Static Scheduling with Predictions Learned through Efficient Exploration

Hugo Richard*1, Flore Sentenac*1, Corentin Odic1, Mathieu Molina1,2, and Vianney Perchet1,3

1ENSAE, CREST, CNRS, GENES, Institut Polytechnique de Paris, France
2INRIA, France
3Criteo AI Lab, France

Abstract

A popular approach to go beyond the worst-case analysis of online algorithms is to assume the existence of predictions that can be leveraged to improve performances. Those predictions are usually given by some external sources that cannot be fully trusted. Instead, we argue that trustful predictions can be built by algorithms, while they run. We investigate this idea in the illustrative context of static scheduling with exponential job sizes. Indeed, we prove that algorithms agnostic to this structure do not perform better than in the worst case. In contrast, when the expected job sizes are known, we show that the best algorithm using this information, called Follow-The-Perfect-Prediction (FTPP), exhibits much better performances. Then, we introduce two adaptive explore-then-commit types of algorithms: they both first (partially) learn expected job sizes and then follow FTPP once their self-predictions are confident enough. On the one hand, ETC-U explores in "series", by completing jobs sequentially to acquire information. On the other hand, ETC-RR, inspired by the optimal worst-case algorithm Round-Robin (RR), explores efficiently in "parallel". We prove that both of them asymptotically reach the performances of FTPP, with a faster rate for ETC-RR. Those findings are empirically evaluated on synthetic data.

1 Introduction

Online algorithms are traditionally evaluated through competitive analysis [4]. To grasp this concept, consider the standard problem of ski rental, where a person goes skiing for an unknown number $n$ of days (depending on the weather,
say). Each day, she has two options, either renting the skis (for the same daily price \( c \)) or buying the skis, once and for all, at price \( B \). The optimal, offline, algorithm \( \text{OPT} \) would obviously be to buy the skis if and only if \( cn > B \), and the score of \( \text{OPT} \) is then \( C_{\text{OPT}}(n) = \min\{cn, B\} \) whereas the price paid by an online algorithm \( \text{ALG} \), that takes sequential decisions and is unaware of the input \( n \), is \( C_{\text{ALG}}(n) = cN_{\text{ALG}} + B \mathbb{1}\{N_{\text{ALG}} < n\} \) where \( N_{\text{ALG}} \) is the day where \( \text{ALG} \) would buy the skis. Notice that \( C_{\text{ALG}} \) can be random, if the buying-decision time is not deterministic.

Those notions can be formally defined in any online problem described by its input \( \eta \): \( C_{\text{OPT}}(\eta) \) still denotes the score of the optimal offline algorithm, while \( C_{\text{ALG}}(\eta) \) is the score of an online algorithm \( \text{ALG} \) over that same input. The performance of \( \text{ALG} \) over some class of input distributions \( D \) is then given by the competitive ratio (CR):

\[
\text{CR}(\text{ALG}) = \min_{D \in D} \frac{\mathbb{E}_{\eta \sim D}[C_{\text{ALG}}(\eta)]}{\mathbb{E}_{\eta \sim D}[C_{\text{OPT}}(\eta)]},
\]

where the expectation is with respect to the (possible) randomness of the algorithm and the input.

The historical approach to worse case competitive analysis, widely covered \cite{[4]}, is to assume the class of input distributions \( D \) encompasses all possible distributions, and that the algorithm has no a-priori knowledge of the data. This postulate has been questioned by a recent line of work considering algorithms using external predictions. Indeed, in many practical applications, some a-priori knowledge about the data can be leveraged to improve performances. For instance, when looking up a word in a dictionary, it is known that "alligator" appears towards the beginning whereas "wheel" will be towards the end. This knowledge can be used to design algorithms more efficient than binary search. This idea has been investigated in many classical online problems such as web page caching \cite{[22]}, ski rental \cite{[30] [27]}, set cover \cite{[1]}, matching \cite{[9]}, clustering \cite{[10]} or scheduling \cite{[16] [21] [19]}.

Typically, given predictions \( \hat{\eta} \) of the true input, most of these works aim at balancing consistency vs. robustness, i.e., designing algorithms whose performances are close to that of optimal offline algorithms if predictions are accurate, but that do not degrade too much if they are terrible. This approach assumes that predictions are given by some oracle, and cannot be trusted fully, hence the need for robustness to bad/incorrect predictions. The trade-off robustness vs. consistency in the prediction is oftentimes characterised by some parameter affecting the performance guarantees.

Instead of relying on an external prediction, we argue that algorithms can learn while they run. Self-learned predictions are by essence more and more accurate as the number of samples increases and therefore can be more trusted as the algorithm runs, in striking contrast with the previous approaches. For the sake of illustration, we shall focus our analysis on static job scheduling on a single machine. In this problem, a set of jobs indexed by \( i \in [N] \) must be processed on a machine. Each job \( i \) has a size \( P_i \), which is the amount of time
needed for the machine to complete it. An algorithm is a policy assigning jobs to the machine and the completion time $e_i$ of a job $i$ is the current time when this job finishes. The cost for an algorithm ALG, also called flow time, is the sum of all completion times:

$$E[C_{ALG}] = E \left[ \sum_{i \in [N]} e_i \right].$$

The objective is to minimize it with the constraint that only one job is processed at a time. Yet, preemption is allowed, i.e., changing processed jobs can be done at any time without additional cost (which implies that two jobs could be run in parallel for twice longer completion times).

For some learning to eventually occur, jobs are assumed to have known “types”. More precisely, there are $K$ different types and $n$ jobs per type, so that $N = nK$. Jobs of type $k \in [K]$ have sizes $(P_k^i)_{i \in [n]}$ drawn i.i.d. from the same exponential distribution $D_k$, parameterized by $\lambda_k$. Even though the family of exponential distributions is quite classical to describe job sizes [18, 13, 8, 6, 7, 26, 12], the purpose of those assumptions is to better illustrate the principles and concepts without dwelling too much into technical details.

**Contributions:** We first explain why we focus on exponential job sizes in the static scheduling problem. Indeed, we show that any algorithm unaware of the parameters of the exponential distributions $(D_k)_{k=1}^K$ and that ignores jobs types has the same asymptotic CR (i.e, 2) as in the adversarial case. This generalizes the lower bound in [25]. On the other side of the spectrum, stands the algorithm that has access to perfect predictions of the parameters of $(D_k)_{k=1}^K$ and schedules jobs with the smallest expected size first. This algorithm, called Follow-The-Perfect-Prediction (FTPP), is the optimal realization agnostic algorithm (see [5] Corollary 2.1). We study the variations of the CR of FTPP with respect to the problem’s parameters and show that FTPP has a CR always smaller than 2. Those two points advocate for learning while running.

Then, we consider algorithms learning the unknown parameters of $(D_k)_{k=1}^K$ by using known job types. We show that the algorithm Explore-Then-Commit with Uniform exploration (ETC-U) achieves asymptotically the same performances as FTPP with an expected difference in CR of order $O(\sqrt{\log(n)/n})$. This algorithm performs successive elimination, maintaining a set of types that are candidates to have the smallest mean. Those candidates are explored uniformly in "series", meaning that a job of each candidate type is selected successively and run until completion.

Lastly, Explore-then-Commit with Round-Robin based exploration (ETC-RR) is introduced. ETC-RR differs from ETC-U as it runs a job of each candidate types in "parallel". This strategy is reminiscent of Round Robin (RR), the optimal online algorithm for adversarial input. The CR of ETC-RR has the same asymptotic behavior in $n$, however, its second order term is smaller, with a lower dependency in the largest job types means than the one of ETC-U.
This demonstrates that structure can be leveraged to further improve learning algorithms.
Those findings are empirically evaluated on synthetic data.

2 Related work

Scheduling problems The scheduling literature and problem zoology is large. We focus on static scheduling on a single machine where the objective is to minimize the flow time. Possible generalizations include dynamic scheduling where jobs arrive at different times [3], weighted flow time [2] where different jobs have different weights, forbidden preemption [17, 13, 23] where jobs are not allowed to be stopped once they are started, multiple machines [20] and many more [11, 29]. The performance of algorithms depends on the available information.

Clairvoyant and non-clairvoyant scheduling: In clairvoyant scheduling, job sizes are assumed to be known. In this setting, scheduling the shortest jobs first gives the lowest flow time [28]. In non-clairvoyant scheduling, job sizes are arbitrary and unknown. The Round Robin (RR) algorithm, that gives the same amount of computing time to all jobs, is the best deterministic algorithm with a competitive ratio of $2 - \frac{2}{N+1} = 2 + o(N)$ [25]. The best randomized algorithm has a competitive ratio of $2 - \frac{1}{N+3} = 2 + o(N)$ [25].

Stochastic scheduling: Stochastic scheduling covers a middle ground where job sizes are known random variables. The field of optimal stochastic scheduling is about designing optimal algorithms for stochastic scheduling (see [5] for a review). When distributions have non-decreasing hazard rate, scheduling the shortest mean first is optimal (see [5] Corollary 2.1).

We consider exponential job sizes (which have a non-decreasing hazard rate) but in contrast to most of the literature on stochastic scheduling, the means are unknown to the algorithm and are learned as the algorithm runs. Exponential random variables for job sizes are very common in scheduling [18, 13, 8, 17, 24, 12] and similarly for the presence of different types of jobs [21, 13, 23].

Learning-augmented scheduling algorithms In static scheduling, the most natural quantity to predict is the job sizes. Bounds depending on the $\ell_1$ distance between the actual and predicted sizes are established in [27]. Measuring the error with the $\ell_1$ distance is problematic as it does not reflect the performance of the algorithm scheduling jobs in increasing order of their predicted sizes [16, 21]. Instead, in [21], authors propose to predict instead the relative order of jobs and they obtain tighter error-dependent bounds on the competitive ratio. They show that learning an order across jobs can in principle be done through empirical risk minimization providing a way to obtain the needed predictions. However, using empirical risk minimization supposes the existence of multiple static scheduling tasks with roughly the same job order. In contrast, we assume that jobs are typed based on their expected size and learn the order among types as the
algorithm runs. In some sense, we deal with cold-start augmented learning as no predictions are available before the algorithm starts.

3 Setting and Notations

We recall that the considered problem is formalized as follows. There are \( N \) jobs that can be of \( K \) different types to be scheduled on a single machine. We assume that \( N = nK \), i.e., there are \( n \) jobs of each type. The different sizes (also called processing times) of the jobs of type \( k \) are noted \((P^k_i)_{i \in [n]}\), where \((P^k_i)_{i \in [n]}\) are independent samples from an an exponential variable of parameter \( \lambda_k \) where we adopt the following convention:

\[
P^k_i \sim \mathcal{E} (\lambda_k) \implies \mathbb{E}[P^k_i] = \lambda_k.
\]

By extension, we call \( \lambda_k \) the mean size of type \( k \). We assume without loss of generality that the mean sizes of the \( K \) types are ordered: \( \lambda_1 \leq \lambda_2 \ldots \leq \lambda_K \) and we denote \( \mathbf{\lambda} = (\lambda_1, \ldots, \lambda_K) \). From now on, \( \text{OPT} \) denotes the optimal offline algorithm, that is aware of each job size realisations and computes them in increasing order.

4 Scheduling jobs with exponential sizes without learning

In this section, we study the two extreme cases. We start with the setting of full knowledge of all expected job sizes (Section 4.1) and then investigate the setting without prior information on expected job sizes (Section 4.2).

4.1 FTPP: Follow The Perfect Predictions - Structure can be leveraged

In this section, we assume that the mean sizes of all jobs are known or in other words that the algorithm has access, prior to seeing any jobs, to a perfect prediction. Follow-The-Perfect-Prediction (FTPP) is the algorithm that completes jobs by increasing expected sizes. It is the optimal strategy for an algorithm unaware of the realisations (see [5] Corollary 2.1). The study of FTPP illustrates the fact that the cases \( K = 1 \) and \( K > 1 \) are structurally and fundamentally different.

The next proposition yields an upper bound on the CR of FTPP:

**Proposition 1.** Assume that \( \lambda_1 \leq \cdots \leq \lambda_K \) with at least one strict inequality. Then the CR of FTPP is strictly lower than 2.

**Proof sketch (full proof in Appendix [B.1]).** Consider an algorithm \( A \) that schedules jobs in order \( \sigma \) such that for a job \( i \) of mean size \( \lambda_i \) and a job of type \( j \) of mean size \( \lambda_j \): \( \lambda_i < \lambda_j \implies P(\sigma(i) < \sigma(j)|i < j) > \frac{\lambda_i}{\lambda_i + \lambda_j} \). We show that \( A \) has
a CR smaller than 2 which entails that FTPP has a CR smaller than 2 as FTPP is the optimal algorithm in this setting.

Proposition 1 shows that, asymptotically, FTPP has a lower CR than RR, the best non-clairvoyant algorithm. In order to quantify this statement, let us be more precise. First, consider the case of \( K = 2 \) types of jobs with \( n \) jobs per type with \( \lambda_1 = 1 \) and \( \lambda_2 = \lambda > 1 \). In this case the asymptotic CR, as a function of \( \lambda \), is given by:

\[
\lim_{n \to \infty} \frac{E[C_{\text{FTPP}}]}{E[C_{\text{OPT}}]} = \frac{2(1 + \lambda)^2 + 4(1 + \lambda)}{(1 + \lambda)^2 + 4\lambda} = 2 - \frac{\lambda - 1}{(1 + \lambda)^2 + 4\lambda}.
\]

We refer the reader to Appendix B.2 for the detailed derivation. This function has value 2 at \( \lambda = 1 \), decreases to reach its minimum value \( \sqrt{2}/2 - 1 \approx 0.7 \) at \( \lambda^* = 1 + 2\sqrt{2} \) and then increases with a limit on the right equal to 2.

Then, we consider the general case:

**Proposition 2.** The asymptotic competitive ratio of FTPP for \( K \) types of job with mean sizes \( \lambda = (\lambda_k)_{k=1}^{K} \) with \( \lambda_1 \leq \cdots \leq \lambda_K \) is given by:

\[
\lim_{n \to \infty} \frac{E[C_{\text{FTPP}}]}{E[C_{\text{OPT}}]} = \frac{\sum_{k=1}^{K} (\frac{1}{2} + K - k)\lambda_k}{\sum_{k=1}^{K} \frac{1}{2}\lambda_k + \sum_{k=1}^{K} \lambda_k = \lambda_k = \lambda_{k+1}} = 2 - \frac{\sum_{k=1}^{K} \lambda_k \sum_{k=1}^{K} \frac{\lambda_k}{\lambda_{k+1}}}{\sum_{k=1}^{K} \frac{1}{2}\lambda_k + \sum_{k=1}^{K} \frac{\lambda_k}{\lambda_{k+1}}} =: \text{CR}_{\text{FTPP}}(\lambda, K).
\]

In addition \( \text{CR}_{\text{FTPP}}(K) \), the minimum value in \( \lambda \) of FTPP asymptotic competitive ratio, decreases in \( K \) and can be computed by solving a constrained optimization problem with strictly quasi-convex objective and compact convex constraints.

See Appendix B.3.1 for a proof.

We can upper-bound the limit as \( K \) increases (details are postponed to Appendix B.3.2) and obtain

\[
\lim_{K \to \infty} \text{CR}_{\text{FTPP}}(K) \leq \frac{4}{\pi} \approx 1.273,
\]

therefore the asymptotic CR of FTPP can reach much lower values than the asymptotic CR of RR.

Even in the average analysis where we suppose that the \( \lambda \) themselves are distributed according to some law, we show that the CR can be strictly lower than the CR of RR. We shall illustrate this by assuming that the \( \lambda_i \) are sampled from a uniform distribution on \([0, 1]\). Then, \( \text{CR}_{\text{FTPP}}(K, n) = \frac{E[C_{\text{FTPP}}]}{E[C_{\text{OPT}}]} \) is decreasing in \( K \) for \( n > 8 \), increasing in \( n \) and

\[
\lim_{K \to \infty} \text{CR}_{\text{FTPP}}(K, n) = 10/(11 - 8 \log(2)) \approx 1.629
\]
Detailed derivations are available in Appendix B.3.3.

Together Proposition 2 and Equation 1 and Equation 2 demonstrate that the gap between the best non-clairvoyant algorithm and the best algorithm aware of the distribution of the jobs is large. This motivates the design of efficient learning algorithms aiming at reaching the performance of FTPP.

4.2 Lower bound - Learning structure is crucial

A “non-clairvoyant” algorithm has no a-priori knowledge on the job sizes, not even the existence of types. There exists a lower bound of $2 - o(1)$ on the CR of any (deterministic or randomized) non-clairvoyant algorithm in the adversarial case [25]. This lower bound is attained when the input is $n$ jobs with exponential sizes, assuming they all have the same mean, which corresponds to $K = 1$ in the above notations. In the next proposition, we show that the lower bound remains of the same order, even if $K \geq 2$, which has inherently more structure as exposed in the previous section.

**Proposition 3.** In the limit of large $n$, any (deterministic or randomized) non-clairvoyant algorithm has competitive ratio at least $2 - o(1)$ on input $(P^k_i)_{i \in [n], k \in [K]}$.

**Proof sketch (full proof in Appendix A).** From [25], the total expected flow time of any deterministic algorithm $A$ is given by the sum of the time spent computing all jobs and the time lost waiting as jobs delay each other. We first show that, asymptotically, only the second matters.

Then, we consider $T^A_{ij}$, the amount of time that job $i$ and job $j$ delay each other. As the algorithm is unaware of the expected job size order, its run is independent of whether the expected size of job $i$ is smaller or greater than that of $j$. This holds because a non-clairvoyant algorithm has no information on job expected sizes and cannot learn them as it ignores that jobs of the same type have same distribution. As an adversary, we can therefore choose the order so that the algorithm incurs the largest flow time. A careful analysis then provides $E[T^A_{ij}] \geq 2E[T^OPT_{ij}]$ where OPT is the optimal realisation-aware algorithm. A similar reasoning is made in the case of randomized algorithms.

Proposition 3 states that, asymptotically and compared to the adversarial case, assuming that jobs have exponential size does not make the problem intrinsically easier. This proposition is easily extended to the more general setting where jobs do not have types (i.e when $k = N$). Note that this extension does not reduce to the adversarial setting studied in [25] as job sizes are sampled from an exponential distribution. We refer the reader to Appendix A for more details.

5 Learning Algorithms

We introduce in this section two different learning algorithms. The first one, ETC-U, naively and uniformly explores to estimate each type’s expected size.
The exploration strategy of the second one, ETC-RR, is inspired by the worst-case optimal algorithm, RR. We prove that the CR of both of them asymptotically matches the optimal one of FTPP. However, the second order term in the upper bound of ETC-RR is smaller than the one of ETC-U. Those findings are confirmed by empirical evidence on synthetic data (see Section 6).

5.1 Explore then commit with uniform exploration: ETC-U

Explore-Then-Commit with Uniform exploration (ETC-U) is a successive elimination algorithm, where the relative ranking of each type is learned through uniform exploration. Had the relative ranks of the means of each types been known, the optimal online algorithm would compute all the jobs with smallest expected mean first. However, that knowledge is not available, it is thus learned.

While ETC-U runs, it maintains a set of types \( A \) that are candidates for lowest mean size among the set \( U \) of types with at least one remaining jobs. At each iteration, ETC-U chooses a job of type \( k \) where \( k \) is the job of type \( A \) that has the lowest number of finished jobs. This job is executed until it finishes. Then \( U \) and \( A \) are updated and the procedure repeats until no more jobs are available.

At a given iteration, note \( m_k \) and \( m_\ell \) the number of jobs of type \( k \) and \( \ell \) that have been computed up to that iteration. Define:

\[
\hat{r}_{k,\ell}^{\min(m_k,m_\ell)} = \sum_{i=1}^{\min(m_k,m_\ell)} 1 \{ P^k_i < P^\ell_i \} \quad \text{and} \quad \delta_{k,\ell}^{\min(m_k,m_\ell)} = \sqrt{\frac{\log(2n^2)}{2 \min(m_k,m_\ell)}}.
\]

A type \( \ell \) is excluded from \( A \) if \( \ell \notin U \) or (Line 6) if there exists a type \( k \) such that

\[
\hat{r}_{k,\ell}^{\min(m_k,m_\ell)} - \delta_{k,\ell}^{\min(m_k,m_\ell)} > 0.5,
\]

which we show in Equation (19) (in appendix) implies \( \lambda_k < \lambda_\ell \) w.h.p. We say that type \( k \) eliminates type \( \ell \). This means job type \( \ell \) is no longer a candidate for the remaining job type with smallest expectation.

Note that since \( \hat{r}_{k,\ell} = 1 - \hat{r}_{k,\ell} \) and \( \delta_{k,\ell} = \delta_{\ell,k} \), if \( \hat{r}_{k,\ell}^{\min(m_k,m_\ell)} + \delta_{k,\ell}^{\min(m_k,m_\ell)} < 0.5 \), then \( 0.5 < \hat{r}_{\ell,k}^{\min(m_k,m_\ell)} - \delta_{\ell,k}^{\min(m_k,m_\ell)} \) and \( k \) is eliminated by \( \ell \).

Whenever \( A \) contains only one type, all jobs of this type are run until they are all finished. Whenever all jobs from type \( p \) are finished, type \( p \) is removed from \( U \) and therefore from \( A \). This means that types that were eliminated by \( p \) can be candidates again.

8
Algorithm 1 ETC-U

1. **Input**: $n \geq 1$ (number of jobs of each type), $K \geq 2$ (number of types)
2. For all pairs of different types $k, \ell$ initialize $\delta_{k, \ell} = 0$, $\hat{r}_{k, \ell} = 0$ and $h_{k, \ell} = 0$
3. For all types $k$, set $m_k = 0$
4. repeat
5. $U$ is the set of types with at least one remaining job
6. $A = \{ \ell \in U, \forall k \neq \ell, \hat{r}_{k, \ell} - \delta_{k, \ell} \leq 0.5 \}$
7. Select the type $\ell$ with the lowest number of finished jobs $\ell = \arg\min_{k \in A} m_k$ and run one job of type $\ell$ yielding a size $P_{m_{\ell} + 1}^\ell$.
8. $m_\ell = m_\ell + 1$
9. for $k, \ell$ in $U$, $k \neq \ell$ do
10. $h_{k, \ell} = \sum_{i=1}^{\min(m_k, m_\ell)} 1\{P_i^k < P_i^\ell\}$
11. $\delta_{k, \ell} = \sqrt{\log(2n^2) \min(m_k, m_\ell)}$
12. $\hat{r}_{k, \ell} = \frac{h_{k, \ell}}{\min(m_k, m_\ell)}$
13. end for
14. until $U$ is not empty

Proposition 4. The expected cost of ETC-U is upper bounded by:

$$E[C_{ETC-U}] \leq E[C_{FPPP}] + \sum_{k \in [K]} \left[ \frac{1}{2} (k-1)(2K-k) + (K-k)^2 \right] \lambda_k n \sqrt{8n \log(2n^2)} + \frac{K^3}{n} E[C_{OPT}]$$

Proof sketch (full proof in Appendix C.1). The proof starts by showing that w.h.p., the empirical quantities $\hat{r}$ concentrate around their means, Equation (19) (in appendix). This implies that w.h.p, the algorithm never takes the wrong decision of eliminating the remaining job type of smallest mean. We then bound the worse possible cost for the algorithm in the case where it takes wrong decisions and show it is always smaller than $2K E[C_{OPT}]$, Equation (21) (in appendix). Finally, we show that when the empirical quantities do concentrate around their expectation, the number of exploration steps before a sub-optimal job type is eliminated is bounded, Equation (23) (in appendix). This bounds the total cost of exploration.

5.2 Explore then commit with Round-Robin based exploration: ETC-RR

As ETC-U, ETC-RR is a successive elimination algorithm, except that the exploration is performed through a Round-Robin type procedure that runs many jobs "in parallel" while ETC-U was running jobs "in series".

As previously, ETC-RR maintains a set $A$ of types that are candidates to have the smallest mean size among types in $U$. During the exploration phase,
ETC-RR chooses one job for every type in \( \mathcal{A} \) and runs them in parallel. When a job finishes, if the job’s type is still active, it is replaced by a new job of the same type. Intuitively, this should diminish the cost of exploration when the chosen jobs have very different sizes: when \(|\mathcal{A}| \) jobs are processed in parallel, the smallest job (with size \( p_{\text{min}} \)) finishes first after an elapsed time \(|\mathcal{A}|p_{\text{min}} \). Thus, while the largest jobs runs (with size \( p_{\text{max}} \)), at least \( \lfloor p_{\text{max}}/p_{\text{min}} \rfloor \) smaller jobs may finish.

The statistics needed to construct \( \mathcal{A} \) are different from the one used in ETC-U. At a given time, \( h_{k,\ell} \) is the number of times a job of type \( k \) has finished while \( \ell \) and \( k \) were run in parallel. Define:

\[
\hat{r}_{k,\ell} = \frac{h_{k,\ell}}{h_{k,\ell} + h_{\ell,k}} \quad \text{and} \quad \delta_{k,\ell} = \sqrt{\frac{\log(2n^2)}{2(h_{k,\ell} + h_{\ell,k})}}.
\]

The rest of the algorithm is the same as ETC-U except that the elimination rule in Line 6 is now justified by Equation (25) (in appendix).

**Proposition 5.** The expected cost of ETC-RR is upper bounded by:

\[
\mathbb{E}[C_{\text{ETC-RR}}] \leq \mathbb{E}[C_{\text{FTPP}}] \sum_{\ell=1}^{K-1} (K-\ell)^2 n \lambda_{\ell} \sqrt{4n \log(2n^2)} + 2(K-1)^2 n \sum_{\ell=1}^{K-1} \lambda_{\ell} + 2K^3 n \mathbb{E}[C_{\text{OPT}}]
\]

*Proof sketch (full proof in Appendix C.2).* The technical arguments differ, but the general structure of the proof is the same as that of Proposition 4.

### 5.3 Comparison of the two algorithms

The competitive ratio of both algorithms is asymptotically the one of FTPP. Indeed, it always holds that \( \mathbb{E}[C_{\text{OPT}}] \geq \lambda_1 \frac{n^2}{2} \), we thus have, according to Propositions 4 and 5:

\[
\text{CR}_{\text{ETC-U}} = \text{CR}_{\text{FTPP}} + \mathcal{O} \left( \sqrt{\frac{\log(n)}{n}} \right) \quad \text{and} \quad \text{CR}_{\text{ETC-RR}} = \text{CR}_{\text{FTPP}} + \mathcal{O} \left( \sqrt{\frac{\log(n)}{n}} \right).
\]

On the one hand, the leading term in the cost is the same for both algorithms. On the other hand, the second order term can be much smaller in the case of algorithm ETC-RR.

To illustrate this claim, let us consider the case where there are two types of jobs. The jobs of type 1 have expected size \( \lambda_1 \) and the jobs of type 2 have expected size \( \lambda_2 \). Instantiating the bounds obtained of Propositions 4 and 5 to this setting we get:
Algorithm 2 ETC-RR

1: \textbf{Input :} \( n \geq 1 \) (number of jobs of each type), \( K \geq 2 \) (number of types),
2: For all pairs of different types \( k, \ell \) initialize \( \delta_{k,\ell} = 0, \hat{r}_{k,\ell} = 0 \) and \( h_{k,\ell} = 0 \)
3: For all types \( k \), set \( c_k = 0 \)
4: \textbf{repeat}
5: \( \mathcal{U} \) is the set of types with at least one remaining job
6: \( \mathcal{A} = \{ \ell \in \mathcal{U}, \forall k \neq \ell, \hat{r}_{k,\ell} - \delta_{k,\ell} \leq 0.5 \} \)
7: Run jobs \((P_{c_{k+1}}^k)_{k \in \mathcal{A}}\) in parallel until a job finishes and denote \( \ell \) the type of this job
8: \( c_{\ell} = c_{\ell} + 1 \)
9: for \( k \in \mathcal{A}, k \neq \ell \) do
10: \( h_{\ell,k} = h_{\ell,k} + 1 \)
11: \( \delta_{\ell,k} = \sqrt{\frac{\log(2n^2)}{2(h_{\ell,k} + h_{k,\ell})}} \)
12: \( \hat{r}_{\ell,k} = \frac{h_{\ell,k}}{h_{\ell,k} + h_{k,\ell}} \)
13: \( \hat{r}_{k,\ell} = \frac{h_{k,\ell}}{h_{k,\ell} + h_{\ell,k}} \)
14: end for
15: until \( \mathcal{U} \) is empty

\[
E[C_{\text{ETC-RR}}] \leq E[C_{\text{FTPP}}] + 2n\lambda_1(\sqrt{4n \log(2n^2)} + 1) + \frac{16}{n} E[C_{\text{OPT}}].
\] (4)

If \( \lambda_2 \gg \lambda_1 \) the bound in Equation (3) is much larger than the bound in Equation (4).

6 Experiments

In this section, we design synthetic experiments to compare ETC-U, ETC-RR, RR and FTPP. All code is written in Python. We use matplotlib [15] for plotting, and numpy [14] for array manipulations. The above libraries use open-source licenses. Computations are run on a laptop in less than 5 minutes.

In a first experiment, reported in Figure 1, algorithms are run with only \( K = 2 \) types and a number of jobs per type fixed to \( n = 2000 \). Jobs of type 1 have mean size \( \lambda_1 \) which varies between 0.02 and 1 and jobs of type 2 have \( \lambda_2 = 1 \).

As can be seen from the figure, the CR of ETC-U and ETC-RR is only slightly above the one of FTPP. On the other hand, both ETC-U and ETC-RR outperform RR. The CR of both learning algorithm vary similarly with \( \lambda_1 \), which is
consistent with the theoretical result stating that they have the same asymptotic performance. In particular, it can be seen that the theoretical value for the minimum of FTPP (plotted as a black dotted line) corresponds to what we observe in practice. As highlighted in Section 5.3, the cost of exploration in ETC-RR does not depend on $\lambda_2$ the highest mean size. Therefore, it is much lower when $\lambda_1$ is low. This can be seen from the figure where we can observe that for $\lambda_1 \in [0.02, 0.2]$, the CR of ETC-RR is almost confounded with the CR of FTPP whereas the CR of ETC-U is much higher. This difference diminishes as $\lambda_1$ grows.

In a second experiment, reported in Figure 2 (left graphic), the number of types $K$ varies while the number of jobs is fixed to $n = 2000$. The mean sizes $\lambda_k \in [K]$ are i.i.d. samples from a uniform distribution in $[0, 1]$.

As seen in Section 4, the expected CR of FTPP decreases with $K$. This behaviour is also observed in this experiment. As can be seen in the figure, for all values of $K$, ETC-RR outperforms ETC-U thanks to its efficient exploration. Both ETC-U and ETC-RR outperform RR.

In a third experiment, reported in Figure 2 (right graphic), the number of jobs per type $n$ varies while the number of types is fixed to $K = 3$. Like in the previous experiment, the mean sizes $\lambda_k \in [K]$ are i.i.d. samples from a uniform distribution in $[0, 1]$.

For small $n$, neither ETC-U nor ETC-RR have enough samples so that they can commit. Therefore ETC-U finishes one job of each type alternatively which is not as efficient as RR. In contrast ETC-RR follows a strategy similar to the one of RR yielding a similar competitive ratio even when no learning occurs.
When the number of samples increases, the gap between ETC-U, ETC-RR and FTPP decreases as the amount of time spent exploring decreases compared to the time spent exploiting.

Figure 2: CR on jobs with $K$ different types. On the left graphic, $n = 2000$ and $K$ varies. On the right graphic $K = 3$ and $n$ takes a grid of value. We take $\lambda_i \sim U(0,1)$. We report the median CR using 100 different seeds. Error bars represent the first and last decile.

Conclusion

We proved that online learning and online algorithms can be efficiently combined to produce algorithms that learn while they run. We developed this idea in the context of static scheduling with exponential processing time, a hypothesis that is shown not to make the problem easier. Two algorithms ETC-U and ETC-RR are introduced. Both of them yield the same asymptotic performance as if job mean sizes were known but ETC-RR, which takes inspiration from the best non-clairvoyant algorithm, has a much faster rate. These results are validated experimentally on synthetic datasets. Future work might focus on whether it is possible to combine our cold start approach with available predictions from external sources to achieve even more efficient exploration. Another possible extension would be to have job types with more structure (for instance, expected size could be an unknown linear mapping of some features representing the job). This theoretical work does not raise any obvious ethical issues.

Acknowledgment and funding disclosure

This work was supported by the French government under management of Agence Nationale de la Recherche as part of the “Investissements d’avenir” program, references ANR20-CE23-0007-01 FAIRPLAY project and LabEx Ecodec/ANR-11-LABX-0047 as well as from the grant number ANR-19-CE23-0026.
References

[1] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. Advances in Neural Information Processing Systems, 33:20083–20094, 2020. (Cited on page 2)

[2] Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. ACM Transactions on Algorithms (TALG), 3(4):39–es, 2007. (Cited on page 4)

[3] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. Journal of the ACM, 51(4):517–539, July 2004. (Cited on page 4)

[4] Allan Borodin and Ran El-Yaniv. Online computation and competitive analysis. cambridge university press, 2005. (Cited on page 1, 2)

[5] Xiaoqiang Cai, Xianyi Wu, and Xian Zhou. Optimal Stochastic Scheduling, volume 4. Springer, 2014. (Cited on page 3, 4, 5)

[6] Xiaoqiang Cai and Xian Zhou. Asymmetric earliness and tardiness scheduling with exponential processing times on an unreliable machine. Annals of Operations Research, 98(1):313–331, 2000. (Cited on page 3, 4)

[7] Xiaoqiang Cai and Xian Zhou. Single-machine scheduling with exponential processing times and general stochastic cost functions. Journal of Global Optimization, 31(2):317–332, 2005. (Cited on page 3, 4)

[8] Andrew A Cunningham and Sujit K Dutta. Scheduling jobs, with exponentially distributed processing times, on two machines of a flow shop. Naval Research Logistics Quarterly, 20(1):69–81, 1973. (Cited on page 3, 4)

[9] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. Advances in Neural Information Processing Systems, 34, 2021. (Cited on page 2)

[10] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In International Conference on Learning Representations, 2019. (Cited on page 2)

[11] Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An adversarial model for scheduling with testing. Algorithmica, 82(12):3630–3675, 2020. (Cited on page 4)

[12] Kevin D Glazebrook. Scheduling tasks with exponential service times on parallel processors. Journal of Applied Probability, 16(3):685–689, 1979. (Cited on page 4)

[13] Toshio Hamada and Kevin D Glazebrook. A bayesian sequential single machine scheduling problem to minimize the expected weighted sum of flowtimes of jobs with exponential processing times. Operations Research, 41(5):924–934, 1993. (Cited on page 3, 4)
[14] Charles R. Harris, K. Jarrod Millman, St’efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern’andez del R’ıo, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. (Cited on page 11)

[15] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007. (Cited on page 11)

[16] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-Clairvoyant Scheduling with Predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 285–294, Virtual Event USA, July 2021. ACM. (Cited on page 2, 4)

[17] Kevin Jeffay, Donald F Stanat, and Charles U Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *IEEE real-time systems symposium*, pages 129–139. US: IEEE, 1991. (Cited on page 4)

[18] Thomas Kämpke. Optimal scheduling of jobs with exponential service times on identical parallel processors. *Operations Research*, 37(1):126–133, 1989. (Cited on page 3, 4)

[19] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020. (Cited on page 2)

[20] Eugene L Lawler and Jacques Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM (JACM)*, 25(4):612–619, 1978. (Cited on page 4)

[21] Alexander Lindermayr and Nicole Megow. Non-Clairvoyant Scheduling with Predictions Revisited. *arXiv:2202.10199 [cs]*, February 2022. (Cited on page 2, 4)

[22] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018. (Cited on page 2)

[23] Sebastián Marbán, Cyriel Rutten, and Tjark Vredeveld. Learning in stochastic machine scheduling. In *International Workshop on Approximation and Online Algorithms*, pages 21–34. Springer, 2011. (Cited on page 4)

[24] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. (Cited on page 4)
[25] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994. (Cited on page 3, 4, 7)

[26] Michael Pinedo and Gideon Weiss. Scheduling jobs with exponentially distributed processing times and intree precedence constraints on two parallel machines. *Operations Research*, 33(6):1381–1388, 1985. (Cited on page 3, 4)

[27] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. *Advances in Neural Information Processing Systems*, 31, 2018. (Cited on page 2, 4)

[28] Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968. (Cited on page 4)

[29] Lai Tsung-Chyan, Yu N Sotskov, N Yu Sotskova, and F Werner. Optimal makespan scheduling with given bounds of processing times. *Mathematical and Computer Modelling*, 26(3):67–86, 1997. (Cited on page 4)

[30] Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. *Advances in Neural Information Processing Systems*, 33:8042–8053, 2020. (Cited on page 2)
A Lower bound: Proof of Proposition 3

Let us order all jobs $i \in [N]$ in order of their increasing expected size. Denote $P_i$ the size of job $i$. Notation $P_i$ and $P_i^{[i/n]} \mod n$ denote the same job, the first one is used in this proof for convenience. Any algorithm has cost:

$$\mathbb{E}[C^A] = \sum_{i=1}^{N} \mathbb{E}[P_i] + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbb{E}[T_{ij}^A]$$

where $T_{ij}^A = D_{ij}^A + D_{ji}^A$ where $D_{ij}^A$ is the amount of time job $i$ delay job $j$.

To prove Proposition 3, we first argue (Lemma 1) that that in the limit $n \rightarrow \infty$,

$$\sum_{i=1}^{N} \mathbb{E}[P_i] = o \left( \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbb{E}[T_{ij}^{OPT}] \right)$$

and then that $\mathbb{E}[T_{ij}^A] \geq 2\mathbb{E}[T_{ij}^{OPT}]$ where OPT is the optimal offline algorithm (Lemma 2).

Lemma 1. As $n \rightarrow \infty$,

$$\sum_{i=1}^{N} \mathbb{E}[P_i] = o \left( \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbb{E}[T_{ij}^{OPT}] \right).$$

Proof of Lemma 1. On one hand,

$$\sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbb{E}[T_{ij}^{OPT}] \geq \sum_{\ell=1}^{K} \sum_{i=(\ell-1)n+1}^{\ell n} \sum_{j=i+1}^{\ell n} \mathbb{E}[T_{ij}^{OPT}]$$

$$= \sum_{\ell=1}^{K} \sum_{i=(\ell-1)n+1}^{\ell n} \sum_{j=i+1}^{\ell n} \lambda_\ell$$

$$= \sum_{\ell=1}^{K} \frac{n(n-1)}{2} \lambda_\ell$$

$$\sim_{n \rightarrow \infty} \frac{n^2}{2} \sum_{\ell=1}^{K} \lambda_\ell.$$

On the other hand

$$\sum_{i=1}^{N} \mathbb{E}[P_i] = n \sum_{\ell=1}^{K} \lambda_\ell$$

$\square$

In this second lemma, we consider the more general setting where $K = N.$
Lemma 2. Consider $K = N$ jobs where job $i \in [N]$ has mean size $\lambda_i$ and $\lambda_1 \leq \cdots \leq \lambda_N$. Consider any algorithm $A$ and let $T_{ij}^A$ the total amount of time spent by $A$ on $i$ or $j$ while both jobs are alive.

$$E[T_{ij}^A] \geq 2E[T_{ij}^{OPT}]$$

where OPT is the optimal offline algorithm.

Proof of Lemma 2. Let us first prove our proposition for any deterministic algorithm $A$. We denote $i(t)$ amount of time that $A$ allocates to job $i$ after a time $t < T_{ij}^A$ is allocated to job $i$ or $j$.

$$E[T_{ij}^A] = \int_{t=0}^{+\infty} P(T_{ij}^A \geq t) dt$$

$$= \int_{t=0}^{+\infty} P(P_i \geq i(t)) P(P_j \geq t - i(t)) dt$$

$$= \int_{t=0}^{+\infty} \exp\left(-\frac{i(t)}{\lambda_i}\right) \exp\left(-\frac{t - i(t)}{\lambda_j}\right) dt$$

$$= \int_{t=0}^{+\infty} \exp\left(-\frac{i(t) + t/2 - t/2}{\lambda_i}\right) \exp\left(-\frac{t - (i(t) + t/2 - t/2)}{\lambda_j}\right) dt$$

$$= \int_{t=0}^{+\infty} \exp\left(-\left(\frac{1}{\lambda_i} + \frac{1}{\lambda_j}\right)\frac{t}{2}\right) \exp\left(-\left(\frac{1}{\lambda_i} - \frac{1}{\lambda_j}\right)(i(t) - \frac{t}{2})\right) dt.$$

Calling $f(t) = \exp\left(-\left(\frac{1}{\lambda_i} + \frac{1}{\lambda_j}\right)\frac{t}{2}\right)$ and $g(t) = |\frac{1}{\lambda_i} - \frac{1}{\lambda_j}|(i(t) - \frac{t}{2})$ it holds that either

$$\int_{t=0}^{+\infty} f(t) \exp(-g(t)) dt \geq \int_{t=0}^{+\infty} f(t) dt$$

or

$$\int_{t=0}^{+\infty} f(t) \exp(g(t)) dt \geq \int_{t=0}^{+\infty} f(t) dt$$

. Otherwise we would have

$$\int_{t=0}^{+\infty} f(t)\frac{1}{2}(\exp(-g(t)) + \exp(g(t))) dt < \int_{t=0}^{+\infty} f(t) dt$$

which cannot be true since $\forall t, \frac{1}{2}(\exp(-t) + \exp(t)) \geq 1$.

Therefore an adversary knowing $i(t)$ can always chose the order of $\lambda_i$ and $\lambda_j$ such that

$$E[T_{ij}^A] \geq \int_{t=0}^{+\infty} \exp\left(-\left(\frac{1}{\lambda_i} + \frac{1}{\lambda_j}\right)\frac{t}{2}\right) dt = 2\frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j}$$

The optimal delay is

$$E[T_{ij}^{OPT}] = E[\min(P_i, P_j)] = \frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j}$$
so our Lemma is proven for any deterministic algorithm $A$.

Consider a randomized algorithm $R$ which can be seen as a probabilistic distribution over the set of deterministic algorithm. Therefore $A, i(t)$ and $g(t)$ are now seen as random variables. By the tower rule, the amount of time job $i$ and $j$ delay each other in $R$ is such that:

$$\mathbb{E}[T_{ij}^R] = \mathbb{E}[\mathbb{E}[T_{ij}^A | A]]$$

$$= \mathbb{E}\left[ \int_{t=0}^{+\infty} f(t) \exp(\text{sign}(\lambda_i - \lambda_j)g(t)) dt \right]$$

By the same argument as in the deterministic case, it holds that either

$$\mathbb{E}\left[ \int_{t=0}^{+\infty} f(t) \exp(-g(t)) dt \right] \geq \int_{t=0}^{+\infty} f(t) dt$$

or

$$\mathbb{E}\left[ \int_{t=0}^{+\infty} f(t) \exp(g(t)) dt \right] \geq \int_{t=0}^{+\infty} f(t) dt$$

Otherwise we would have

$$\mathbb{E}\left[ \int_{t=0}^{+\infty} f(t) \frac{1}{2} (\exp(-g(t)) + \exp(g(t))) dt \right] < \int_{t=0}^{+\infty} f(t) dt$$

which implies that there exist a deterministic function $g$ such that

$$\int_{t=0}^{+\infty} f(t) \frac{1}{2} (\exp(-g(t)) + \exp(g(t))) dt < \int_{t=0}^{+\infty} f(t) dt$$

which cannot be true as shown in the deterministic case. The rest of the argument is the same as in the deterministic case and therefore omitted. \(\square\)

Now we are ready to prove Proposition 3.

**Proof of Proposition 3.** Take any algorithm $A$

$$\mathbb{E}[C_A] = \sum_{i=1}^{N} \mathbb{E}[P_i] + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbb{E}[T_{ij}^A]$$

(5)

$$\geq \sum_{i=1}^{N} \lambda_i + 2 \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbb{E}[T_{ij}^{OPT}]$$

(6)

$$\sim_{n \to \infty} 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[T_{ij}^{OPT}]$$

(7)

where (6) comes from Lemma 2 and (7) comes from Lemma 1.

From Lemma 1:

$$\mathbb{E}[C_{OPT}] \sim_{n \to \infty} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[T_{ij}^{OPT}]$$

\(\square\)
The generalization to $K = N$, can be carried on by replacing Lemma [1] by the following Lemma:

**Lemma 3.** Consider $K = N$ jobs where job $i \in [N]$ has mean size $\lambda_i$ and $\lambda_1 \leq \cdots \leq \lambda_N$. As $N \to \infty$, assuming $\lambda_N = o(N)$

$$\sum_{i=1}^{N} \mathbb{E}[P_i] = o \left( \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbb{E}[T_{ij}^{OPT}] \right)$$

**Proof.** We wish to show that as $N \to \infty$,

$$\sum_{i=1}^{N} \lambda_i = o \left( 2 \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j} \right)$$

Which is equivalent to

$$\sum_{i=1}^{N} \lambda_i = o \left( 2 \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j} \right)$$

and using the symmetry of $\frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j}$ the previous expression is equivalent to

$$\sum_{i=1}^{N} \lambda_i = o \left( 2 \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j} \right)$$

Using the fact that $\forall i, \lambda_i \in [1, M]$

$$\frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j} \geq \frac{\max(\lambda_i, \lambda_j)}{1 + M}$$

So in particular we have

$$\frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j} \geq \frac{\lambda_i}{1 + M}$$

Therefore, it becomes sufficient to prove

$$\sum_{i=1}^{N} \lambda_i = o \left( \sum_{i=1}^{N} \sum_{j \in [N], j \neq i} \frac{\lambda_i}{1 + M} \right)$$

$$\iff \sum_{i=1}^{N} \lambda_i = o \left( \sum_{i=1}^{N} (N - 1) \frac{\lambda_i}{1 + M} \right)$$

$$\iff 1 = o \left( \frac{N - 1}{1 + M} \right)$$

which is true since $M = o(N)$. \hfill \qed
B FTPP: Follow the perfect prediction

In this section, for all jobs $i \in [N]$, we call $P_i$ the job size of job $i$. Jobs are ordered in increasing order of their expected size (Notation $P_i$ and $P_{\lceil i/n \rceil}$ mod $n$ denote the same job).

B.1 CR lower than 2: Proof of Proposition 1

Let’s work directly in the more general setting $K = N$, $n = 1$. Assume $\lambda_1 \leq \cdots \leq \lambda_m$. Consider an algorithm $A$ that schedules jobs in order $\sigma$ such that for a job $i$ of mean $\lambda_i$ and a job of type $j$ of mean $\lambda_j$:

$$
\lambda_i \neq \lambda_j \implies P(\sigma(i) < \sigma(j)|i < j) > \frac{\lambda_j}{\lambda_i + \lambda_j}.
$$

We show that $A$ has a CR smaller than 2 which entails that FTPP has a CR smaller than 2 as FTPP is the optimal algorithm in this setting.

Proof. Denoting $T^A_{i,j}$ the amount of time job $i$ and $j$ delay each other, i.e. the amount of time spend on either of them before at least one is done computing:

$$
E[C_A] = \sum_i \lambda_i + \sum_{i<j} E[T^A_{i,j}]
$$

$$
= \sum_i \lambda_i + \sum_{i<j} \left( \lambda_j P(\sigma(j) < \sigma(i)|i < j) + \lambda_i P(\sigma(i) < \sigma(j)|i < j) \right)
$$

$$
< \sum_i \lambda_i + \sum_{i<j} \left( \lambda_j \left(1 - \frac{\lambda_j}{\lambda_i + \lambda_j}\right) + \lambda_i \frac{\lambda_j}{\lambda_i + \lambda_j} \right)
$$

$$
= \sum_i \lambda_i + \sum_{i<j} 2 \frac{\lambda_i \lambda_j}{\lambda_i + \lambda_j}
$$

$$
= 2E[C_{OPT}]
$$

\[\square\]

B.2 Asymptotic CR with $K = 2$

We show that with $K = 2$ types of jobs with $n$ jobs per type with $\lambda_1 = 1$ and $\lambda_2 = \lambda > 1$, the asymptotic CR is a function of $\lambda$ given by:

$$
\lim_{n \to \infty} \frac{E[C_{FTPP}]}{E[C_{OPT}]} = \frac{2(1 + \lambda)^2 + 4(1 + \lambda)}{(1 + \lambda)^2 + 4\lambda} = 2 - 4\frac{\lambda - 1}{(1 + \lambda)^2 + 4\lambda}.
$$
\textit{Proof.} The expected cost of OPT is given by

\[ E[C_{\text{OPT}}] = \sum_{i=1}^{2n} \sum_{j=i}^{2n} \mathbb{E}[\min(P_i, P_j)] \]

\[ = \sum_{i=1}^{n} \sum_{j=i}^{n} (\mathbb{E}[\min(P_i^1, P_j^1)] + \mathbb{E}[\min(P_i^2, P_j^2)]) + \sum_{i=1}^{n} \sum_{j=1}^{i} \min(P_i^1, P_j^2) \]

\[ = n(n+1) \frac{1}{2} + n^2 \frac{\lambda}{1+\lambda} + \frac{n(n+1) \lambda}{2} \quad (8) \]

since if \( X \sim \mathcal{E}(\lambda_1) \) and \( Y \sim \mathcal{E}(\lambda_2) \) then \( \mathbb{E}[\min(X,Y)] = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \).

Knowing the value of \( \lambda_1 \) and \( \lambda_2 \), FTPP will schedule the jobs with the shortest mean \( \lambda_1 \) first and then the jobs with mean \( \lambda_2 \) yielding a cost:

\[ E[C_{\text{FTPP}}] = \sum_{i=1}^{n} (2n - i + 1) \mathbb{E}[P_i^1] + \sum_{i=1}^{n} (n - i + 1) \mathbb{E}[P_i^2] \]

\[ = \frac{n(n+1)}{2} + n^2 + \frac{n(n+1) \lambda}{2}. \]

This with Equation (8) gives that the competitive ratio of FTPP is

\[ \frac{E[C_{\text{FTPP}}]}{E[C_{\text{OPT}}]} = \frac{\frac{n(n+1)}{2} + n^2 + \frac{n(n+1) \lambda}{2}}{\frac{n(n+1)}{2} + n^2 + \frac{n(n+1) \lambda}{1+\lambda} + \frac{n(n+1) \lambda}{2}} \]

As \( n \) grows large we have

\[ \lim_{n \to \infty} \frac{E[C_{\text{FTPP}}]}{E[C_{\text{OPT}}]} = \frac{\frac{1}{2}(1+\lambda) + \frac{1}{4}(1+\lambda) + \frac{1}{2} \lambda}{\frac{1}{2}(1+\lambda) + \frac{1}{4} \lambda} \]

\[ = \frac{2(1+\lambda)^2 + 4(1+\lambda)}{(1+\lambda)^2 + 4\lambda} =: f(\lambda). \]

\[ \square \]

\textbf{B.3 Asymptotic CR with \( K \) types}

\textbf{B.3.1 Expression, convexity and decreasing minimum: Proof of Proposition 2}

\textit{Proof.} We reuse the notation \( T_{t,q}^A \) for \((t, q) \in [N]^2\) to describe the amount of time job \( q \) and job \( t \) delay each other under algorithm \( A \). Let us express the expected cost of any algorithm in terms of \( \mathbb{E}[T_{t,q}^A] \) for \( k \in [K] \):
\[ \mathbb{E}[C_A] = \mathbb{E} \left[ \sum_{t=1}^{N} P_t + \sum_{t=1}^{N} \sum_{q=t+1}^{N} T_{t,q}^A \right] \]

\[ = \sum_{t=1}^{K} \mathbb{E}[P_t] + \sum_{t=1}^{K} \sum_{q=t+1}^{K} \mathbb{E}[T_{t,q}^A] \]

\[ = \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} \mathbb{E}[P_t] + \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} \left[ \left( \sum_{\ell=k+1}^{K} \sum_{q=(\ell-1)n+1}^{\ell n} \mathbb{E}[T_{t,q}^A] \right) \right] \]

\[ + \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} \mathbb{E}[T_{t,q}^A] \]

Instantiating Equation 9 with \( A = \text{FTPP} \):

\[ \mathbb{E}[C_{\text{FTPP}}] = \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} \lambda_k + \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} \left( \left( \sum_{\ell=k+1}^{K} \sum_{q=(\ell-1)n+1}^{\ell n} \lambda_k \right) + \sum_{q=t}^{kn} \lambda_k \right) \]

\[ = \sum_{k=1}^{K} n \lambda_k + \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} (nK - t) \lambda_k \]

\[ = \sum_{k=1}^{K} \left( n \lambda_k + \sum_{t'=1}^{n} (nK - k + n - t') \lambda_k \right) \]

\[ = \sum_{k=1}^{K} \left( \frac{n(n+1)}{2} \lambda_k + n^2(K-k) \lambda_k \right). \quad (10) \]

Instantiating Equation 9 with \( A = \text{OPT} \):

\[ C_{\text{OPT}} = \sum_{k=1}^{K} n \lambda_k + \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} \left( \left( \sum_{\ell=k+1}^{K} \sum_{q=(\ell-1)n+1}^{\ell n} \frac{\lambda_k \lambda_\ell}{\lambda_k + \lambda_\ell} \right) + \sum_{q=t}^{kn} \lambda_k \right) \]

\[ = \sum_{k=1}^{K} n \lambda_k + \sum_{k=1}^{K} \sum_{\ell=k+1}^{K} \sum_{t=(k-1)n+1}^{kn} n^2 \frac{\lambda_k \lambda_\ell}{\lambda_k + \lambda_\ell} + \sum_{k=1}^{K} \sum_{t=(k-1)n+1}^{kn} (kn - t) \frac{\lambda_k}{2} \]

\[ = \sum_{k=1}^{K} \left( n \lambda_k + n^2 \sum_{\ell=k+1}^{K} \frac{\lambda_k \lambda_\ell}{\lambda_k + \lambda_\ell} + \sum_{t'=1}^{n} (n - t') \frac{\lambda_k}{2} \right), \quad \text{with } t' \leftarrow t - (k-1)n \]

\[ = \sum_{k=1}^{K} \left( \frac{n(n+3)}{2} \lambda_k + n^2 \sum_{\ell=k+1}^{K} \frac{\lambda_k \lambda_\ell}{\lambda_k + \lambda_\ell} \right) \quad (11) \]
We define $CR_{FTPP}(\lambda, n) = \sum_{k=1}^{K} \frac{n(n+1)}{2} \lambda_k + n^2(K-k)\lambda_k$ \[\sum_{k=1}^{K} \left( \frac{n(n+1))}{2} \lambda_k + n^2 \sum_{k=1}^{K} \frac{\lambda_k \lambda_{k+1}}{\lambda_k + \lambda_{k+1}} \right) \] \[\rightarrow_{n \to \infty} \frac{\sum_{k=1}^{K} (\frac{1}{2} + K - k)\lambda_k}{\sum_{k=1}^{K} \left( \frac{1}{2} \lambda_k + \sum_{k=1}^{K} \frac{\lambda_k \lambda_{k+1}}{\lambda_k + \lambda_{k+1}} \right)} := CR_{FTPP}(\lambda). \] (13)

We define $CR_{FTPP}(K) = \inf_{\lambda \in D_K} CR_{FTPP}(\lambda)$ where $D_k = \{ \lambda \in \mathbb{R}^K \mid 0 \leq \lambda_1 \leq \ldots \leq \lambda_K \} \setminus \{0\}$.

We have that for any $\alpha \in \mathbb{R}^*$, $CR_{FTPP}(\alpha \lambda, n) = CR_{FTPP}(\lambda, n)$. We can thus re-scale every $\lambda_k$ by $\lambda_K$ when searching for the inf, and we can do this as for any sequence in $D_K$, $\lambda_K > 0$ (otherwise by the constraints $\lambda = 0 \notin D_K$). We fix $\lambda_K = 1$. Therefore we have $CR_{FTPP}(k) = \inf_{\lambda \in D'_K} \varphi(\lambda)$ with $D'_K = \{ \lambda \in \mathbb{R}^{K-1} \mid 0 \leq \lambda_1 \leq \ldots \leq \lambda_{K-1} \leq 1 \}$ and with

$$\varphi_K(\lambda) = \frac{\frac{1}{2} + \sum_{k=1}^{K-1} (\frac{1}{2} + K - k)\lambda_k}{\frac{1}{2} + \sum_{k=1}^{K-1} \left( \frac{1}{2} \lambda_k + \frac{\lambda_k}{\lambda_{k+1}} + \sum_{k=1}^{K-1} \frac{\lambda_k \lambda_{k+1}}{\lambda_k + \lambda_{k+1}} \right)}.$$ (14)

This function is well defined on $D'_K$ even when any or all $\lambda_i = 0$. This is because for $x \geq 0$ and $y \geq 0$, $0 \leq xy/(x+y) \leq \min(x,y)/2$ has a limit when $(x,y) \to (0,0)$ by squeeze theorem (note that it is not the case if $x$ or $y$ can take negative values).

The set $D'_K$ is bounded and closed hence compact. The constraints are linear, therefore $D'_K$ is also convex. The function $\varphi_K$ is continuous on $D'_K$, and thus admits a minimum. With Lemma 4 we know that $\varphi_K$ is strictly quasi-convex on $D'_K$, which means that the minimum is unique and we will denote it by $\lambda^*_K$. All in all, finding $CR_{FTPP}(K)$ can be reduced to solving an almost convex optimization problem (with the objective being strictly quasi convex instead of convex).

Now to see that $CR_{FTPP}(K)$ is decreasing in $K$, consider the point $\hat{\lambda}^{K+1} = (0, \lambda_1^{K}, \ldots, \lambda_K^{K+1})$ which is in $D'_{K+1}$. We have that $\varphi_{K+1}(\hat{\lambda}^{K+1}) = \varphi_K(\lambda^{*,K}) = CR_{FTPP}(K)$. Hence $CR_{FTPP}(K+1) \leq \varphi_{K+1}(\hat{\lambda}^{K+1}) = CR_{FTPP}(K) \square$

**Lemma 4.** The function $\varphi_K$ is strictly quasi-convex on $D'_K$.

**Proof.** We will first show that the denominator of $\varphi_K$ is strictly concave and strictly positive.

We have that $f : x \mapsto x/(1+x)$ is strictly concave on $[0,1]$. Indeed $f''(x) = -2/(1+x)^3 < 0$.

We will show that $f : (x, y) \mapsto xy/(x+y)$ is concave on $(0,1]^2$. We will compute the partial derivatives, to then compute the Hessian $H(x, y)$. For $(x, y) \in (0,1]^2$:

$$\begin{align*}
\frac{\partial f(x, y)}{\partial x} &= \frac{y(x+y)-(x+y)}{(x+y)^2} = \frac{y^2}{(x+y)^2} \\
\frac{\partial f(x, y)}{\partial y} &= \frac{x(x+y)-(x+y)}{(x+y)^2} = \frac{x^2}{(x+y)^2}
\end{align*}$$

(15)

24
Thus
\[
\begin{align*}
\frac{\partial^2 f(x,y)}{\partial x^2} &= -2y^2 \frac{1}{(x+y)^3} \\
\frac{\partial^2 f(x,y)}{\partial y^2} &= -2x^2 \frac{1}{(x+y)^3} \\
\frac{\partial^2 f(x,y)}{\partial x \partial y} &= \frac{2g(x+y) - 2y(x+y)^2}{(x+y)^4} = \frac{2xy}{(x+y)^3} - \frac{2y^2(x+y)}{(x+y)^4}.
\end{align*}
\]  
(16)

Let \(\nu_1(x,y)\) and \(\nu_2(x,y)\) be the two singular values of the Hessian matrix \(H(x,y)\) of \(f\). We have \(\nu_1 \nu_2 = \det(H(x,y)) = 0 = 4x^2y^2/(x+y)^6 - 4x^2y^2/(x+y)^6 = 0\), hence either \(\nu_1\) or \(\nu_2\) is equal to 0. We also have that \(\nu_1 + \nu_2 = \text{Tr}(H(x,y)) = -2(x^2 + y^2)/(x+y)^3 \leq 0\) on \((0,1]^2\), which is either the value of \(\nu_1(x,y)\) or \(\nu_2(x,y)\) (the other being 0). All singular values are negative on \((0,1]^2\), hence \(f\) is concave.

The denominator of \(\varphi_k\) is a sum of concave and strictly concave functions hence is strictly concave on \(D'_k\). It is also strictly positive on \(D'_k\).

The function \(\varphi_k\) is a ratio of a linear function and strictly concave and strictly positive function on \(D'_k\), it is therefore strictly quasi-convex on \(D'_k\). Indeed, if \(f : E \to \mathbb{R}\) is linear and \(g : E \to \mathbb{R}\) strictly concave and strictly positive, for \(\alpha \in (0,1)\) and for \((x,y) \in E^2\) we have:

\[
\frac{f(\alpha x + (1-\alpha)y)}{g(\alpha x + (1-\alpha)y)} \leq \frac{\alpha f(x) + (1-\alpha)f(y)}{\alpha g(x) + (1-\alpha)g(y)} = \frac{\alpha g(x) + (1-\alpha)g(y)}{\alpha g(x) + (1-\alpha)g(y)} \cdot \frac{f(x)}{g(x)} + \frac{(1-\alpha)g(x)}{\alpha g(x) + (1-\alpha)g(y)} \cdot \frac{f(y)}{g(y)} \leq \max\{f(x)/g(x), f(y)/g(y)\}.
\]

\(\square\)

### B.3.2 Derivation of the upper bound on the minimum asymptotic CR for large \(k\) in Equation (11)

Finding a closed form for the minimum of the asymptotic CR is difficult, we propose another point \(\lambda_0\) which is reasonably close to the results of the numerical optimization:

\[
\hat{\lambda}_k = \frac{1}{(K-k+1)^2}.
\]  
(17)

This particular point can also be used as an initialization for the numerical optimization. Here is a comparison with the result of a numerical optimization:
We express the competitive ratio using \( \tilde{\lambda} \):

\[
CR_{FTP}(\tilde{\lambda}) = \frac{\sum_{k=1}^{K} (\frac{1}{2} + K - k) \tilde{\lambda}_k}{\sum_{k=1}^{K} \left( \frac{1}{2} \lambda_k + \sum_{\ell=k+1}^{K} \frac{1}{\lambda_k + \lambda_\ell} \right)}
\]

\[
= \frac{\sum_{k=1}^{K} (\frac{1}{2} + K - k) \frac{1}{(K-k+1)^2}}{\sum_{k=1}^{K} \left( \frac{1}{2} \frac{1}{(K-k+1)^2} + \sum_{\ell=k+1}^{K} \frac{1}{(K-k+1)^2 + (K-\ell+1)^2} \right)}
\]

\[
= \frac{\sum_{k=1}^{K} \left( \frac{1}{2} + 1 \frac{1}{k^2} + \sum_{\ell=1}^{k-1} \frac{1}{\ell^2 + \ell^2} \right)}{\frac{1}{2} B_K + A_K}
\]

with \( H_K = \sum_{k=1}^{K} \frac{1}{k} \), \( B_K = \sum_{k=1}^{K} \frac{1}{k^2} \), and \( A_K = \sum_{k=1}^{K} \sum_{\ell=1}^{k-1} \frac{1}{k^2 + \ell^2} \).

We know that for the harmonic number \( H_K = \Theta(\log(K)) \), and that for the partial sum of the Basel problem \( 0 \leq B_K \leq \sum_{k=1}^{\infty} k^{-2} = \pi^2/6 = O(1) \).

Let us bound \( A_k \). Using the fact that for \( y > 0 \) and \( x > 0 \) the function
$f : (x, y) \mapsto (x^2 + y^2)^{-1}$ is decreasing in $x$, for $(k, \ell) \in [K]^2$ we have
\[
\int_{\ell}^{\ell+1} \frac{1}{k^2 + t^2} dt \leq \frac{1}{k^2 + \ell^2} \leq \int_{\ell-1}^{\ell+1} \frac{1}{k^2 + t^2} dt.
\]
\[
\frac{1}{k^2} \int_{\ell}^{\ell+1} \frac{1}{(t/k)^2 + 1} dt \leq \frac{1}{k^2 + \ell^2} \leq \frac{1}{k^2} \int_{\ell-1}^{\ell+1} \frac{1}{(t/k)^2 + 1} dt.
\]
\[
\frac{1}{k} (\arctan(\frac{\ell + 1}{k}) - \arctan(\frac{\ell}{k})) \leq \frac{1}{k^2 + \ell^2} \leq \frac{1}{k} (\arctan(\frac{\ell}{k}) - \arctan(\frac{\ell - 1}{k})).
\]
Hence by summing for $1 \leq \ell < k \leq K$:
\[
\sum_{k=1}^{K} \frac{1}{k} (\arctan(1) - \arctan(\frac{1}{k})) \leq A_K \leq \sum_{k=1}^{K} \frac{1}{k} (\arctan(\frac{k - 1}{k}) - \arctan(0)),
\]
\[
\sum_{k=1}^{K} \frac{1}{k} \left( \frac{\pi}{4} - \arctan(\frac{1}{k}) \right) \leq A_K \leq \sum_{k=1}^{K} \frac{1}{k} \arctan(\frac{k - 1}{k}).
\]
For the right hand-side we use that \(\arctan\) is increasing, thus
\[
A_K \leq \sum_{k=1}^{K} \frac{1}{k} \arctan(\frac{k - 1}{k}) \leq \frac{\pi}{4} H_K.
\]
Using that \(\arctan(x) \leq x\) for $x \geq 0$, we have
\[
A_K \geq \sum_{k=1}^{K} \frac{1}{k} \left( \frac{\pi}{4} - \frac{1}{k} \right) = \frac{\pi}{4} H_K - B_K.
\]
Combining everything we obtain the following inequality:
\[
\frac{H_K - \frac{1}{2} B_K}{\frac{\pi}{4} H_K + \frac{1}{4} B_K} \leq \text{CR}_{\text{FTPP}}(\tilde{\lambda}) \leq \frac{H_K - \frac{1}{2} B_K}{\frac{\pi}{4} H_K - \frac{1}{4} B_K}.
\]
Therefore
\[
\lim_{K \to \infty} \text{CR}_{\text{FTPP}}(\tilde{\lambda}) = \frac{4}{\pi} \approx 1.273.
\]

**B.3.3 Derivation of the expected CR under uniform distribution of type sizes in Equation (2)**

We will use equation (10) and rewrite the cost of the algorithm as:
\[
C_{\text{FTPP}} = \frac{n(n+1)}{2} \sum_{i=1}^{k} \lambda_i + \sum_{i=1}^{k} \sum_{j=i+1}^{k} n^2 \min(\lambda_i, \lambda_j).
\]
We can now take the expectation in $\lambda$ to compute the expected cost of FTPP and by using Lemma 5 we have

$$E_\lambda[C_{\text{FTPP}}] = \frac{n(n + 1) k}{2} + \sum_{i=1}^{k} \sum_{j=i+1}^{k} n^2 E_\lambda[\min(\lambda_i, \lambda_j)]$$

and for OPT:

$$E_\lambda[C_{\text{OPT}}] = \frac{n(n + 3) k}{2} + n^2 \frac{k(k - 1)}{2} \frac{1}{3} (1 - \log(2)).$$

Hence

$$CR_{\text{FTPP}}(k, n) = \frac{(n+1)^4}{4} + \frac{(k-1)n^6}{6} - \frac{6(n + 1) + 4(k - 1)n}{2(1 - \log(2))} \rightarrow k \rightarrow \infty \approx 1.629.$$ 

Note that for two types of jobs:

$$\lim_{n \rightarrow \infty} CR_{\text{FTPP}}(2, n) = \frac{10}{11 - 8 \log(2)} \approx 1.833.$$ 

Let us look at the growth of $CR_{\text{FTPP}}$ in $k$ or $n$. We have

$$\partial_n CR_{\text{FTPP}}(k, n) = \frac{9(6 + 4(k - 1)) - 6(3 + 8(k - 1)(1 - \log(2)))}{(3(n + 3) + 8(k - 1)n(1 - \log(2)))^2}$$

$$\geq \frac{k(36 - 48(1 - \log(2))}{(3(n + 3) + 8(k - 1)n(1 - \log(2)))^2} \geq 0.$$ 

It is increasing in $n$. And for $k$ we have:

$$\partial_k CR_{\text{FTPP}}(k, n) = \frac{4n(3(n + 3)) - 6(n + 1)8n(1 - \log(2))}{(3(n + 3) + 8(k - 1)n(1 - \log(2)))^2}$$

$$= 4n \frac{(3(n + 3)) - 12(n + 1)(1 - \log(2))}{(3(n + 3) + 8(k - 1)n(1 - \log(2)))^2}$$

$$= 4n \frac{(12 \log(2) - 9)n + 12 \log(2) - 3}{(3(n + 3) + 8(k - 1)n(1 - \log(2)))^2},$$

which is negative for $n \geq \lceil (12 \log(2) - 3)/(9 - 12 \log(2)) \rceil = 8$ and positive for $n \leq \lfloor (12 \log(2) - 3)/(9 - 12 \log(2)) \rfloor = 7$. Hence the function is decreasing in $k$ when $n \geq 8$, and increasing otherwise.
Lemma 5.

\[ \int_0^1 \int_0^1 \min(x, y) \, dx \, dy = \frac{1}{3} \quad \text{and} \quad \int_0^1 \int_0^1 \frac{xy}{x+y} \, dx \, dy = \frac{2}{3} (1 - \log(2)) \quad (18) \]

Proof. For the first double integral:

\[ \int_0^1 \int_0^1 \min(x, y) \, dx \, dy = \int_0^1 \left( \int_0^x y \, dy + \int_x^1 x \, dx \right) \]

\[ = \int_0^1 \frac{x^2}{2} + x(1-x) \, dx \]

\[ = \left( \frac{1}{6} + \frac{1}{2} - \frac{1}{3} \right) = \frac{1}{3} \]

For the second double integral:

\[ \int_0^1 \int_0^1 \frac{xy}{x+y} \, dx \, dy = \int_0^1 x \int_0^1 \left( 1 - \frac{x}{x+y} \right) \, dx \, dy \]

\[ = \int_0^1 x - x^2 \left( \log(x+1) - \log(x) \right) \, dx. \]

There are three terms under the integral. The first one is equal to 1/2, and the two other will be computed by change of variables and integrating by parts:

\[ \int_0^1 x^2 \log(x) \, dx = \left[ \frac{x^3}{3} \log(x) \right]_0^1 - \int_0^1 \frac{x^2}{3} \, dx = -\frac{1}{9} \]

We have:

\[ \int_0^1 x^2 \log(x+1) \, dx = \int_1^2 (1-x')^2 \log(x') \, dx' = \int_1^2 \log(x) + x^2 \log(x) - 2x \log(x) \, dx. \]

We need to compute each of the three terms under the integral:

\[ \int_1^2 \log(x) \, dx = \left[ x \log(x) - x \right]_1^2 = 2 \log(2) - 1 \]

\[ \int_1^2 x \log(x) \, dx = \left[ \frac{x^2}{2} \log(x) \right]_1^2 - \int_1^2 \frac{x}{2} \, dx = 2 \log(2) - \frac{3}{4} \]

\[ \int_1^2 x^2 \log(x) \, dx = \left[ \frac{x^3}{3} \log(x) \right]_1^2 - \int_1^2 \frac{x^2}{3} \, dx = \frac{8}{3} \log(2) - \frac{7}{5} \]

Finally:

\[ \int_0^1 \int_0^1 \frac{xy}{x+y} \, dx \, dy = \frac{1}{2} - \frac{1}{9} - (2 \log(2) - 1 + \frac{8}{3} \log(2) - \frac{7}{9} + \frac{3}{2} - 4 \log(2)) \]

\[ = \frac{2}{3} (1 - \log(2)) \]

□
C Learning algorithms

C.1 Upper bound for ETC-U: Proof of Proposition 4

We start with the following technical lemma, isolated to be reused in other proofs. Pick some \( \alpha \in \mathbb{N} \). Let \((X^1_i)_{i \in [\alpha n]}\) and \((X^2_i)_{i \in [\alpha n]}\) be independent exponential variables of parameters \( \lambda_1 \) and \( \lambda_2 \) respectively. Define for any \( m \in [\alpha n] \):

\[
\hat{r}^m = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{X^1_i < X^2_i}
\]

and

\[
\delta^{(m, n)} = \sqrt{\frac{\log(2n^2)}{2m}}.
\]

Let \( r \) denote the expectation \( r := \mathbb{E} \left[ \mathbb{1}_{X^1_i < X^2_i} \right] \).

**Lemma 6.** For any \( m \in [\alpha n] \), the estimator \( \hat{r}^m \) is within \( \delta^{(m, n)} \) of its expectation w.h.p:

\[
P \left( \exists m \in [\alpha n] \text{ s.t. } |\hat{r}^m - r| \geq \delta^{(m, n)} \right) \leq \frac{\alpha}{n}.
\]

**Proof.** By Hoeffding’s inequality:

\[
\forall m \in [\alpha n], \quad P \left( |\hat{r}^m - r| \geq \sqrt{\frac{\log(2n^2)}{2m}} \right) \leq \frac{1}{n^2}
\]

The lemma is then obtained by a union bound over the \( \alpha n \) possible values of \( m \). \( \square \)

We are now ready to prove Proposition 4.

**Proof.** Recall that we assumed without loss of generality that \( \lambda_1 \leq \cdots \leq \lambda_K \). Recall also the definition for any \((k, \ell) \in [K]^2\), for any \((m_\ell, m_k) \in [n]^2\), of:

\[
\hat{r}^{\min(m_k, m_\ell)}_{k, \ell} = \frac{1}{\min(m_k, m_\ell)} \sum_{i=1}^{\min(m_k, m_\ell)} \mathbb{1}_{p^i_k < p^i_\ell}.
\]

Let us define the good event \( \mathcal{E} \) as:

\[
\mathcal{E} := \left\{ \forall (k, \ell) \in [K]^2, \forall m \in [n], |\hat{r}^m_{k, \ell} - \mathbb{E}[r^m_{k, \ell}]| < \delta^{(m, n)} \right\}
\]

By Lemma 6 applied with \( \alpha = 1 \), for any couple \((\ell, p)\) it holds that :

\[
P \left( \exists m \in [n] \text{ s.t. } |\hat{r}^m_{k, \ell} - \mathbb{E}[r^m_{k, \ell}]| > \delta^{(m, n)} \right) \leq \frac{1}{n}.
\]

30
A union bound over the \( \frac{K(K-1)}{2} \) possible pairs gives the following bound:

\[
\Pr(\mathcal{E}) \leq \frac{K^2}{2n}.
\] (19)

We decompose the performance of the algorithm based on the occurrence of event \( \mathcal{E} \).

\[
\mathbb{E}[C_{ETC-U}] \leq \mathbb{E}[C_{ETC-U}|\mathcal{E}] + \mathbb{E}[C_{ETC-U}|\overline{\mathcal{E}}] \Pr(\mathcal{E}).
\] (20)

We start by bounding the second term.

ETC-U is a non-preemptive algorithm, and the algorithm’s decision to start a job only depends on the jobs type. We thus have:

\[
\mathbb{E}[C_{ETC-U}|\mathcal{E}] = \sum_{s=1}^{N} \sum_{k=1}^{K} \sum_{i=1}^{n} (N - s + 1) \mathbb{E}\left[ P_k^k \mathbbm{1}\{ \text{job } P_k^k \text{ computed in position } s \} \mid \mathcal{E} \right]
\]

\[
= \sum_{s=1}^{N} \sum_{k=1}^{K} \sum_{i=1}^{n} (N - s + 1) \lambda_k \mathbb{E}\left[ \mathbbm{1}\{ \text{job } P_k^k \text{ computed in position } s \} \mid \mathcal{E} \right]
\]

\[
\leq N \sum_{k=1}^{K} \sum_{i=1}^{n} \lambda_k \mathbb{E}\left[ \sum_{s=1}^{N} \mathbbm{1}\{ \text{job } P_k^k \text{ computed in position } s \} \mid \mathcal{E} \right]
\]

\[
\leq N \sum_{k=1}^{K} \lambda_k
\]

\[
= Kn^2 \sum_{k=1}^{K} \lambda_k
\]

\[
\leq 2K \mathbb{E}[C_{OPT}].
\] (21)

Combined with equation (19), this gives:

\[
\mathbb{E}[C_{ETC-U}|\mathcal{E}] \Pr(\mathcal{E}) \leq \frac{K^3}{n} \mathbb{E}[C_{OPT}].
\] (22)

It remains to bound \( \mathbb{E}[C_{ETC-U}|\overline{\mathcal{E}}] \).

Consider any couple \((k, \ell) \in [K]^2 \) s.t. \( \ell \leq k \). Let us call \( m_{\ell,k}^* \) the number of comparisons performed between jobs of type \( \ell \) and \( k \) before the algorithm detects \( \lambda_\ell \leq \lambda_k \). A first obvious upper bound is \( m_{\ell,k}^* \leq n \). A second upper is obtained by noting that \( m_{\ell,k}^* \) is smaller than any \( m' \) s.t.

\[
\delta(m',n) < \frac{1}{2} \left| \frac{\lambda_k}{\lambda_k + \lambda_\ell} - 0.5 \right|.
\]

This implies the following upper bound on \( m_{\ell,k}^* \):

\[
m_{\ell,k}^* \leq \min \left( n, 8 \left( \frac{\lambda_k + \lambda_\ell}{\lambda_k - \lambda_\ell} \right)^2 \log (2n^2) \right).
\] (23)
We decompose the run of the algorithm. For each \( \ell \in [K] \), we call phase \( \ell \) the iterations at which jobs of type \( \ell \) are the jobs with smallest mean still not terminated. Note that during phase \( \ell \), job type \( \ell \) is always active, as the contrary would mean event \( E \) does not hold. This implies that the number of job of any type \( k > \ell \) computed during phase \( \ell \) is lower than \( m_{\ell,k}^* \). We have the following bound (in the following computation the conditioning on event \( E \) is omitted for readability):

\[
E[C_{\text{ETC-U}} | E] = E[C_{\text{FTPP}}] + \sum_{\ell=1}^{K} \sum_{k=\ell+1}^{K} \sum_{(i,j) \in [n]^2} (\lambda_k - \lambda_\ell)E[1\{P_k^i \text{computed before } P_j^j\}]
\]

\[
\leq E[C_{\text{FTPP}}] + \sum_{\ell=1}^{K} \sum_{k=\ell+1}^{K} \sum_{(i,j) \in [n]^2} (\lambda_k - \lambda_\ell)E[1\{P_k^i \text{computed before phase } \ell + 1\}]
\]

\[
\leq E[C_{\text{FTPP}}] + \sum_{\ell=1}^{K} \sum_{k=\ell+1}^{K} \sum_{o=1}^{\ell} n(\lambda_k - \lambda_\ell)E\left[ \sum_{o=1}^{\ell} 1\{P_k^i \text{computed during phase } o\} \right]
\]

\[
\leq E[C_{\text{FTPP}}] + \sum_{\ell=1}^{K} \sum_{k=\ell+1}^{K} \sum_{o=1}^{\ell} E[m_{\ell,k}^*]n(\lambda_k - \lambda_\ell)
\]

\[
\leq E[C_{\text{FTPP}}] + \sum_{\ell=1}^{K} \sum_{k=\ell+1}^{K} \sum_{o=1}^{\ell} E[m_{o,k}^*]n(\lambda_k - \lambda_o)
\]

\[
= E[C_{\text{FTPP}}] + \sum_{k=1}^{K} \sum_{o=1}^{k-1} E[m_{o,k}^*]n(k-o)(\lambda_k - \lambda_o)
\]

\[
\leq E[C_{\text{FTPP}}] + \sum_{k=1}^{K} \sum_{\ell=1}^{k-1} E[m_{\ell,k}^*]n(K-\ell)(\lambda_k - \lambda_\ell)
\]

\[
\leq E[C_{\text{FTPP}}] + \sum_{k=1}^{K} \sum_{\ell=1}^{k-1} \min\left(n, 8\left(\frac{\lambda_k + \lambda_\ell}{\lambda_k - \lambda_\ell}\right)^2 \log (2n^2)\right) n(K-\ell)(\lambda_k - \lambda_\ell).
\]

For any \( \lambda_k \geq \lambda_\ell \), we have:

\[
(\lambda_k - \lambda_\ell) \min\left(n, 8\left(\frac{\lambda_k + \lambda_\ell}{\lambda_k - \lambda_\ell}\right)^2 \log (2n^2)\right) \leq (\lambda_k + \lambda_\ell)\sqrt{8n \log(2n^2)}.
\]
This implies:

\[
E[C_{ETC-U}\mid \mathcal{E}] \leq E[C_{FTP}] + \sum_{k=1}^{K} \sum_{\ell=1}^{K-1} n(K - \ell)(\lambda_k + \lambda_\ell)\sqrt{8n\log(2n^2)}
\]

\[
= E[C_{FTP}] + \sum_{k=1}^{K} \left[ \frac{1}{2}(k - 1)(2K - k) + (K - k)^2 \right] \lambda_k n \sqrt{8n\log(2n^2)}.
\]

Combining this with equation (22) gives the result of the proposition. \(\square\)

### C.2 Upper bound for ETC-RR: Proof of Prop 5

**Proof.** For any couple \((k, \ell)\) if at some iteration \(h_{k,\ell} + h_{\ell,k} = m\), we define the more precise notation for \(\hat{r}_{\ell,k}^m\) at that iteration as \(\hat{r}_{\ell,k}^m\). Note that there are at most \(2n\) possible values for \(m\) as there are \(2n\) jobs of type \(\ell\) or \(k\). Let us start by showing that the estimators \(\hat{r}_{\ell,k}\) concentrate around their expectation w.h.p.

Exponential random variables are memory-less, i.e. if \(X_i \sim \text{Exp}(\lambda_i)\), the law of \(X_i\) conditionally on it being larger than a constant is unchanged. For any couple \((k, \ell)\) \(\in [K]^2\), any \(m \in [2n]\), we thus have:

\[
\hat{r}_{\ell,k}^m \leq \frac{1}{m} \sum_{i=1}^{m} 1\{X_i^\ell < X_i^k\},
\]

with \((X_i^\ell)_{i \in [m]}\) and \((X_i^k)_{i \in [m]}\) independent exponential variables of parameters \(\lambda_\ell\) and \(\lambda_k\) respectively.

Let \(\mathcal{E}\) be the event

\[
\mathcal{E} = \left\{ \forall(k, \ell) \in [K]^2, \forall m \in [2n], \left| \hat{r}_{\ell,k}^m - \frac{\lambda_k}{\lambda_\ell + \lambda_k} \right| < \delta_{m,n} \right\}.
\]

recall that for any \(m \in [2n]\), \(E[\hat{r}_{\ell,k}^m] = \frac{\lambda_k}{\lambda_\ell + \lambda_k}\). By Lemma \(6\) applied with \(\alpha = 2\), and a union bound over the \(K(K - 1)/2\) possible pairs, we have:

\[
P(\mathcal{E}) \leq \frac{K(K - 1)}{n}.
\]

We study separately the cost of the algorithm depending on the realisation of event \(\mathcal{E}\).

\[
E[C_{ETC-RR}] \leq E[C_{ETC-RR}\mid \mathcal{E}] + P(\mathcal{E})E[C_{ETC-RR}\mid \mathcal{E}] + P(\mathcal{E})E[C_{ETC-RR}\mid \mathcal{E}^c].
\]

As the algorithm’s decision to start jobs is agnostic to the job sizes realisations,
we have:

\[ E[C_{ETC-RR}|E] = \sum_{i \in [n]} \sum_{j \in [n]} \sum_{\ell \in [K]} \sum_{p \in [K]} E(\text{Computing time of job } P^\ell_i \text{ before } P^p_j \text{ is finished}|E) \]

\[ \leq \sum_{i \in [n]} \sum_{j \in [n]} \sum_{\ell \in [K]} \sum_{p \in [K]} E(P^p_j|E[\{P^p_j \text{ started before } P^\ell_i \text{ finished }\} |E]) \]

\[ = \sum_{j \in [n]} \sum_{p \in [K]} \lambda_p E\left[ \sum_{i \in [n]} \sum_{\ell \in [K]} \{P^p_j \text{ started before } P^\ell_i \text{ finished }\}|E]\right] \]

\[ \leq Nn \sum_{k \in [K]} \lambda_k \]

\[ = Kn^2 \sum_{k \in [K]} \lambda_k \]

\[ \leq 2K E[C_{OPT}]. \]

This together with equation (25) gives:

\[ P(\mathcal{E}) E[C_{ETC-RR}|E] \leq \frac{2K^3}{n} E[C_{OPT}]. \]

(27)

It remains to bound \( E[C_{ETC-RR}|\mathcal{E}] \).

For any \( k > \ell \), note \( m^\text{max}_{\ell,k} \) the maximum value attained by \( m \) before job type \( k \) is detected to have a larger mean than the job type \( \ell \). Under event \( \mathcal{E} \), \( m^\text{max}_{\ell,k} \) is smaller than any \( m' \) s.t.:

\[ \delta^{(m',n)} < \frac{1}{2} \left| \frac{\lambda_k}{\lambda_k + \lambda_k} - 0.5 \right|. \]

Under event \( \mathcal{E} \) the following bound holds:

\[ m^\text{max}_{\ell,k} \leq \min \left( 2n, 2 \left\lfloor \frac{\log(2n^2)}{0.5 - \frac{\lambda_k}{\lambda_k + \lambda_k}} \right\rfloor \right) \]

\[ = \min \left( 2n, 8 \left( \frac{\lambda_k + \lambda_k}{\lambda_k - \lambda_k} \right)^2 \log(2n^2) \right) = m^*_\ell,k. \]

We are now ready to bound \( E[C_{ETC-RR}|\mathcal{E}] \). Note \( b^\ell_i \) and \( e^\ell_i \) the beginning and end dates of the computation of the \( i^{th} \) job of type \( \ell \), and \( r_\ell(t) \) the number
of jobs of type $\ell$ not done computing at time $t$. We have the following:

$$A := \mathbb{E}[C_{\text{ETC-RR}}|\mathcal{E}] - \mathbb{E}[C_{\text{FTPP}}]$$

$$\leq \mathbb{E} \left[ \sum_{i \in [n]} \sum_{j \in [n]} \sum_{k \in [K]} \sum_{\ell \in [1,k-1]} \lambda_k \mathbb{1}\{b^k_j < e^\ell_i\} - \lambda_\ell \mathbb{1}\{e^\ell_j < b^k_i\} \right]$$

Now note $r_\ell(t)$ the number of jobs of type $\ell$ not done computing at time $t$. If a job of type $k > \ell$ starts computing at time $t$ where $r_\ell(t) > 0$, the following two facts hold:

1. this job starts computing before $r_\ell(t)$ jobs of type $\ell$ are done,

2. the previous job of type $k$ that was computing finished before at least $r_\ell(t) - 1$ jobs of type $\ell$ started.

Thus, for any $(k, j) \in [K, n]$, if job $P^k_j$ is started at time $t$, we have:

$$\sum_{i \in [n]} \lambda_k \mathbb{1}\{b^k_j < e^\ell_i\} - \lambda_\ell \mathbb{1}\{e^\ell_j < b^k_i\} \leq \sum_{\ell=1}^{k-1} [\lambda_k r_\ell(t) - \lambda_\ell (r_\ell(t) - 1)] \mathbb{1}\{r_\ell(t) > 0\}$$

$$\leq \sum_{\ell=1}^{k-1} [n(\lambda_k - \lambda_\ell) + \lambda_\ell] \mathbb{1}\{r_\ell(t) > 0\}.$$ 

As we did for the analysis of ETC-U, the run of the algorithm is split into phases. Let $T$ be the ensemble of times at which new jobs were started. For each $\ell \in [K]$, we call phase $\ell$ the times at which jobs of type $\ell$ are the jobs with smallest mean still not terminated. Note that during phase $\ell$, job type $\ell$ is always active, as the contrary would mean event $\mathcal{E}$ does not hold. Note $n_k(\ell)$ the number of jobs of any type $k > \ell$ started during phase $\ell$. The following bound holds:

$$\mathbb{E}[n_k(\ell)|\mathcal{E}] \leq \frac{\lambda_\ell}{\lambda_\ell + \lambda_k} m_{\ell,k}^*.$$ 

We thus have (conditioning on event $\mathcal{E}$ is omitted in the following computa-
E\[C_{ETC-RR}|E\] − E\[C_{FTPP}\]

\[
\leq \mathbb{E} \left[ \sum_{j \in [n]} \sum_{\ell \in [T]} \sum_{k \in [K]} [n(\lambda_k - \lambda_{\ell}) + \lambda_{\ell}] \mathbb{I}\{r_{\ell}(t) > 0\} \mathbb{I}\{P_j^k\text{started at } t\} \right]
\]

\[
= \mathbb{E} \left[ \sum_{j \in [n]} \sum_{\ell \in [T]} \sum_{k \in [K]} [n(\lambda_k - \lambda_{\ell}) + \lambda_{\ell}] \mathbb{I}\{P_j^k\text{started before phase } \ell + 1\} \right]
\]

\[
\leq \sum_{k \in [K]} \sum_{\ell = 1}^{k-1} \left[ (n(\lambda_k - \lambda_{\ell}) + \lambda_{\ell}) \sum_{o=1}^{\ell} \frac{\lambda_o}{\lambda_o + \lambda_k} m_{o,k}^* \right]
\]

\[
\leq \sum_{k \in [K]} \sum_{\ell = 1}^{k-1} \sum_{o=1}^{\ell} n(\lambda_k - \lambda_o) \frac{\lambda_o}{\lambda_o + \lambda_k} m_{o,k}^* + \sum_{k \in [K]} \sum_{\ell = 1}^{k-1} \ell \lambda_{\ell} \frac{\lambda_{\ell}}{\lambda_{\ell} + \lambda_k} m_{\ell,k}^*
\]

\[
\leq \sum_{k \in [K]} \sum_{\ell = 1}^{k-1} \left[ (K - \ell)n(\lambda_k - \lambda_{\ell}) \frac{\lambda_{\ell}}{\lambda_{\ell} + \lambda_k} m_{\ell,k}^* + \ell \lambda_{\ell} \frac{\lambda_{\ell}}{\lambda_{\ell} + \lambda_k} m_{\ell,k}^* \right].
\]

For any \(\lambda_k \geq \lambda_{\ell}\), we have:

\[
(K - \ell)n(\lambda_k - \lambda_{\ell}) \frac{\lambda_{\ell}}{\lambda_{\ell} + \lambda_k} m_{\ell,k}^* = \frac{\lambda_{\ell}(\lambda_p - \lambda_{\ell})}{\lambda_p + \lambda_{\ell}} \min \left(2n, 8 \left(\frac{\lambda_p + \lambda_{\ell}}{\lambda_p - \lambda_{\ell}}\right)^2 \log(2n^2)\right)
\]

\[
\leq 2(K - \ell)n\lambda_{\ell}\sqrt{4n\log(2n^2)}.
\]

On the other hand, for any \(\ell < K\):

\[
\ell \lambda_{\ell} \frac{\lambda_{\ell}}{\lambda_{\ell} + \lambda_k} m_{\ell,k}^* \leq 2n(K - 1)\lambda_{\ell}.
\]

This implies:

\[
\mathbb{E}\[C_{ETC-RR}|E\] - \mathbb{E}\[C_{FTPP}\] \leq \sum_{k \in [K]} \sum_{\ell = 1}^{k-1} \left[ 2(K - \ell)n\lambda_{\ell}\sqrt{4n\log(2n^2)} + 2n(K - 1)\lambda_{\ell} \right]
\]

\[
\leq \sum_{\ell = 1}^{K-1} 2n(K - \ell)^2\lambda_{\ell}\sqrt{4n\log(2n^2)} + 2n(K - 1)^2n \sum_{\ell = 1}^{K-1} \lambda_{\ell}.
\]

Combining this with Equation \(27\) gives the proposition. \(\square\)