Scheduling with Untrusted Predictions

Evripidis Bampis[1] , **Konstantinos Dogeas**[1] , **Alexander Kononov**[2,3] ,
Giorgio Lucarelli[4] and **Fanny Pascual**[1]
[1]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[2]Novosibirsk State University, Novosibirsk, Russia
[3]Sobolev Institute of Mathematics, Novosibirsk, Russia
[4]Université de Lorraine, LCOMS, Metz, France
evripidis.bampis@lip6.fr, konstantinos.dogeas@lip6.fr, alvenko@math.nsc.ru,
glucarelli@univ-lorraine.fr, fanny.pascual@lip6.fr

Abstract

Using machine-learned predictions to create algorithms with better approximation guarantees is a very fresh and active field. In this work, we study classic scheduling problems under the learning augmented setting. More specifically, we consider the problem of scheduling jobs with arbitrary release dates on a single machine and the problem of scheduling jobs with a common release date on multiple machines. Our objective is to minimize the sum of completion times. For both problems, we propose algorithms which use predictions when making their decisions. Our algorithms are consistent – i.e. when the predictions are accurate, the performances of our algorithms are close to those of an optimal offline algorithm–, and robust – i.e. when the predictions are wrong, the performance of our algorithms are close to those of an online algorithm without predictions. In addition, we confirm the above theoretical bounds by conducting experimental evaluation comparing the proposed algorithms to the offline optimal ones for both the single and multiple machines settings.

1 Introduction

Scheduling is one of the most studied problems in computer science. In its typical setting, we have a set of jobs that need to be executed in a set of machines. However, in the majority of the related literature, research works share a common assumption, i.e. the processing time of each job is known (clairvoyance) either in advance or when the job becomes available. On one hand, such an assumption can lead to good, even optimal, algorithms but on the other hand, knowing the exact processing times is not often the case on real world systems. The setting that tackles this issue is called *non-clairvoyant* scheduling, where only the existence of a job is revealed to the algorithm, which learns the exact processing time of that job only after the job finishes its execution. However, nowadays, machine learning algorithms are being used to provide predictions when there is not enough or accurate data. The community has started incorporating predicted values as part of the input in order to design algorithms with better performance guarantees. In the scheduling framework, predictions is a way to bridge the gap between the clairvoyant and the non-clairvoyant setting. Ideally, we want to use the predictions to produce algorithms that are *consistent* and *robust*. We say an algorithm is consistent, when it performs close to the best offline algorithm, if the predictor is good. In addition, an algorithm is robust, when it performs close to the online algorithm, without predictions. Finally, we would like the algorithm's performance to deteriorate smoothly as the predictions are less accurate.

Problem formulation and Contributions. In this work, we consider a *non-clairvoyant* scheduling problem on one or multiple machines. We are given a set of jobs $\mathcal{J}$ whose actual processing times are not known in advance. However, a predicted value of the processing time is given for each job. Each job is also characterized by a *release time (date)*. In the whole extent of this paper, preemption and migration are allowed at no extra cost. In other words, jobs can be stopped and continue execution in a later time (preemption) and eventually on a different machine (migration). Each machine can execute at most one job at a time. In the first version, we consider scheduling $\mathcal{J}$ on a single machine. Jobs arrive over time, and the algorithm has no prior knowledge on the existence of a job. In the second version, we consider scheduling $\mathcal{J}$ on m identical machines. In this case, we assume that all release times are zero and the algorithm has knowledge of the total number of jobs as well as their predicted processing times in the beginning. The objective in both problems is to minimize the sum of completion times. Our goal is to design algorithms that are both consistent and robust.

A common strategy to obtain these two properties is to construct a preferential algorithm that runs simultaneously a consistent and a robust algorithm, getting in this way the best of the two worlds. In this direction, we provide in Section 3 a consistent optimal algorithm, called *Shortest Remaining Predicted Processing Time*, when the predictions are accurate for the single machine problem. By combining this with the robust non-clairvoyant Round Robin algorithm, we give a preferential algorithm for the problem of minimizing the sum of completion times on a single machine with release times. The Round Robin algorithm schedules for an equal amount of time all the ready jobs, at any time moment. In

Section 4 we give again a consistent optimal algorithm, called *Shortest Predicted Processing Time First*, when the predictions are accurate for the multiple machines problem. It is also known that Round Robin is 2-competitive for the non-clairvoyant problem in multiple machines [Motwani *et al.*, 1994]. Then, we give the first preferential algorithm for the problem of minimizing the sum of completion times on multiple machines.

Related Work. The non-clairvoyant scheduling problem is studied in [Motwani *et al.*, 1994] and the authors consider different variants. In their work, non-clairvoyant algorithms (such as Round Robin) are compared to optimal clairvoyant ones resulting in both randomized and deterministic lower and upper bounds. In a very recent work, [Moseley and Vardi, 2022] study Round Robin's performance for the ℓ_p-norm of the completion times when scheduling n preemptive jobs on a single machine. For the version of the problem without release times on a single machine, they prove Round Robin's approximation ratio to be $\sqrt[p]{p+1}$, while when jobs arrive over time, Round Robin's competitive ratio is shown to be at most 4 for any $p \geqslant 1$. Garg et al. consider the online problem of scheduling non-clairvoyant jobs on identical machines, where jobs have precedence constraints [Garg *et al.*, 2019]. Their objectives is to minimize the total weighted completion time and the total weighted flow time.

Even though the idea of using advice to obtain semi-online algorithms is not new (see the following survey [Boyar *et al.*, 2017]), in a recent line of work, [Medina and Vassilvitskii, 2017] and [Lykouris and Vassilvitskii, 2018] suggested to introduce machine learned predictions to improve the performance of online algorithms. The first one use a predictor oracle to improve revenue optimization in auctions by setting a good reserve (or minimum) price while the second develops the novel framework even more by introducing the notions of consistency and robustness. In the second work, the online caching problem with predictions is considered. Following these works, a series of learning augmented results appeared in various fields. Problems such as caching [Antoniadis *et al.*, 2020; Jiang *et al.*, 2020; Rohatgi, 2020], ski-rental [Anand *et al.*, 2020; Bamas *et al.*, 2020b; Gollapudi and Panigrahi, 2019; Wang and Li, 2020], clustering [Dong *et al.*, 2020] and others [Aleksandrov and Walsh, 2017; Hsu *et al.*, 2019; Lattanzi *et al.*, 2020; Mitzenmacher, 2020; Lee *et al.*, 2021] have been studied under the new setting. [Bamas *et al.*, 2020a] consider the problem of speed scaling with predictions and in a different work [Bamas *et al.*, 2020b] show how to incorporate predictions that advise the online algorithm about the next action to take using the primal-dual schema. [Purohit *et al.*, 2018] applied this novel setting to the classic ski rental problem and the non-clairvoyant scheduling on a single machine without release times. For the same problems, [Wei and Zhang, 2020] provide a set of non-trivial lower bounds for competitive analysis using machine-learned predictions. Focused on the single machine non-clairvoyant scheduling problem, [Im *et al.*, 2021] propose a new error measure for prediction quality and design scheduling algorithms under this measure.

Here, we extend the work on non-clairvoyant scheduling towards the two directions proposed by [Purohit *et al.*, 2018] about scheduling jobs on multiple machines as well as scheduling jobs with release times on a single machine.

2 Notations and Preliminaries

We consider a set of n jobs $\mathcal{J}$ to be scheduled either on a single machine or on m parallel machines. Each job $j \in \mathcal{J}$ is characterized by a *release time* r_j and an *actual* (real) processing time x_j. We assume that $x_j > 1$ for each job j. For each job, we have also a *predicted* value of its processing time, denoted by y_j. At the release time of a job, the algorithm is informed of the existence of this job as well as of the predicted value of its processing time. The algorithm learns the actual processing time of the job, only after assigning x_j units of time to the job and hence knows that the job has been completed. The completion time of a job j is denoted by C_j. In other words, our model is *non-clairvoyant*. We denote by η_j the error of the prediction for job j, i.e. $\eta_j = |y_j - x_j|$, and $\eta = \sum_j \eta_j$ is the total error of the input.

It is known that the optimal strategy for the objective of minimizing the sum of completion times in the clairvoyant case is to follow the *Shortest Processing Time First (SPT)*[Smith, 1956] rule when all release times are zero. When arbitrary release times are introduced in the model, the *Shortest Remaining Processing Time First (SRPT)* rule is optimal. In the SRPT algorithm, at each time t, the *active* job with the shortest remaining processing time is chosen to be executed. A job is removed from the active jobs, when it has received x_j units of processing time.

In our work, we want to find algorithms that use the predictions and perform as good as the SPT and SRPT algorithms, in other words optimal, when the predictions are accurate and in the same time not too bad when the predictions are wrong.

A Preferential Algorithm. In order to get the best out of each world, we will consider two algorithms. Let $\mathcal{A}$ be a consistent algorithm with a competitive ratio α, and $\mathcal{B}$ be a robust algorithm with a competitive ratio β. [Purohit *et al.*, 2018] call a non-clairvoyant scheduling algorithm *monotonic* if it has the following property.

Definition 1 (Monotonicity)**.** *Given two instances with identical inputs and actual job processing times $(x_1, \ldots, x_n)$ and $(x'_1, \ldots, x'_n)$ such that $x_j \leqslant x'_j$ for all j, the objective function value found by the algorithm for the first instance is no higher than that for the second.*

[Purohit *et al.*, 2018] give also the following lemma on how to combine the two aforementioned monotonic algorithms into a preferential one.

Lemma 1. [Purohit *et al.*, 2018] *Given two monotonic algorithms, $\mathcal{A}$ and $\mathcal{B}$, with competitive ratios α and β respectively for the minimum total completion time problem with preemptions, and a parameter $\lambda \in (0, 1)$, one can obtain an algorithm with competitive ratio* $\min\{\frac{\alpha}{\lambda}, \frac{\beta}{1-\lambda}\}$.

In what follows, we focus on designing a consistent algorithm when the predictions are good and a robust algorithm otherwise. We will use Lemma 1 to get a preferential algorithm. Proofs are omitted due to space limitation.

3 Single Machine

In this section, we deal with the scheduling problem with release dates on a single machine. We first present the *Shortest Remaining Predicted Processing Time First (SRPPT)*, and show that it is a consistent algorithm for the objective of minimizing the sum of completion times. In the Appendix, we give a simple analysis showing that the Round Robin algorithm 4-approximates the objective in the setting with release dates. Our analysis is based on the dual-fitting technique as proposed in [Garg *et al.*, 2019]. Note that this result has been very recently presented also in [Moseley and Vardi, 2022], but using a more tedious analysis based on potential functions. Finally, we combine the two results in Theorem 7.

3.1 A Consistent Algorithm

We distinguish jobs as overestimated, $\mathcal{V}$, and underestimated ones, $\mathcal{U}$, according to their predicted and actual processing times. More formally, a job is said to be *overestimated* if $y_j \geqslant x_j$. On the other hand, a job is *underestimated* if $y_j < x_j$. Finally, we denote the total error of each subset as $\eta^{\mathcal{V}} = \sum_{j\in\mathcal{V}} \eta_j$ for the overestimated jobs and $\eta^{\mathcal{U}} = \sum_{j\in\mathcal{U}} \eta_j$ for the underestimated ones.

Consider a schedule σ. Given a time t, we denote by $e_j^{[0,t)}$ the *elapsed* time of j, i.e. the amount of processing time units received from the beginning of the schedule. We call a job j *active* at time t if it has been released (i.e. $t \geqslant r_j$), and if its elapsed time is smaller than its actual processing time, $e_j^{[0,t)} < x_j$. We denote by $x_j(t) = x_j - e_j^{[0,t)}$ the *remaining processing time* of an active job j at time t, and by $y_j(t) = y_j - e_j^{[0,t)}$ its *remaining predicted processing time*. To distinguish the completion times of a job j in various schedules we write $C_j\big|_t^\sigma$, where σ denotes the schedule and t the time. When not necessary, the time subscript is omitted. In what follows, we may have multiple schedules for a specific time t. In such case, the schedules are created using different instances and hence the distinction between these schedules is clear from the context.

Algorithm

In the *Shortest Remaining Predicted Processing Time First (SRPPT)* algorithm, at each time t, the active job (or one of the active jobs) with the shortest remaining predicted processing time is chosen to be executed. A job is added to the active jobs at time r_j and it is removed from the active jobs after receiving x_j units of processing time. In other words, a job can be removed from the active jobs before receiving y_j units of processing time if it is overestimated. Similarly, if a job is underestimated, it remains active even if it is has received y_j units of time. In this case, at the moment an underestimated job receives y_j units of processing time, it will be executed until completion, i.e. until it has received x_j units of processing time, as the job with the highest priority, having $y_j(t) < 0$. Only one such job can exist at each time.

To analyze this algorithm, we propose a transformation from an optimal schedule to the schedule produced by the SRPPT algorithm. An optimal schedule can be obtained by running the SRPT algorithm using the actual processing times of the jobs. The transformation consists of two steps. The first step (Transformation step I below) takes place only once in the beginning and its purpose is to bound the cost inflicted by overestimated jobs in the objective function. The second step (Transformation step II below), happens iteratively over time starting at time 0. In this second step, we bound the cost of the underestimated jobs and in the same time we slightly refine the cost of the overestimated jobs in the objective function.

Transformation Step I

To bound how much overestimated jobs can increase the value of the objective function, we create an intermediate schedule, called $\sigma^{\mathcal{V}}$. Consider the auxiliary instance, I^{aux}, created as follows from an instance I. For the overestimated jobs of I, we include the predicted processing time in the auxiliary instance I^{aux} while for the underestimated jobs of I we include the actual processing time in I^{aux}, i.e. $I^{aux} = \{y_j : j \in \mathcal{V}\} \cup \{x_j : j \in \mathcal{U}\}$. Let σ^* be an optimal schedule of this instance, and let j be a job j in this schedule. This job may be preempted several times in the schedule. Denote by $\ell_j^1, \ell_j^2, \ldots, \ell_j^k$ the length of the partial execution of job j in σ^* (with $k \geqslant 1$ and $\sum_k \ell_j^k = x_j$). Underestimated jobs have the same length in the auxiliary instance as in the optimal schedule while overestimated jobs have bigger length in the auxiliary instance. Therefore, for the underestimated jobs, we keep the same length for each part. On the contrary, for the overestimated jobs, we change the length of only the last part and we increase it by its error, i.e. $\ell'^1_j = \ell_j^1, \ell'^2_j = \ell_j^2, \ldots, \ell'^k_j = \ell_j^k + \eta_j$, such as $\sum_k \ell'^k_j = y_j$. As a result the total length of the overestimated job is equal to the predicted value of I^{aux}. We now create the intermediate schedule $\sigma^{\mathcal{V}}$, by scheduling the jobs in the auxiliary instance in the same order as in the optimal schedule σ^*, using the partial lengths we created above.

The following lemma gives us a relation between the intermediate and the optimal schedule.

Lemma 2. *We have* $\sum_j C_j\big|^{\sigma^{\mathcal{V}}} \leqslant \sum_j C_j\big|^{\sigma^*} + n\eta^{\mathcal{V}}$.

Example of Step I

Consider the following instance. A job is described by a triplet $j = (r_j, x_j, y_j)$. $\mathcal{I} = \{1 = (0,1,3), 2 = (0,2,2), 3 = (3,5,3), 4 = (3,4,4), 5 = (4,1,1.5), 6 = (7,2,2), 7 = (7,3,1)\}$. The overestimated jobs are $\mathcal{V} = \{1,2,4,5,6\}$ while the underestimated jobs are $\mathcal{U} = \{3,7\}$. In Figure 1, you can first see the optimal schedule, σ^*, produced by the

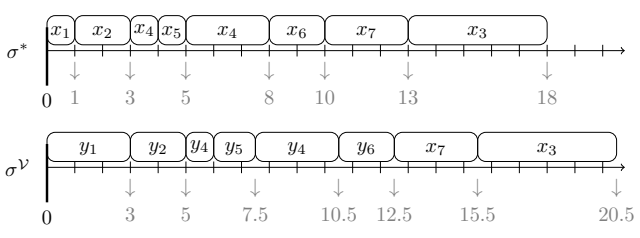


Figure 1: An example of the first step of the transformation.

SRPT algorithm and the values x_j. Finally, you can see the intermediate schedule, $\sigma^{\mathcal{V}}$, created as described in step I of the transformation.

Transformation Step II

Let σ^A be the schedule produced by the SRPPT algorithm on instance I. We create an initial auxiliary schedule, σ^{aux}, by applying the SRPT rule to the intermediate instance, I^{aux}, created in step I. Then, in what follows, we will transform this initial auxiliary schedule into the one produced by the SRPPT algorithm (σ^A), in a structured way that allow us to calculate the effect of each miss-predicted job in the objective function.

Before doing this, let us compare the sum of completion times in σ^{aux} and in $\sigma^{\mathcal{V}}$. We denote by $C_j|_{init}^{\sigma^{aux}}$ the completion time of job j in the initial schedule σ^{aux}. Since both schedules ($\sigma^{\mathcal{V}}$ and σ^{aux}) use the same instance, I^{aux}, and since σ^{aux} uses the optimal SRPT algorithm, we have:

$$\sum_j C_j|_{init}^{\sigma^{aux}} \leqslant \sum_j C_j|^{\sigma^{\mathcal{V}}} \tag{1}$$

Suppose an auxiliary schedule coincides with σ^A until time t. We denote such a schedule as σ_t^{aux}. We create subsets of the jobs as follows. Denote by $\mathcal{F}^{\mathcal{V}}$ and $\mathcal{F}^{\mathcal{U}}$ the overestimated and underestimated jobs, respectively, that finish execution in $[0,t)$. We also create a subset, denoted by $\mathcal{M}^{\mathcal{U}}$, containing the underestimated jobs that were miss-placed in $[0,t)$ (case 3 below). We define the reduced instance at time t as $I_t^{aux} = \{y_j(t) : j \in \mathcal{V}\} \cup \{0 : j \in \mathcal{F}^{\mathcal{V}}\} \cup \{x_j(t) : j \in \mathcal{U}\} \cup \{0 : j \in \mathcal{F}^{\mathcal{U}}\} \cup \{y_j(t) : j \in \mathcal{M}^{\mathcal{U}}\}$. The auxiliary schedule σ_t^{aux} consists of a fixed part which coincides with σ^A in the interval $[0,t)$ and part which is produced by executing the SRPT algorithm using the reduced instance I_t^{aux}.

Our transformation, starting at time $t = 0$, checks if the schedules σ^A and σ^{aux} coincide (i.e. schedule the same tasks) until time $t' > t$. If so, we augment the time until the time moment where the two schedules have the first difference. This difference may occur for one of the three following reasons.

1. An overestimated job is completed in σ^A
2. An underestimated job is completed in σ^{aux}
3. An underestimated job is misplaced in σ^{aux}

The transformation handles each case separately. Before that, we present Lemma 3 which has a two-fold meaning. On the one hand, it dictates when the second case arises while on the other hand, it rules out any other possibility for the two schedules to differ.

Lemma 3. *Let σ_t^{aux} be a schedule that coincides with σ^A until time t, where t is maximal. Let i be the job that starts or continues at time t in σ^A and j be the job that starts or continues at time t in σ_t^{aux}. If $i \neq j$, then i is underestimated.*

Given that σ^{aux} and σ^A coincide in the interval $[0,t)$, the transformation handles the aforementioned differing reasons accordingly.

- **Case 1.** For a job $j \in \mathcal{V}$, suppose l is the largest index of a part of a job that is executed in $[0,t)$. If $\sum_{i=1}^{l} \ell_j^i = x_j$, then remove all parts with indexes $i < l \leqslant k$ from the schedule. Remove job j from the overestimated set, $\mathcal{V}$, and add it to the finished overestimated set, i.e. $\mathcal{F}^{\mathcal{V}}$. Create a new auxiliary schedule by fixing the schedule at time interval $[0,t)$ and complete the rest of the schedule using the SRPT rule with the reduced instance. In this case the objective function decreases by at least η_j. Moreover, each overestimated job completed by the time t with $\eta_j > 0$ will create the transformation described in case 1.
- **Case 2.** For a job $j \in \mathcal{U}$. Suppose l is the largest index of a part of a job that is executed in $[0,t)$. If $\sum_{i=1}^{l} \ell_j^i = y_j$ and $\sum_k \ell_j^k = y_j$, then this is the last part of the job to be executed in $[0,t)$ and has finished execution using its predicted time. We can continue the execution of this job until time $t' = t + \eta_j$. Remove job j from the miss-placed underestimated set, $\mathcal{M}^{\mathcal{U}}$, and add it to the finished underestimated set , i.e. $\mathcal{F}^{\mathcal{U}}$. Create a new auxiliary schedule by fixing the schedule at time interval $[0,t')$ and complete the rest of the schedule using the SRPT rule with the reduced instance. In this case the objective function increases by at most $n\eta_j$. Moreover, if an underestimated job creates this transformation, then in latter time, it creates the transformation described in case 3.
- **Case 3.** The underestimated job j is scheduled in σ^A but not in σ^{aux}. This case arises when another job has shorter processing time than job j. On the other hand, SRPPT chose to execute job j, because its predicted processing time is shorter than any other job. Remove job j from the underestimated set, $\mathcal{U}$, and add it to the miss-placed underestimated set, $\mathcal{M}^{\mathcal{U}}$. This way, we enforce the SRPT algorithm to execute job j first. Create a new auxiliary schedule by fixing the schedule at time interval $[0,t)$ and complete the rest of the schedule using the SRPT rule with the new reduced instance. In this case the objective function decreases by at least η_j. Notice, a job that triggers this case, has already been called by case 2, resulting in a total difference of the $(n-1)\eta_j$ in the objective function.

Example of Step II

An example of step II of the transformation is illustrated in Figure 2. Starting from the initial auxiliary schedule, we notice that it coincides with σ^A until time instant 3, when an overestimated job finishes execution in σ_{init}^{aux} (Case 1). We remove the part of the job corresponding to the error and create a new auxiliary schedule, σ_3^{aux}, using the remaining instance and the SRPT rule, resulting in the third schedule of the figure. We then observe that an underestimated job, job 3, is misplaced in σ_3^{aux} (Case 3). In order to force the SRPT rule to place it correctly, we add job 3 in the $\mathcal{M}^{\mathcal{U}}$, and re-run the SRPT algorithm resulting in the fourth schedule in the figure. We next augment the time until time instant 5, where the schedules σ_5^{aux} and σ^A stop coinciding due to Case 1 again. By removing the error part, η_5, re-running SRPT with the reduced instance and augmenting the time, we come up with schedule σ_7^{aux}. Here, we come across Case 2 where an underestimated job is finished in σ_7^{aux}. We then run job 3 until completion, i.e. until it receives x_3 units of processing time and we augment the time resulting in σ_9^{aux}. Notice here, an-

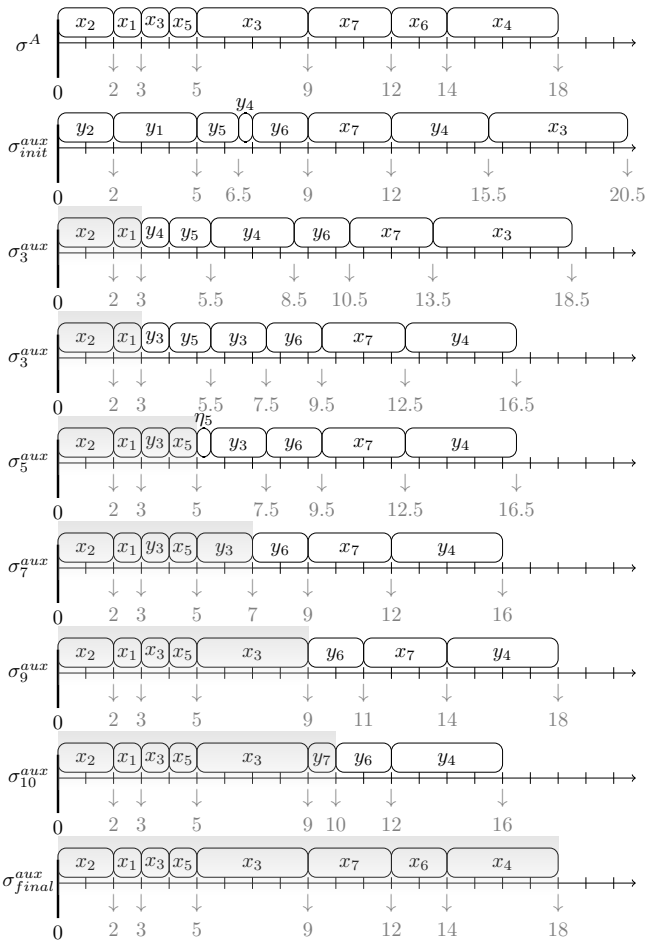


Figure 2: An example of the second step of the transformation.

other underestimated job is misplaced in σ^{aux}_9 (Case 2). We add job 7 in $\mathcal{M}^{\mathcal{U}}$ and re-run the SRPT algorithm. Augmenting the time of the common schedule to time instant 10, we observe that job 7 is finished without interruption. We arrive again at Case 2 where we fix the schedules to coincide until time 12. The final two jobs are correctly placed and exactly predicted, resulting in the last schedule of the figure, σ^{aux}_{final}.

Lemma 4. *We have* $\sum_j C_j\big|^{\sigma^A} \leqslant \sum_j C_j\big|^{\sigma^{aux}} - \eta^{\mathcal{V}} + (n-1)\eta^{\mathcal{U}}$.

Theorem 5. *The SRPPT algorithm has competitive ratio at most* $(1 + \frac{2\eta}{n})$*, where* $\eta = \sum_j \eta_j$.

Lemma 6. *Algorithm SRPPT is monotonic.*

3.2 A Preferential Algorithm

Theorem 7. *The* Preferential Round Robin *algorithm with release dates and parameter* $\lambda \in (0,1)$ *has competitive ratio at most* $\min\left\{\frac{1}{\lambda}(1+\frac{2\eta}{n}), \frac{4}{1-\lambda}\right\}$. *In particular, it is* $\frac{4}{1-\lambda}$*-robust and* $\frac{1}{\lambda}$*-consistent.*

4 Multiple Machines

In this section, we consider the problem of scheduling jobs on a set of m multiple machines using machine-learned predictions. Let $\mathcal{J}$ be an instance of n jobs with the execution time of each job being $x_j : 1 \leqslant j \leqslant n$. Each job has also a *predicted* execution time, $y_j : 1 \leqslant j \leqslant n$. Similarly to the previous section, our model is *non-clairvoyant*, meaning that the actual processing time is revealed to the algorithm only when x_j units of processing time are assigned to a specific job. In this section we consider the version of the problem where all jobs have release dates equal to 0. In what follows, jobs can be preempted and migrated to a different machine at no cost at any time. Our goal is to design a learning augmented algorithm taking into account the predictions under the objective of minimizing the total sum of completion times.

We first present the consistent algorithm *Shortest Predicted Processing Time First for Multiple Machines (SPPT(m))*. We show that SPPT(m) is optimal if the predictions are accurate. For this specific setting, Round Robin is shown to be 2- competitive in [Motwani *et al.*, 1994]. Finally, we combine the two results in Theorem 10.

4.1 A Consistent Algorithm

We define a grouping between the jobs using their predicted processing times.

Definition 2. *Let* $\mathcal{J}$ *be any instance of size* n *with the predicted processing times of the jobs being such that* $y_1 \geqslant y_2 \geqslant \ldots \geqslant y_n$ *and* j_i *refer to the job with predicted processing time* y_i*. For* $1 \leqslant k \leqslant \lceil \frac{n}{m} \rceil$*, define* $G_{Pr}(k)$ *of jobs to be* $G_{Pr}(k) = \{j_i | (k-1)m < i \leqslant km\}$.

Groups give the execution ordering of the jobs in each machine. A job of group 1 is scheduled on a machine in the last position, a job of group 2 is scheduled second to last on a machine, etc. All groups, except possibly the last one, consist of m consecutive jobs.

Algorithm

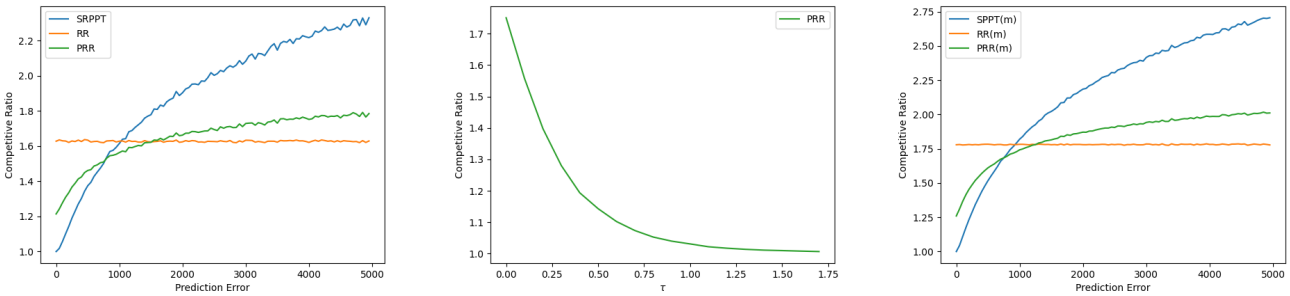
The SPPT(m) algorithm first creates $\lceil n/m \rceil$ groups $G_{Pr}(k)$ of the jobs, considering the jobs sorted in non-increasing order of the predicted values of their processing times (as seen above, $G_{Pr}(1)$ contains the m tasks with the largest predicted values, and so forth). Then, each machine receives (at most) one task of each group. On each machine, the tasks are scheduled in a non decreasing order of their predicted values, and the processing times of each task is its actual processing time.

Theorem 8. *The SPPT*(m) *algorithm has competitive ratio at most* $1 + \frac{2m\eta}{n}$*, where* $\eta = \sum_j \eta_j$.

Lemma 9. *Algorithm SPPT(m) is monotonic.*

4.2 A Preferential Algorithm

Theorem 10. *The* Preferential Round Robin for multiple machines *algorithm with parameter* $\lambda \in (0,1)$ *has competitive ratio at most* $\min\left\{\frac{1}{\lambda}(1+\frac{2m\eta}{n}), \frac{2}{1-\lambda}\right\}$. *In particular, it is* $\frac{2}{1-\lambda}$*-robust and* $\frac{1}{\lambda}$*-consistent.*



(a) Single machine, $\tau = 0.1$ (b) Single machine, $\sigma = 1000$ (c) Multiple machines

Figure 3: Experimental results

5 Experimental Results

In this section we present experimental results which confirm the bounds stated on Theorems 7 and 10 for the single and multiple machines respectively. The code is publicly available at: https://github.com/ildoge/SUP.

We create artificial instances, each one consisting of $n = 50$ jobs for the single machine case, following the same approach as in [Purohit *et al.*, 2018]. We draw the actual processing time values, x_j, independently for each job from a Pareto distribution with a parameter $\alpha = 1.1$. An error value, η_j, is drawn from a normal distribution with mean 0 and standard deviation σ. Finally, we set the predicted processing time of each job to be $y_j = x_j + \eta_j$. For the release times of the jobs, we first set the interval $[0, \tau \sum_j x_j]$ using the actual processing times created before, where $\tau \geqslant 0$ is a parameter that allows us to scale the size of the interval. We then draw n values uniformly at random from this interval and assign them to the n jobs. Notice that, when the release dates are dense (jobs become available close to each other), then there is not a lot of idle time in the schedule. On the contrary, if jobs become available sparsely, then both the algorithm and the optimal are forced to take similar scheduling decisions. For this reason, we set $\tau = 0.1$ for the first simulations.

We implement the algorithms: Round Robin (RR), Shortest Remaining Predicted Processing Time First (SRPPT) and Preferential Round Robin (PRR). We compare the performance of the algorithms for different values of error by parameterizing the σ of the normal distribution. We use values of σ that belong in $[0, 5000]$ using a step of 50. The performance of the algorithms is compared to the optimal Shortest Remaining Processing Time First (SRPT) algorithm using the actual processing values. The ratio of an algorithm is taken as the average over 2000 independent runs. As we see in Figure 3a, the SRPPT algorithm performs really well when the error is small enough, but deteriorates fast otherwise. We observe also the robust behavior of Round Robin independently of the total error. In between, we have our Preferential Round Robin algorithm, with $\lambda = 1/2$, which performs well for small values of error while remaining competitive to Round Robin when the error is big.

Notice that for $\sigma = 1000$, our Preferential Round Robin algorithm out-performs both the SRPPT and RR. By fixing this value of σ, we compare the performance of PRR for various values of the parameter τ (Figure 3b). As expected, the ratio of the algorithm deteriorates as the interval becomes smaller. Interestingly, we observe that we get the worst case ratios when all release times are equal to zero. On the other hand, when the interval is big and the release times sparsely distributed in it, our algorithm is nearly optimal.

For the multiple machines setting, we implement the algorithms: Round Robin (RR(m)), Shortest Predicted Processing Time First (SPPT(m)) and Preferential Round Robin (PRR(m)). We follow the same approach to produce the actual processing values, the error and the predicted processing values. In this case, we have no release times. Here, we follow the same framework of executions as before, however we compare each algorithm to the optimal Shortest Processing Time First (SPT(m)) algorithm for multiple machines. We note that we run these experiments using $m = 5$ machines and $n = 250$ jobs in order to maintain enough jobs and load in each machine. The results can be seen in Figure 3c and can be interpreted in the same way as the results of the single machine setting.

6 Conclusion

In this work, we studied the problem of integrating predictions to improve the performance of online algorithms for the non-clairvoyant scheduling problem on a single machine with release dates and on multiple machines without release dates. Using predictions to improve the performance of algorithms is a very versatile technique and can be applied to various fields. We are actively interested to find more state of the art algorithms that can adopt this approach. As a future work, we would like to find more efficient ways to combine algorithms as this will improve the trade-off between the consistent and the robust algorithms.

Acknowledgments

This work was partially supported by the French National Research Agency (Energumen ANR-18-CE25-0008 and Algoridam ANR-19-CE48-0016). The research of the third author was supported by the Russian Science Foundation RSF-ANR 21-41-09017.

References

[Aleksandrov and Walsh, 2017] Martin Aleksandrov and Toby Walsh. Most competitive mechanisms in online fair division. In *KI*, volume 10505 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2017.

[Anand *et al.*, 2020] Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ML predictions for online algorithms. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 303–313. PMLR, 2020.

[Antoniadis *et al.*, 2020] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020.

[Bamas *et al.*, 2020a] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.

[Bamas *et al.*, 2020b] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.

[Boyar *et al.*, 2017] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.

[Dong *et al.*, 2020] Yihe Dong, Piotr Indyk, Ilya P. Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *ICLR*. OpenReview.net, 2020.

[Garg *et al.*, 2019] Naveen Garg, Anupam Gupta, Amit Kumar, and Sahil Singla. Non-clairvoyant precedence constrained scheduling. In *ICALP*, volume 132 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Gollapudi and Panigrahi, 2019] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019.

[Hsu *et al.*, 2019] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *ICLR*. OpenReview.net, 2019.

[Im *et al.*, 2021] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *SPAA*, pages 285–294. ACM, 2021.

[Jiang *et al.*, 2020] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *ICALP*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Lattanzi *et al.*, 2020] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *SODA*, pages 1859–1877. SIAM, 2020.

[Lee *et al.*, 2021] Russell Lee, Jessica Maghakian, Mohammad H. Hajiesmaili, Jian Li, Ramesh K. Sitaraman, and Zhenhua Liu. Online peak-aware energy scheduling with untrusted advice. In *e-Energy ACM*, pages 107–123. ACM, 2021.

[Lykouris and Vassilvitskii, 2018] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018.

[Medina and Vassilvitskii, 2017] Andres Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *NeurIPS*, pages 1858–1866, 2017.

[Mitzenmacher, 2020] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, volume 151 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Moseley and Vardi, 2022] Benjamin Moseley and Shai Vardi. The efficiency-fairness balance of round robin scheduling. *Operations Research Letters*, 50(1):20–27, 2022.

[Motwani *et al.*, 1994] Rajeev Motwani, Steven J. Phillips, and Eric Torng. Non-clairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, 1994.

[Purohit *et al.*, 2018] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *NeurIPS, Montréal, Canada*, pages 9684–9693, 2018.

[Rohatgi, 2020] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, pages 1834–1845. SIAM, 2020.

[Smith, 1956] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[Wang and Li, 2020] Shufan Wang and Jian Li. Online algorithms for multi-shop ski rental with machine learned predictions. In *AAMAS*, pages 2035–2037. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[Wei and Zhang, 2020] Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *NeurIPS*, 2020.