p}_j\}|, \quad \forall i \in J_o \text{ and } j \in J_u. \tag{1}$$

By definition, we have $\hat{p}_i \ge p_i$ for $i \in J_o$, and hence we have $\min\{\hat{p}_i, p_j\} \ge \min\{p_i, p_j\}$ and $\min\{p_i, \hat{p}_j\} \le \min\{\hat{p}_i, \hat{p}_j\}$. Hence,

$$\min\{\hat{p}_i, p_j\} - \min\{p_i, \hat{p}_j\} \ge \min\{p_i, p_j\} - \min\{\hat{p}_i, \hat{p}_j\}.$$

Similarly, since $\hat{p}_j \le p_j$ for $j \in J_u$, we have $\min\{\hat{p}_i, p_j\} \ge \min\{\hat{p}_i, \hat{p}_j\}$ and $\min\{p_i, \hat{p}_j\} \le \min\{p_i, p_j\}$. Hence,

$$\min\{\hat{p}_i, p_j\} - \min\{p_i, \hat{p}_j\} \ge \min\{\hat{p}_i, \hat{p}_j\} - \min\{p_i, p_j\}.$$

In either case, we have shown (1) holds, as desired. □

Knowing how much error we can use for each pair of jobs, we are now ready to give an overview of the analysis. We let $D_k(i, j)$ denote the delay i causes to j in a non-RR round k. Note that $D_k(i, j) = q_{k,i}$ if j is still alive while i gets processed in round k; otherwise, $D_k(i, j) = 0$.

1. Total delay involving jobs with zero remaining predicted sizes. We show that the delay involving the following jobs across all non-RR rounds is at most $O(\epsilon) \cdot \textsc{opt}$.

$$\hat{Z}_k := \{j \in J_k \mid \hat{p}_{k,i} = 0\}.$$

Fix a job $i \in \hat{Z}_k$. Note that such a job i gets processed by at most $3\epsilon\tilde{m}_k$. Further, if job j gets processed before i, it implies $\hat{p}_{k,j} = 0$, where j can delay i by at most $3\epsilon\tilde{m}_k$ in the round. Similarly, job i can delay another job by at most $3\epsilon\tilde{m}_k$ in the round. It is an easy exercise to see that total delay involving a job i with $\hat{p}_{k,i} = 0$ is at most $3\epsilon\tilde{m}_k n_k$. As there are at most n_k jobs remaining in this round,

$$\begin{aligned}
&\sum_{k\in\text{NRR}} \sum_{i\in\hat{Z}_k} \sum_{j\in J_k: j\ne i} (D_k(i, j) + D_k(j, i)) \\
&\le \sum_{k\in\text{NRR}} 3\epsilon\tilde{m}_k n_k^2 \le O(\epsilon)\textsc{opt}(J \setminus J_{K+1}) \le O(\epsilon)\textsc{opt},
\end{aligned} \tag{2}$$

where the second inequality follows from Lemma 21.

2. Total delay involving jobs that execute but do not complete. We show the total delay across all non-RR rounds is at most $O(\epsilon)\textsc{opt} + O(1/\epsilon^2)\nu$. To precisely articulate what we aim to prove, define:

$$U_k := \{i \in J_k \mid 0 < \hat{p}_{k,i} \le (1 + \epsilon)\tilde{m}_k \text{ and } p_{k,i} > \hat{p}_{k,i} + 3\epsilon\tilde{m}_k\},$$

which, roughly speaking, are the jobs with relatively small non-zero remaining predicted sizes that execute but do not complete in round k. Note that if $i \in U_k$, then $\hat{p}_{k,i} > 0$ and $\hat{p}_{k+1,i} = 0$. Therefore, the family $\{U_k\}_{k\in[K]}$ is disjoint.

The following bounds the total delay incurred due to jobs in U_k.

Lemma 29. *For each $i \in U_k$, let $D_{k,i} := \sum_{j\in J_k: j\ne i}(D_k(i, j) + D_k(j, i)) = \Big(\sum_{j\in J_k: j\ne i} q_{k,j} + \sum_{j\in J_k: j\ne i, C_j > L_{k,i}} (3\epsilon\tilde{m}_k + \hat{p}_{k,i})\Big)$ be the total delay involving job i in a non-RR round k, where $L_{k,i}$ denotes the last time when i gets processed in round k and C_j is j's completion time. Then, we have*

$$\sum_{i\in U_k} D_{k,i} \le O(\epsilon)\tilde{m}_k n_k^2 + O(1/\epsilon^2) \sum_{i\in U_k} \sum_{j\in J_k: j\ne i} \nu(i, j).$$

Note that in $D_{k,i}$, the first term is how much other jobs delay i and the second is how much job i delays other jobs: job i delays job j in the round by exactly $\hat{p}_{k,i} + 3\epsilon\tilde{m}_k$ if j is still alive when the algorithm stops processing i in the round. The proof is a bit subtle and is omitted for space, but the intuition is the following. Suppose we made a bad mistake by working on job $i \in U_k$ in round k—we thought the job was small based on its prediction but it turned out to be big. This means that job i's processing delays many jobs in J_k, which we could have avoided had we had known that i was in fact big. Thus, to charge the delay, we show that the considerable underprediction of job i creates a huge error as it makes a large difference in how much i delays other big jobs.

Assuming Lemma 29, we have,

$$\begin{aligned}
\sum_{k\in\text{NRR}} \sum_{i\in U_k} D_{k,i} &\le \sum_{k\in\text{NRR}} O(\epsilon)\tilde{m}_k n_k^2 + O(1/\epsilon^2) \sum_{i\in U_k} \sum_{j\in J_k: j\ne i} \nu(i, j) \\
&\le O(\epsilon)\textsc{opt} + O(1/\epsilon^2) \sum_{k\in\text{NRR}} \sum_{i\in U_k, j\in J: j\ne i} \nu(i, j) \\
&\qquad [\text{Lemma 21 and } J_k \subseteq J] \\
&\le O(\epsilon)\textsc{opt} + O(1/\epsilon^2) \sum_{i\ne j\in J} \nu(i, j) \\
&\qquad [U_1, \ldots, U_K \text{ are disjoint}] \\
&\le O(\epsilon)\textsc{opt} + O(1/\epsilon^2) \cdot \nu \qquad [\text{Lemma 28}].
\end{aligned} \tag{3}$$

3. Total delay due to the other jobs. Finally we consider delay not considered by the two cases. Let us see for which pairs of jobs we did not consider their pairwise delay. For job i to delay job j in a non-RR round k, i must be processed, meaning that $\hat{p}_{k,i} \le (1+\epsilon)\tilde{m}_k$.

Since Case 1 already considered $\hat{p}_{k,i} = 0$, $i \in \hat{Z}_k$, we assume $\hat{p}_{k,i} > 0$. Further, if i does not complete in round k, we already covered the delay in Case 2 as $i \in U_k$. Thus, we only need to consider the case when $i \in V_k$, where V_k is defined as follows.

$$V_k := \{i \in J_k \mid 0 < \hat{p}_{k,i} \le (1+\epsilon)\tilde{m}_k, p_{k,i} \le \hat{p}_{k,i} + 3\epsilon\tilde{m}_k\}.$$

Note that every $i \in V_k$ completes in round k. The following upper bounds the total delay we did not consider in the previous cases.

Lemma 30. *For any $i \in V_k$, $j \in J_k \setminus (\hat{Z}_k \cup U_k)$, the delay i causes to j in non-RR round k, $D_k(i,j)$, is at most $\min\{p_i, p_j\} + \nu(p_i, p_j) + 3\epsilon\tilde{m}_k$.*

Note that $\{V_k\}_k$ is a family of disjoint job sets. This is because every job $i \in V_k$ has non-zero remaining predicted size and gets processed in the round; see Observation 16. Thus, k is the last round where i's remaining predicted size is non-zero. Therefore, we have,

$$\sum_{k \in \text{NRR}} \sum_{i \in V_k, j \in J_k \setminus (\hat{Z}_k \cup U_k): \hat{p}_{k,j} > \hat{p}_{k,i}} D_k(i,j) \tag{4}$$

$$= \sum_{k \in \text{NRR}} \sum_{i \in V_k, j \in J_k \setminus (\hat{Z}_k \cup U_k): \hat{p}_{k,j} > \hat{p}_{k,i}} \left(\min\{p_i, p_j\} + \nu(p_i, p_j) + 3\epsilon\tilde{m}_k \right)$$

[Lemma 30]

$$\le \sum_{\{i,j\} \subseteq J: i \ne j} \left(\min\{p_i, p_j\} + \nu(p_i, p_j) \right) + \sum_{k \in [K]} 3\epsilon\tilde{m}_k n_k^2$$

[$V_1, \ldots, V_K$ are disjoint]

$$\le \text{OPT} - \left(\sum_{i \in J} p_i \right) + \nu + O(\epsilon)\text{OPT} \quad \text{[Proposition 3, Lemmas 28, 21].}$$

Putting all pieces together. Note that the delay incurred between every pair of jobs i and j in every non-RR round k falls into at least one of the above three categories. Thus, from (2), (3), and (4), the total pairwise delay in non-RR rounds is at most,

$$O(\epsilon)\text{OPT} + O(1/\epsilon^2)\nu + \text{OPT} - \left(\sum_{i \in J} p_i \right) + \nu + O(\epsilon)\text{OPT}. \tag{5}$$

We are now ready to give the final upper bound on the objective of our algorithm, which is obtained by combining the upper bound in Lemma 24 and (5) and by factoring in the total job size, $\sum_{i \in J} p_i$. We state the result with ϵ scaled appropriately by a constant factor.

Theorem 31. *Under Assumption 17, Algorithm 3's objective is at most $(1+\epsilon)\text{OPT} + 2\text{OPT}(J_{K+1}) + O(1/\epsilon^2)\nu$.*

4.3 Removing Simplifying Assumptions

Our goal here is to extend Theorem 31 by removing Assumption 17. We say that a *bad event* B_k occurs in round k if $\tilde{m}_k$ fails to be $(1+\delta)$-approximate or $\tilde{\eta}_k$ fails to be $(1+\epsilon)$-approximate; by Lemmas 13, 15, B_k occurs with probability at most $2/n^2$. If B_k does not occur, we know that a constant fraction of jobs complete in round k thanks to Lemma 20. Thus, if no bad events occur, we have $K = O(\log n)$. By a union bound, bad events occur with probability $O((\log n)/n^2)$.

We now factor in the extra delays due to estimating m_k and η_k, assuming no bad events occur. In the median estimation, we took a sample S of size $O(\frac{\log 2n}{\delta^2})$ and processed every job in S by exactly $\tilde{m}_k$. So, the maximum delay due to the processing is at most $(\tilde{m}_k) \cdot |S| \cdot |J_k| = O((\log n)\tilde{m}_k n_k)$. Similarly, in estimating η_k, we took a sample P of size $O(\frac{1}{\epsilon^2}\log n)$ and processed both jobs in each pair in P up to $(1+\epsilon)\tilde{m}_k$ units. Thus, this processing cause total extra delay at most $2(1+\epsilon)\tilde{m}_k \cdot |P| \cdot |J_k| = O(\frac{1}{\epsilon^2}(\log n)\tilde{m}_k n_k)$.

Hence, the extra delay cost due to the estimation is bounded by

$$O(\frac{1}{\epsilon^2})(\log n) \sum_{k \in [K]} \tilde{m}_k n_k$$

$$\le O(\epsilon) \sum_{k \in [K]} \tilde{m}_k n_k^2 \quad [n_k = |J_k| \ge \tfrac{1}{\epsilon^3}\log n \text{ for all } k \in [K]]$$

$$\le O(\epsilon)\text{OPT}(J \setminus J_{K+1}) \qquad \text{[Lemma 21],}$$

with probability $1 - O((\log n)/n^2)$.

Thus, the extra delay is negligible w.h.p. Further, knowing that any (non-idle) algorithm, including ours, is n-approximate, the bad events can increase the objective by $O((\log n)/n^2)n \cdot \text{OPT}$ in expectation, which is again negligible.

The above discussions, Theorem 19, and Theorem 31, yield:

Theorem 32. *Algorithm 3's objective is at most $\min\{O(1)\text{OPT}, (1+\epsilon)\text{OPT} + 2\text{OPT}(J_{K+1}) + O(1/\epsilon^2)\nu\}$ with high probability, where $|J_{K+1}| \le \frac{1}{\epsilon^3}\log n$. Further, the same bound holds in expectation.*

Corollary 33. *Suppose for any $Z \subseteq J$ with $|Z| \le \frac{1}{\epsilon^3}\log n$, $\text{OPT}(Z) \le \epsilon\text{OPT}$. Then, Algorithm 3's objective is at most $\min\{O(1)\text{OPT}, (1+\epsilon)\text{OPT} + O(1/\epsilon^2)\nu\}$ with high probability. Further, the same bound holds in expectation.*

4.4 Guarantees in Expectation

Previously we showed high probability guarantees on our algorithm's objective. However, high probability guarantees inherently require $\Omega(\log n)$ samples and therefore we are forced to stop sampling once the number of jobs alive becomes $o(\log n)$. Here we show that we can further continue to sample, if we only need guarantees in expectation, until we have $O(\frac{1}{\epsilon^3}\log\frac{1}{\epsilon})$ unfinished.

Towards this end, we slightly change the algorithm.

(1) Reduce the sample sizes: for estimating m_k take a sample of size $\frac{1}{\delta^2}\log 2n_k$ in Algorithm 1 and for estimating η_k take a sample of size $\frac{1}{\epsilon^2}\log n_k$ in Algorithm 2.

(2) Run Round-Robin concurrently: We divide each instantaneous time to Round-Robin by ϵ fraction of time and to run our algorithm by $1-\epsilon$ fraction of time. Since this can only slow down the execution of our algorithm by a factor of $1-\epsilon$, the bound in Theorem 32 only increases by a factor of $1/(1-\epsilon)$, which has no effect on our asymptotic bounds. But by running the 2-competitive Round-Robin concurrently, our final schedule will always be $2/\epsilon$-competitive.

(3) Stop sampling if $n_k \le O(\frac{1}{\epsilon^3}\log\frac{1}{\epsilon})$ (Line 2, Algorithm 3): This is doable as we can withstand higher probabilities of bad events thanks to the concurrent execution of Round-Robin.

(4) In the final round $K+1$, process all jobs in increasing order of their predicted size: As we only have $|J_{K+1}| = O(\frac{1}{\epsilon^3}\log\frac{1}{\epsilon})$ jobs left, following the prediction blindly will not hurt much!

Theorem 34. *For any sufficiently small $\epsilon > 0$, there exists an algorithm whose expected objective is at most*

$$\max\left\{ \frac{2}{\epsilon}\text{OPT}, (1+\epsilon)\text{OPT} + O(\frac{1}{\epsilon^3}\log\frac{1}{\epsilon})\nu \right\}.$$

5 LOWER BOUNDS

We first show our analysis of Algorithm 3 is tight. Theorem 31 implies that the algorithm's objective is at most $(1+\epsilon)\text{OPT} + \frac{1}{\epsilon^2}\nu$ if for any subset $Z \subseteq J$ of jobs whose size is polylogarithmic in n, $\text{OPT}(Z) = o(\text{OPT})$. The lower bound instance used in the proof of the following theorem satisfies the property.

Theorem 35. *For any $\gamma > 0$, there exists a sufficiently small $\epsilon > 0$, such that there is an instance that shows our algorithm's objective is greater than $(1+\epsilon)OPT + \Omega(1/\epsilon^{2-\gamma})\nu$.*

Proof. Let $\beta := \epsilon^{1-\gamma}$. There are two groups of jobs, X and Y. Group X consists of βn jobs that each have true size $p_j = 1+\beta$ and predicted size $\hat{p}_j = 1$. Group Y consists of the remaining $(1-\beta)n$ jobs with unit true and predicted sizes, i.e., $p_j = \hat{p}_j = 1$ for all $j \in Y$.

Let A denote the total completion time of the schedule found by our algorithm, and let OPT denote that of the optimal solution for the true job sizes. In this case, it is easy to verify that $\text{OPT} = \Theta(n^2)$ and the total error $\nu = \eta(X) = \Theta(\beta^3 n^2)$. For brevity, assume that the algorithm's median and error estimation is exact. Then, $\tilde{m}_1 = 1$ and $\tilde{\eta}_1 = \Theta(\epsilon\beta^2 n^2)$. Thus, the algorithm's first round is non-RR. All jobs have the same predicted size and therefore are indistinguishable to our algorithm. Say it first considers jobs in X. Unfortunately, it finishes no jobs in X in the first round as $\beta = \omega(\epsilon)$. Thus, the algorithm has at least $|X|\cdot|Y|$ more units of delay than the optimum solution that first completes all jobs in Y and then those in X. Thus, we have $A - \text{OPT} \geq (1+\epsilon)|X||Y| \geq \beta(1-\beta)n^2 = \Theta(\beta n^2)$. But since $\epsilon = o(\beta)$ and $\text{OPT} = \Theta(n^2)$, we have $A - (1+\epsilon)\text{OPT} \geq \Omega(\beta n^2) = \Omega(\frac{1}{\beta^2})\nu$. □

Next, we show no deterministic algorithm can improve upon $(1+\epsilon)$-consistency and $O(1/\epsilon)$-robustness simultaneously.

Theorem 36. *For any sufficiently small $\epsilon > 0$ and $\gamma > 0$, no deterministic algorithm's objective is at most $(1+\epsilon)OPT + O(1/\epsilon^{1-\gamma})\nu$.*

Proof. Consider the following lower bound instance. There are $n := 1/\epsilon^{1-\gamma}$ jobs, where $\gamma > 0$ is a sufficiently small constant. All jobs have predicted sizes 1. Suppose all jobs have true sizes exactly 1, except one job having size 2, which we call *big*. Note that $\text{OPT} = n(n+1)/2 + 1$ and $\nu = 1$. Thus, we have $\epsilon\text{OPT} + O(1/\epsilon)\nu = O(1/\epsilon^{1-2\gamma} + 1/\epsilon) = O(1/\epsilon^{1-2\gamma})$. Since our goal is to show that $A = (1+\epsilon)\text{OPT} + O(1/\epsilon)\nu$, it suffices to show $A - \text{OPT} = \omega(1/\epsilon^{1-2\gamma})$. The adversary lets the big job to be the first the algorithm has processed by at least one unit. This is a valid strategy for the adversary as all jobs are indistinguishable to the algorithm until it processes jobs by one unit or more. Thus, the big job delays each of the rest of the jobs by at least one unit in the adversary. Let A denote a fixed algorithm's objective. We have $A - \text{OPT} \geq n - 1 = \omega(1/\epsilon^{1-2\gamma})$. □

6 CONCLUSIONS AND OPEN PROBLEMS

In this paper we defined a new prediction error measure based on the desiderata we established. We believe that our new measure could be useful for other optimization problems with ML predictions where ℓ_1-norm measure is not ideal. Applying our approach to other problems could lead to new algorithmic solutions.

Regarding the specific problem considered in the paper, interesting directions include finding a deterministic algorithm with similar guarantees, obtaining a better dependence on ν, and extending the error notion to the setting where jobs have different arrival times.

ACKNOWLEDGMENTS

Im and Montazer Qaem are supported in part by NSF grants CCF-1617653 and CCF-1844939. Part of this work was done while Im was visiting Google.

REFERENCES

[1] Anders Aamand, Piotr Indyk, and Ali Vakilian. 2019. (Learned) frequency estimation algorithms under Zipfian distribution. arXiv:1908.05198.
[2] Maryam Amiri and Leyli Mohammad-Khanli. 2017. Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications* 82 (2017), 93–113.
[3] Keerti Anand, Rong Ge, and Debmalya Panigrahi. 2020. Customizing ML predictions for online algorithms. In *ICML*. 303–313.
[4] Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. 2021. Online bin packing with predictions. arXiv:2102.03311.
[5] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. 2020. Online Metric Algorithms with Untrusted Predictions. In *ICML*. 345–355.
[6] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. 2020. Secretary and online matching problems with machine learned advice. In *NeurIPS*.
[7] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. 2020. Learning augmented energy minimization via speed scaling. In *NeurIPS*.
[8] Étienne Bamas, Andreas Maggiori, and Ola Svensson. 2020. The primal-dual method for learning augmented algorithms. In *NeurIPS*.
[9] Edith Cohen, Ofir Geri, and Rasmus Pagh. 2020. Composable sketches for functions of frequencies: Beyond the worst case. In *ICML*. 2057–2067.
[10] Sreenivas Gollapudi and Debmalya Panigrahi. 2019. Online algorithms for rent-or-buy with expert advice. In *ICML*. 2319–2327.
[11] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. 2019. Learning-based frequency estimation algorithms. In *ICLR*.
[12] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. 2017. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. *JACM* 65, 1 (2017), 1–33.
[13] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. 2020. Online algorithms for weighted paging with predictions. *ICALP* (2020), 69:1–69:18.
[14] Bala Kalyanasundaram and Kirk Pruhs. 2000. Speed is as powerful as clairvoyance. *JACM* 47, 4 (2000), 617–643.
[15] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *SIGMOD*. 489–504.
[16] Ravi Kumar, Manish Purohit, and Zoya Svitkina. 2018. Improving online algorithms Using ML predictions. In *NeurIPS*. 9661–9670.
[17] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. 2020. Online scheduling via learned weights. In *SODA*. 1859–1877.
[18] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. 2020. Learnable and instance-robust predictions for online matching, flows and load balancing. arXiv:2011.11743.
[19] Thodoris Lykouris and Sergei Vassilvtiskii. 2018. Competitive caching with machine learned advice. In *ICML*. 3296–3305.
[20] Andréa Matsunaga and José AB Fortes. 2010. On the use of machine learning to predict the time and resources consumed by applications. In *Cluster, Cloud and Grid Computing*. 495–504.
[21] Michael Mitzenmacher. 2018. A model for learned Bloom filters and optimizing by sandwiching. In *NeurIPS*. 464–473.
[22] Michael Mitzenmacher. 2020. Scheduling with predictions and the price of misprediction. In *ITCS*. 14:1–14:18.
[23] Rajeev Motwani, Steven Phillips, and Eric Torng. 1994. Nonclairvoyant scheduling. *Theoretical Computer Science* 130, 1 (1994), 17–47.
[24] Ilia Pietri, Gideon Juve, Ewa Deelman, and Rizos Sakellariou. 2014. A performance model to estimate execution time of scientific workflows on the Cloud. In *Workshop on Workflows in Support of Large-Scale Science*. 11–19.
[25] Kirk Pruhs, Jirí Sgall, and Eric Torng. 2004. Online Scheduling. In *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*, Joseph Y.-T. Leung (Ed.). Chapman and Hall/CRC.
[26] Dhruv Rohatgi. 2020. Near-Optimal Bounds for Online Caching with Machine Learned Advice. In *SODA*. 1834–1845.
[27] Tim Roughgarden. 2020. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press.
[28] Kapil Vaidya, Eric Knorr, Tim Kraska, and Michael Mitzenmacher. 2021. Partitioned learned Bloom filter. In *ICLR*.