ORDINAL PREDICTIONS SUFFICE FOR NON-CLAIRVOYANT SCHEDULING: CLOSING THE GAP IN THE CONSISTENCY–ROBUSTNESS TRADE-OFF

Anonymous authors
Paper under double-blind review

ABSTRACT

We study the scheduling of a set of jobs on a single machine, with the objective of minimizing the *total completion time*: the sum of the times at which the individual jobs finish. The problem has three variants, differing in what the algorithm is told about the running times. In the classical variant they are given in advance, and it is well known that the least possible total completion time, the *optimum*, is reached by the shortest-processing-time rule, which runs the jobs in increasing order of running time. In the *non-clairvoyant* variant the running times are hidden, and an algorithm learns one only when the job in question finishes; it must then allocate the machine before it can tell short jobs from long ones, which puts the optimum beyond reach. Round Robin, which shares the machine equally among the unfinished jobs, stays within a factor two of the optimum, and no deterministic algorithm does better. The third variant, *learning-augmented* scheduling, is the subject of this paper: the algorithm is given, in addition, a prediction of the running times. It is judged by two quantities. Its *consistency* is the largest ratio of its total completion time to the optimum when the prediction is correct, and its *robustness* is the largest such ratio over all instances and predictions. How favorably the two can be traded against each other has been open, with a quadratic gap between what is achievable and what is forbidden. At consistency $1+\varepsilon$ the best robustness known was $\Theta(1/\varepsilon)$, attained by *time sharing*, which divides the machine between Round Robin and an algorithm that follows the prediction. On the other side, the lower bound of Wei and Zhang forbids robustness below order $1/\sqrt{\varepsilon}$ for every deterministic algorithm, and does so even when the prediction supplies the running times as numbers. We close the gap by introducing a new algorithm, *Prediction-Drift Round Robin* (PDRR). It is deterministic, and it consults only the order of the predicted running times, never their values, so the prediction it requires is purely *ordinal*. A single parameter ε controls how far PDRR follows that order. It holds the unfinished jobs at a common level of accumulated service, as Round Robin does, with one exception: the job predicted to be shortest is never more than a factor $1/\varepsilon$ above that level. PDRR is $(1+\varepsilon)$-consistent and $\min\{2/\sqrt{\varepsilon},\ 2+1/\sqrt{\varepsilon}\}$-robust, matching the lower bound up to a constant factor. The optimal order of the trade-off is thus reached from ordinal information alone.

1 INTRODUCTION

We consider the classical *scheduling* problem, where we run a set of jobs on a single machine with the objective of achieving *optimality*: minimizing the *total completion time*, that is, the sum of the times at which the individual jobs finish. When the running times are known in advance, it is well known that optimality is achieved by running the shortest jobs first (Smith, 1956). The problem has three variants, differing in what the algorithm is told about the running times: the classical one just described, where they are given in advance; the non-clairvoyant one, where they are hidden; and the learning-augmented one, where they are hidden but a prediction is supplied.

In the *non-clairvoyant* variant (Motwani et al., 1994), the running times are hidden: an algorithm observes only which jobs have completed, so it learns how long a job takes at the moment that job finishes and not before. It must therefore commit machine time before it can learn which jobs are

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

shortest, which puts optimality beyond reach. We then judge an algorithm by the ratio between its total completion time and the optimum defined above. The best achievable ratio is two, attained by the *Round Robin* algorithm, which shares the machine equally among the unfinished jobs.

In the *learning-augmented* variant, the algorithm is given a prediction of the running times. The prediction comes with no guarantee: it may be correct or wrong, and the algorithm is judged in both cases (Lykouris & Vassilvitskii, 2018; Kumar et al., 2018). We say that an algorithm is *deterministic* if the schedule it produces is determined by the instance and the prediction alone; every algorithm considered in this paper is deterministic. Its *consistency* is the largest ratio it can incur when the prediction is correct; its *robustness* is the largest ratio it can incur over all instances and predictions. We would like consistency close to one and robustness close to two. Consistency one is conceivable: an algorithm that follows the prediction runs the shortest jobs first on every instance where the prediction happens to be correct, and is optimal there, even though it never learns whether the prediction is correct. Robustness two means matching the worst-case guarantee of Round Robin. The two pull against each other: the more an algorithm commits to the prediction, the more it loses when the prediction is wrong, and the less it commits, the less it gains when the prediction is correct.

A quadratic gap. How favorably can consistency and robustness be traded (Elenter et al., 2024)? Suppose we insist on consistency $1+\varepsilon$, so that a correct prediction leaves us within a factor $1+\varepsilon$ of the optimum, and ask how small the robustness can then be made. The standard way to achieve consistency $1+\varepsilon$ while keeping robustness bounded is *time sharing* (Kumar et al., 2018; Lindermayr & Megow, 2022), an algorithm assembled from two others. A fixed fraction $\lambda \in [0,1]$ of the machine's unit speed is given to *Follow the Prediction*, which processes the jobs one at a time in the predicted order, and the remaining $1-\lambda$ to Round Robin; at every instant each of the two is run at its own share of the speed. With a correct prediction Follow the Prediction runs the jobs shortest first, so on the full machine it would be optimal. Inside time sharing it runs at speed λ, which multiplies all its completion times by at most $1/\lambda$; choosing $\lambda = 1/(1+\varepsilon)$ therefore gives consistency $1+\varepsilon$. Round Robin is then left with speed $\varepsilon/(1+\varepsilon)$, which multiplies its completion times by at most $(1+\varepsilon)/\varepsilon$ and turns its ratio of two into $2(1+\varepsilon)/\varepsilon$, of order $1/\varepsilon$. On the other side, the lower bound of Wei & Zhang (2020) implies that no deterministic algorithm with consistency $1+\varepsilon$ can have robustness $o(1/\sqrt{\varepsilon})$ (Appendix D). The two orders, the achievable $1/\varepsilon$ and the forbidden $1/\sqrt{\varepsilon}$, differ quadratically, and no algorithm was known to reach anything between them with static predictions. The gap has resisted recent effort. Benomar et al. (2025) design new algorithms for this trade-off, but on general instances their robustness is again of order $1/\varepsilon$.

We close the gap. We give a deterministic algorithm, *Prediction-Drift Round Robin* (PDRR). It keeps the unfinished jobs at a common level of accumulated service, as Round Robin does, with a single exception. The job that the prediction ranks first among the unfinished ones, the *preferred* job, may run ahead of that level by a factor $f = 1/\varepsilon$, and no further. Once it has reached that lead, it advances only fast enough to maintain it. When the preferred job completes, the next job in the predicted order takes its place and advances until it too is a factor f ahead or completes. The lead is a ratio, so the schedule carries no built-in unit of time, and at $\varepsilon = 1$ the lead disappears and PDRR becomes Round Robin. For every $\varepsilon \in (0,1]$, PDRR is $(1+\varepsilon)$-consistent (Theorem 7) and $\min\{2/\sqrt{\varepsilon},\ 2+1/\sqrt{\varepsilon}\}$-robust (Theorem 14). This matches the lower bound of Wei & Zhang (2020) up to a constant factor, so the quadratic gap closes on the lower bound's side.

Ordinal predictions suffice. The prediction that PDRR consults is a permutation: it states which job is expected to be shorter, with no estimate of how long any job is. Such an *ordinal* prediction (Lindermayr & Megow, 2022; Eliáš et al., 2024) is strictly weaker than the numerical processing-time estimates assumed in most of the learning-augmented scheduling literature, whether given for all jobs (Kumar et al., 2018; Wei & Zhang, 2020; Im et al., 2021; Benomar et al., 2025) or for a subset of them (Benomar & Perchet, 2024), since estimates induce a ranking whereas a ranking says nothing about magnitudes. Using only the order is an advantage in two respects. In practice, such a prediction can be easier to produce: a model can learn a ranking directly (Fu et al., 2024), or from pairwise preferences (Burges et al., 2005), without estimating running times. In theory, the guarantee is stronger: the lower bound of Wei & Zhang (2020) is proved in the richer numerical model, so it binds an algorithm that reads only the order as well, and PDRR meets it from that weaker input. The numerical values are therefore not needed to attain the optimal order.

Proof overview. The pairwise decomposition (Motwani et al., 1994) splits the total completion time into three parts: the processing times, the service that smaller jobs receive before larger ones finish,

and the service that larger jobs receive before smaller ones finish. The optimum alone pays for the first two, so everything rests on the third. There the multiplicative lead acts pairwise: if $p_i \le p_j$, job j receives at most $\min\{p_j,\ f \cdot p_i\}$ of service before i completes.

Summing that bound over all pairs is where the square root appears. Let $N(x)$ count the jobs of size at least x. The sum equals $\int_0^\infty N(x)\,N(x/f)\,dx - \mathrm{OPT}$, which pairs the profile of job sizes with a copy of itself rescaled by f. Cauchy–Schwarz bounds the integral by $\sqrt{f}\int_0^\infty N(x)^2\,dx$, taking the square root of the factor f that the rescaling contributes, and $\int_0^\infty N(x)^2\,dx \le 2\mathrm{OPT}$ then gives robustness $2\sqrt{f} = 2/\sqrt{\varepsilon}$. Lowering the leading coefficient from 2 to 1, which is the better bound once $\varepsilon < \frac{1}{4}$, needs one further step. From each larger-to-smaller delay we subtract the p_i that Round Robin would have delivered anyway, leaving the *preferred excess* that following the prediction creates, and charge that excess to disjoint layers of the optimum. The charges stay disjoint because a job that becomes preferred remains preferred until it completes.

2 Preliminaries

In this section we fix the model and the notation used throughout the paper: the scheduling problem and its optimum, the ordinal prediction and the two measures by which an algorithm is judged, and the decomposition of total completion time into pairwise interactions on which every later argument rests. To keep the presentation compact, all proofs are deferred to Appendix A.

The problem. An instance consists of n jobs $J = \{1, \dots, n\}$ with *processing requirements* $p_j > 0$, all available from time 0; we also speak of the running time or the size of a job, meaning the same quantity. The machine has unit speed, that is, it performs one unit of work per unit of time, and this capacity may be split among the jobs. The allocation may change at any moment, and a job given rate zero simply stops accruing service, keeping what it already has; in the standard terminology this is the preemptive setting, and nothing is charged for changing the allocation. The *accrued service* of a job is the amount of processing it has received so far. Job j completes once its accrued service reaches p_j, and we write C_j for the time at which this happens. The objective is to minimize the total completion time $\sum_j C_j$.

We number the jobs so that $p_1 \le \cdots \le p_n$, breaking ties arbitrarily. This is a choice of names for the jobs, not an assumption about the instance, since every instance can be numbered this way; its effect is that $i < j$ implies $p_i \le p_j$ throughout the paper, and that the numbering is itself the true order against which a prediction is compared. We write OPT for the least total completion time that any schedule achieves. It is a classical fact that this value is attained by the schedule that runs the jobs one at a time, shortest first. Job j then finishes at $p_1 + \cdots + p_j$, and summing these completion times counts each p_j once for every job that finishes no earlier than j, so $\mathrm{OPT} = \sum_{j=1}^{n} (n - j + 1)\,p_j$.

Algorithms. The instance fixes the jobs; an algorithm decides how the machine is used. At each instant it chooses a service rate for each unfinished job, and we require these rates to sum to one, since leaving part of the machine idle can only delay completions. The algorithms considered here are *non-clairvoyant*: the requirements p_j are not part of their input, and the only thing an algorithm learns about a job is the moment at which it completes. Round Robin, which splits the machine equally among the unfinished jobs, is one such algorithm. All of them are also *deterministic*, so the schedule is fixed once the input is fixed.

Predictions. In the *learning-augmented* setting an algorithm receives, besides the instance, a prediction about the running times, and may let its rates depend on it. Most of the literature takes the prediction to be numerical, an estimate $\hat{p}_j$ of each running time. This paper takes it to be *ordinal*: a bijection $\hat{\pi} : J \to \{1, \dots, n\}$ that assigns to each job j its *predicted rank* $\hat{\pi}(j)$, a smaller rank meaning that j is predicted to be shorter. The prediction is *correct* if it never gives a strictly larger job a smaller rank than a strictly smaller one, that is, if $\hat{\pi}(i) < \hat{\pi}(j)$ implies $p_i \le p_j$. Since equal-sized jobs are interchangeable, we may relabel jobs within ties so that a correct prediction assigns job j the rank j. The ranking conveys only relative order: it fixes no numerical estimate of any p_j, and instances with arbitrarily different processing times may carry the same prediction.

A ranking is always available from numerical estimates: order the jobs by $\hat{p}$, so that $\hat{p}_i < \hat{p}_j$ implies $\hat{\pi}(i) < \hat{\pi}(j)$, breaking ties among equal estimates arbitrarily. The result is a bijection whether or not the estimates are accurate. Reading only a ranking is the harder task of the two: numerical estimates

determine a ranking, so whatever an ordinal algorithm achieves a numerical one achieves as well, while the converse need not hold. The lower bound of Wei & Zhang (2020) whose order we match is proved for algorithms given numerical estimates, and therefore binds those given only a ranking (Appendix D). Matching it from a ranking alone is thus a stronger statement than matching it with the numbers in hand.

Example 1. *Take three jobs a, b and c with requirements 5, 2 and 9. A correct ordinal prediction ranks b first, then a, then c, that is $\hat{\pi}(b) = 1$, $\hat{\pi}(a) = 2$ and $\hat{\pi}(c) = 3$; the prediction $\hat{\pi}(c) = 1$, $\hat{\pi}(b) = 2$, $\hat{\pi}(a) = 3$ is incorrect, since it gives the longest job the smallest rank. Indexing in Shortest Processing Time (SPT) order renames b, a, c as 1, 2, 3, so that $p_1 = 2$, $p_2 = 5$, $p_3 = 9$ and $\mathrm{OPT} = 3 \cdot 2 + 2 \cdot 5 + 1 \cdot 9 = 25$; after this renaming the correct prediction is the one that assigns job j the rank j. A numerical prediction is correct when it gives the true requirements, so $\hat{p} = (5, 2, 9)$ is correct for these jobs while $\hat{p} = (6, 3, 20)$ is not, even though the two induce the same ranking. Ordinal correctness is the weaker requirement, and it is scale-blind: three jobs with requirements 1, 100 and 1000 carry the same correct ordinal prediction as the three above.*

Consistency and robustness. We write ALG for the total completion time $\sum_j C_j$ that an algorithm produces on a given instance and prediction. Both consistency and robustness compare ALG with the optimum of the same instance and quantify over all instances; they differ only in which predictions they admit. The algorithm is β-*consistent* if $\mathrm{ALG} \le \beta \cdot \mathrm{OPT}$ for every instance and every prediction that is correct for that instance, and γ-*robust* if $\mathrm{ALG} \le \gamma \cdot \mathrm{OPT}$ for every instance and every prediction, correct or not.

Pairwise delays. Our analysis uses a standard decomposition of total completion time into pairwise interactions. For an algorithm A and distinct jobs i and j, let $D^A_{i,j}$ denote the amount of service that A gives to job i by the time job j completes, and let $\delta^A_{i,j} = D^A_{i,j} + D^A_{j,i}$ be their *pairwise delay*. Since job i receives at most p_i units of service by the time job j completes, $D^A_{i,j} \le p_i$, with equality whenever i completes no later than j. Sums of the form $\sum_{\{i,j\}}$ range over unordered pairs of distinct jobs. The decomposition below says that a job's completion time is its own work plus whatever the machine spent on other jobs before it finished. Since the machine is never idle, the time elapsed at any moment is exactly the work done by then, so summing over jobs splits the total completion time into a diagonal term, the sum of the requirements, which every algorithm pays, and one term per pair, which measures how much the two jobs delayed each other. For the optimum each pair contributes as little as it can: the shorter job runs to completion first, so the pair costs exactly the shorter requirement.

Lemma 2 (Pairwise decomposition, Motwani et al., 1994)**.** *For any algorithm A, $C^A_j = p_j + \sum_{i \ne j} D^A_{i,j}$, and hence $\sum_j C^A_j = \sum_j p_j + \sum_{\{i,j\}} \delta^A_{i,j}$. For the SPT optimum, $\delta^{\mathrm{OPT}}_{i,j} = \min\{p_i, p_j\}$, so $\mathrm{OPT} = \sum_j p_j + \sum_{\{i,j\}} \min\{p_i, p_j\}$.*

The diagonal term $\sum_j p_j$ appears unchanged in both the algorithmic cost and OPT, so all excess cost comes from pairwise interactions. That diagonal is never a negligible part of the optimum, and the next lemma says how small it can be: at least a $2/(n+1)$ fraction, attained exactly when all jobs have the same size. This is what sharpens the consistency guarantee. An algorithm that pays a surcharge on the pairwise terms pays none on the diagonal, so a factor $1 + \varepsilon$ on the pairs becomes the smaller factor $1 + \left(1 - \frac{2}{n+1}\right)\varepsilon$ overall.

Lemma 3 (Diagonal lower bound)**.** $\sum_j p_j \ge \frac{2}{n+1} \cdot \mathrm{OPT}$*, with equality if and only if all processing times are equal.*

Round Robin. This algorithm ignores the prediction and shares the machine equally: whenever k jobs remain unfinished, each receives rate $1/k$. Every unfinished job therefore carries the same accrued service at every instant. For distinct jobs i and j, equal sharing gives both of them $\min\{p_i, p_j\}$ units of service by the time the shorter one completes, so $\delta^{\mathrm{RR}}_{i,j} = 2\min\{p_i, p_j\}$, twice the corresponding term for the optimum. Lemma 2 then gives the exact cost

$$\sum_j C^{\mathrm{RR}}_j = \sum_j p_j + 2\sum_{\{i,j\}} \min\{p_i, p_j\} = 2 \cdot \mathrm{OPT} - \sum_j p_j,$$

and Lemma 3 converts this into the classical guarantee $\sum_j C^{\mathrm{RR}}_j \le \left(2 - \frac{2}{n+1}\right)\mathrm{OPT}$, which is optimal among deterministic non-clairvoyant algorithms (Motwani et al., 1994). Both the diagonal $\sum_j p_j$ and the bound of Lemma 3 are therefore tight exactly when all processing times are equal.

3 PREDICTION-DRIFT ROUND ROBIN

We now define PDRR, together with the one structural property its analysis needs. In Section 3.1 we motivate the multiplicative lead and state the algorithm. In Section 3.2 we prove the invariant that bounds how far ahead of the others any one job can be, on which both the consistency and the robustness bounds are built.

3.1 THE ALGORITHM

Two extreme algorithms stand behind PDRR: one ignores the prediction, the other follows it without reservation. The known way of combining them is *time sharing* (Kumar et al., 2018), and PDRR differs from it precisely in how the two are put together.

At one extreme is Round Robin (RR), which ignores the prediction entirely: it divides the machine equally among the unfinished jobs, so at every instant they all carry the same accrued service, that is, the same amount of processing received so far. A job that turns out to be long therefore never runs ahead of the others, which is why RR is never worse than a factor 2 from the optimum; but by the same rule a job predicted to be short is never served first, so the prediction brings no gain. At the other extreme is *Follow the Prediction* (FP), which runs the jobs one at a time in the predicted order: it is optimal when the prediction is correct, while on a wrong order its ratio can reach $\Theta(n)$.

Time sharing (Kumar et al., 2018; Lindermayr & Megow, 2022) combines RR and FP by dividing the machine's speed between them: a fixed fraction λ of the unit speed goes to FP and the remaining $1-\lambda$ to RR. PDRR combines RR and FP in a different way. It preserves the common level of RR for every job but one. The exception is the *preferred* job, the one of smallest predicted rank, which is the job FP would be running. It alone may run ahead of the others: its accrued service may exceed the common level, but only by a bounded factor. How large that factor is depends on the one parameter of the algorithm, the *drift* $\varepsilon \in (0, 1]$. The preferred job may stand at most $f = 1/\varepsilon \geq 1$ times the common level, and we call f the *lead factor*. At $\varepsilon = 1$ the lead factor is 1, the two levels coincide, and PDRR degenerates to RR. As $\varepsilon \to 0$ the permitted gap grows without bound and PDRR approaches FP; this end is a limit rather than a member of the family, since $\varepsilon = 0$ leaves the lead factor $f = 1/\varepsilon$ undefined.

Algorithm 1 states PDRR in full, and the rest of this subsection follows it. At any time let A be the set of unfinished jobs. The preferred job is then $q = \mathrm{pref}(A)$, the job in A of smallest predicted rank, which is unique since $\hat{\pi}$ is a bijection, and the remaining unfinished jobs form the *pack*. All pack jobs stand at one common level of accrued service, the *pack level* s, and the preferred job may stand at up to f times it; that ceiling, $f \cdot s$, is its *lead*. The algorithm alternates two kinds of phase. In a *catch-up* (lines 4–8) the preferred job runs alone until it reaches its lead, and the pack stands still. In an *advance* (lines 9–12) the pack rises again, the preferred job being served just fast enough to keep its lead, until some job completes.

Initially nothing has been served, so the pack level is zero, and the lead, being f times the pack level, is zero as well. The preferred job is therefore already at its lead, the first catch-up is empty, and PDRR begins with an advance.

Catch-up (lines 4–8). Catch-up is activated when, and only when, the preferred job completes: a successor must then be picked and brought up to the lead. The successor runs alone, and two things can happen. Either its accrued service reaches the lead, and the algorithm moves to an advance; or it completes before doing so, and the next job in predicted order is picked and runs alone in its turn. Several jobs may finish one after another, each running alone, before any advance takes place.

Advance (lines 9–12). Once the preferred job sits at its lead, the pack resumes rising and the preferred job is served just fast enough to keep the lead. With a pack jobs, raising the pack level by one unit takes a units of service on the pack and f on the preferred job, a total of $a + f$. Dividing the machine in that proportion gives each pack job the rate $1/(a + f)$ and the preferred job the rate $f/(a + f)$, which uses the machine fully. The phase ends as soon as some job completes, and the algorithm returns to catch-up.

Three properties follow directly from the definition. PDRR never consults a processing requirement: the only external information it uses is the prediction and the completions it observes, which is what makes it implementable in the non-clairvoyant model. It does not even track how much service the

Algorithm 1: PDRR, drift $\varepsilon \in (0, 1]$, $f = 1/\varepsilon$

Require: jobs J, ranking $\hat{\pi}$, drift ε
all jobs at service 0; $s \leftarrow 0$
while some job is unfinished **do**
  $A \leftarrow$ unfinished jobs; $q \leftarrow \mathrm{pref}(A)$
  ▷ *Catch-up*: raise q to the lead $f \cdot s$
  run q alone until its service reaches $f \cdot s$, or until it completes
  if q completes **then**
    remove q and **continue** ▷ next preferred job
  end if
  $a \leftarrow |A| - 1$ ▷ pack jobs
  ▷ *Advance*: raise the pack, holding q at $f \cdot s$
  increase s at rate $1/(a + f)$, serving each pack job at $1/(a + f)$ and q at $f/(a + f)$, until some job completes
  remove all jobs that completed
end while

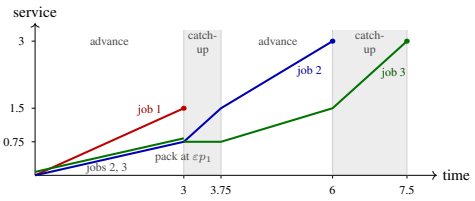

Figure 1: A run of PDRR with $\varepsilon = \frac{1}{2}$, so $f = 2$, on three correctly predicted jobs, $p_1 = 1.5$ and $p_2 = p_3 = 3$. Curves show accrued service; job 3 is drawn slightly above job 2 before $t = 3$, where the two coincide, so that both remain visible. Shaded intervals are catch-up phases, in which the preferred job runs alone; unshaded intervals are advance phases, in which the pack rises. The preferred job is held at twice the pack level, and the pack stands at εp_1 when job 1 completes.

individual jobs have received. Its whole state is the set of unfinished jobs together with the single number s, since every pack job stands at s by construction and a catch-up lasts $(f-1)s$: no pack job is served during it, so the pack level stays at s while the preferred job climbs from s to $f \cdot s$. A job is preferred through a single contiguous stretch ending at its completion, since once preferred it never rejoins the pack; the robustness analysis relies on this. And because the lead is a ratio rather than an absolute amount, PDRR introduces no absolute scale of its own, so multiplying every processing requirement by a common factor rescales the whole schedule by that factor. Figure 1 traces a small run. Its total is 16.5 against $\mathrm{OPT} = 13.5$, a ratio of 1.22, within the guarantee 1.25 of Theorem 7.

3.2 The multiplicative invariant

Both the consistency and the robustness analysis rest on a single structural fact: at any moment, the accrued service of an unfinished job lies between the pack level and f times the pack level.

Lemma 4 (Multiplicative invariant)**.** *At any moment while some job is unfinished, let s be the pack level. Every unfinished job has then accrued at least s and at most $f \cdot s$.*

Proof idea. The pack jobs rise together from service 0, so each carries exactly s. A job enters the preferred role carrying s and only gains service afterwards, so it too carries at least s. It never passes $f \cdot s$: during a catch-up its service rises from s toward $f \cdot s$ and the phase ends when it arrives there, and during an advance it is held at $f \cdot s$ exactly while the pack rises beneath it.

The two analyses use the invariant in different ways.

Consistency. Under a correct prediction the preferred job completes before any strictly larger job, and it does so exactly at its lead. The pack level when a job of size p_i completes is therefore $s = p_i/f = \varepsilon p_i$, every job still unfinished has accrued exactly that much, and each such pair incurs only εp_i of delay beyond its SPT contribution. Section 4 proves this and turns it into the guarantee; Figure 1 shows the pack standing at $0.75 = \varepsilon p_1$ at the moment job 1 completes.

Robustness. Here nothing is assumed about the prediction, so the preferred job may be far larger than the others, and the invariant is all that limits how much service such a job absorbs before a smaller one finishes. Fix $i < j$ and let s be the pack level when job i completes. The invariant caps job j at $f \cdot s$; it also places every unfinished job at s or above, so $p_i \geq s$, job i having been unfinished until that moment. Job j has therefore received at most $f \cdot p_i$, and with the trivial bound p_j this gives $D_{j,i}^{\mathrm{PDRR}} \leq \min\{p_j,\ f \cdot p_i\}$, the pairwise estimate that Section 5 sums over all pairs.

The cap could in principle be any increasing function of the pack level, and the bound above would still follow. The multiplicative form is nevertheless forced: consistency $1 + \varepsilon$ requires a cap of at least s/ε at every scale, while the Cauchy–Schwarz step in the proof of Lemma 11 requires one of at most $L\,s$ and returns robustness $2\sqrt{L}$. The two meet only at $f \cdot s$, where both hold with equality.

4 CONSISTENCY

We assume in this section that the prediction is correct. With the jobs indexed so that $p_1 \le \cdots \le p_n$, and ties broken as in Section 2, this means $\hat{\pi}(j) = j$ for every job j. We show that PDRR is $\big(1 + (1 - \frac{2}{n+1})\varepsilon\big)$-consistent, and that this factor is attained, so it tends to $1 + \varepsilon$ as n grows.

The argument has three steps, joined by the pairwise decomposition of Lemma 2. First, a correct prediction makes PDRR complete the jobs in order of size (Lemma 5), so the preferred job is always the smallest unfinished one. Second, a preferred job then completes exactly at its lead, never before reaching it; this fixes the pack level at εp_i when job i completes, and turns every pairwise delay into $(1+\varepsilon)\min\{p_i, p_j\}$ (Lemma 6), a surcharge of exactly ε on each pair. Third, the decomposition adds these up. The surcharge falls only on the pairwise terms and never on the diagonal $\sum_j p_j$, which every algorithm pays in full; since the diagonal is at least a $2/(n+1)$ fraction of the optimum (Lemma 3), the overall factor is $1 + \big(1 - \frac{2}{n+1}\big)\varepsilon$ rather than $1 + \varepsilon$ (Theorem 7). Equal jobs are the one instance where that diagonal bound is tight, and they are exactly the instance that attains the guarantee (Proposition 8).

Lemma 5. *Under a correct prediction, $C_1 \le C_2 \le \cdots \le C_n$.*

Proof idea. While both jobs are unfinished, the one of larger predicted rank never carries more service than the one of smaller rank, and it is no shorter, so it cannot finish first.

Lemma 6. *Under a correct prediction, when job i completes the pack level is $s = \varepsilon p_i$. Consequently, for every pair $i < j$, $\delta_{i,j}^{\mathrm{PDRR}} = (1+\varepsilon)\, p_i = (1+\varepsilon)\min\{p_i, p_j\}$.*

Proof idea. By induction on i. Under a correct prediction the preferred job is no shorter than the job before it, whose size is its lead, so it cannot complete before reaching its lead; it therefore completes exactly at its lead, when the pack level is $p_i/f = \varepsilon p_i$. Every larger job is still in the pack at that moment, so its accrued service stands at exactly εp_i in total, and the pair delay is $p_i + \varepsilon p_i$.

Theorem 7 (Consistency)**.** *Under a correct prediction,*

$$\sum_j C_j^{\mathrm{PDRR}} \;=\; (1+\varepsilon)\mathrm{OPT} - \varepsilon \sum_j p_j \;\le\; \Big(1 + \big(1 - \tfrac{2}{n+1}\big)\varepsilon\Big)\mathrm{OPT} \;\le\; (1+\varepsilon)\cdot \mathrm{OPT}.$$

Proof idea. Lemma 6 turns every pairwise delay into $(1+\varepsilon)\min\{p_i, p_j\}$, so in the decomposition of Lemma 2 the surcharge ε multiplies the off-diagonal mass $\mathrm{OPT} - \sum_j p_j$ and leaves the diagonal untouched. Lemma 3 then converts the identity into a uniform guarantee.

Proposition 8 (Tightness of the consistency bound)**.** *For every n and every $\varepsilon \in (0, 1]$, there is a correctly predicted instance on which PDRR has competitive ratio $1 + (1 - \frac{2}{n+1})\varepsilon$. Consequently, the finite-n consistency guarantee of Theorem 7 is tight, and the ratio tends to $1 + \varepsilon$ as $n \to \infty$.*

Proof idea. Take n jobs of equal size, the instance on which Lemma 3 is tight, and read off the identity of Theorem 7.

5 ROBUSTNESS

Throughout this section the prediction is arbitrary; in particular, it need bear no relation to the SPT order, so the preferred job may be of any size. Recall that jobs are indexed so that $p_1 \le \cdots \le p_n$, and that $f = 1/\varepsilon \ge 1$. We show that PDRR is $\min\big\{\frac{2}{\sqrt{\varepsilon}},\, 2 + \frac{1}{\sqrt{\varepsilon}}\big\}$-robust (Theorem 14), and that its own worst case is not far below this (Proposition 16).

The argument has four steps. The first isolates what can go wrong. In the pairwise decomposition the optimum already pays for the diagonal and for the service that smaller jobs receive before larger ones finish, so everything turns on the *larger-to-smaller* delay B: the service a large job absorbs before a smaller one completes. The second bounds one such delay. The multiplicative invariant caps it at $f \cdot p_i$, and no job receives more than its own size, so it is at most $\min\{p_j,\, f \cdot p_i\}$ (Lemma 9); this is the only step where the algorithm enters, and everything after it is arithmetic on job sizes. The third sums that bound over all pairs. As a layer-cake integral the sum pairs the profile of job sizes with a copy of itself rescaled by f, and Cauchy–Schwarz costs only $\sqrt{f}$ rather than f (Lemma 11). This is where the optimal order comes from, and it already gives robustness $2/\sqrt{\varepsilon}$ (Proposition 12).

The fourth lowers the leading coefficient from 2 to 1: subtracting from each delay the service Round Robin would have delivered anyway leaves the *preferred excess*, which can be charged to disjoint layers of the optimum (Lemma 13) and yields $2 + 1/\sqrt{\varepsilon}$, the smaller bound when $\varepsilon < \frac{1}{4}$.

5.1 Reduction to the larger-to-smaller delay

By Lemma 2, $C_j^A = p_j + \sum_{i \neq j} D_{i,j}^A$; summing over all jobs and splitting each pair according to the SPT order,

$$\sum\nolimits_j C_j^A = \sum\nolimits_j p_j + \sum\nolimits_{i<j} D_{i,j}^A + \sum\nolimits_{i<j} D_{j,i}^A.$$

For PDRR the first two groups are paid for by the optimum alone: since $D_{i,j}^{\mathrm{PDRR}} \leq p_i$ for every $i < j$,

$$\sum\nolimits_j p_j + \sum\nolimits_{i<j} D_{i,j}^{\mathrm{PDRR}} \;\leq\; \sum\nolimits_j p_j + \sum\nolimits_{i<j} p_i \;=\; \sum\nolimits_i (n-i+1)p_i \;=\; \mathrm{OPT}.$$

Everything therefore reduces to the *larger-to-smaller* delay:

$$\sum\nolimits_j C_j^{\mathrm{PDRR}} \;\leq\; \mathrm{OPT} + B, \qquad B = \sum\nolimits_{i<j} D_{j,i}^{\mathrm{PDRR}}. \tag{1}$$

The simple bound below shows $B \leq (2\sqrt{f}-1)\mathrm{OPT}$; the refined bound shows $B \leq (1+\sqrt{f})\mathrm{OPT}$.

5.2 A uniform delay bound and the optimal order

Lemma 9 (Uniform delay bound)**.** *For every pair $i < j$ and every prediction, $D_{j,i}^{\mathrm{PDRR}} \leq \min\{p_j,\, f \cdot p_i\}$.*

Proof idea. The bound p_j is immediate. For the other, let s be the pack level when i completes. The multiplicative invariant caps j's service at $f \cdot s$, and every unfinished job carries at least s, so applying this to i at its completion gives $s \leq p_i$.

Example 10. *Take two jobs, $p_1 = 1$ and $p_2 = 100$, with $\varepsilon = \frac{1}{2}$ so $f = 2$, and a prediction that ranks the long job first. Job 2 is preferred throughout, yet by the time job 1 completes it has received only $f \cdot p_1 = 2$ units of service, which is the bound of Lemma 9. Follow the Prediction would instead run job 2 to completion first and delay job 1 until time 101. Capping the lead is what turns a delay of p_2 into a delay of $f \cdot p_1$.*

Let $N(x) = \big|\{j : p_j \geq x\}\big|$. The layer-cake identity gives $\mathrm{OPT} = \sum_{i \leq j} \min\{p_i, p_j\} = \int_0^\infty \binom{N(x)+1}{2}\, dx$ and $\int_0^\infty N(x)^2\, dx = \sum_{i,j} \min\{p_i, p_j\} = 2\mathrm{OPT} - \sum_i p_i$.

Lemma 11 (Min-kernel inequality)**.** *For all reals $0 \leq p_1 \leq \cdots \leq p_n$ and every $f \geq 1$,*

$$\sum\nolimits_{i<j} \min\{p_j,\, f \cdot p_i\} \;\leq\; (2\sqrt{f} - 1) \cdot \mathrm{OPT}.$$

Proof idea. Write both sides as layer-cake integrals in the profile $N(x) = |\{j : p_j \geq x\}|$. The left-hand side becomes $\int_0^\infty N(x)N(x/f)\, dx - \mathrm{OPT}$, which pairs the profile with a copy of itself rescaled by f. Cauchy–Schwarz together with the substitution $y = x/f$ bounds that integral by $\sqrt{f}\int_0^\infty N(x)^2\, dx \leq 2\sqrt{f} \cdot \mathrm{OPT}$. The square root of f, rather than f itself, is what the optimal order rests on.

Proposition 12 (Optimal order)**.** *For every $\varepsilon \in (0, 1]$ and every prediction, $\sum_j C_j^{\mathrm{PDRR}} \leq 2\sqrt{f} \cdot \mathrm{OPT} = \frac{2}{\sqrt{\varepsilon}}\mathrm{OPT}$, matching the order of the Wei–Zhang lower bound.*

5.3 Refining the constant: excess layers

Fix a pair $i < j$. Until job i completes the pack level is at most p_i, so the part of j's service at or below the contemporaneous pack level is at most p_i. Any further service j receives before i completes is obtained while j is preferred and lies strictly above the pack level; we call this the *preferred excess* $E_{j,i} \geq 0$. Thus

$$D_{j,i}^{\mathrm{PDRR}} \leq p_i + E_{j,i}, \qquad B \leq \sum\nolimits_{i<j} p_i + \sum\nolimits_{i<j} E_{j,i} = \mathrm{OPT} - \sum\nolimits_i p_i + E, \tag{2}$$

where $E = \sum_{i<j} E_{j,i}$. By equation 1, it therefore suffices to prove $E \leq \sqrt{f} \cdot \mathrm{OPT}$, for then $B \leq \mathrm{OPT} - \sum_i p_i + \sqrt{f} \cdot \mathrm{OPT} \leq (1 + \sqrt{f})\mathrm{OPT}$.

The decisive point is to parameterize preferred excess by its *true service height* x, rather than by the contemporaneous pack level. For a preferred-excess layer at height x, define the normalized level $s = x/f$. Since the actual pack level u satisfies $x \leq f \cdot u$, we have $s \leq u$. The change of variables $x = f \cdot s$ contributes the Jacobian f, which is what ultimately yields the $\sqrt{f}$ rather than f dependence in the charging argument, and enables the leading coefficient 1 in the refined bound.

Lemma 13 (Excess-layer bound)**.** *For every instance and every prediction,* $E \leq \sqrt{f} \cdot \mathrm{OPT}$.

Proof idea. Each unit of preferred excess is indexed by its *true* service height x rather than by the contemporaneous pack level, and normalized to $s = x/f$; the change of variables contributes the Jacobian f, which is what turns f into $\sqrt{f}$. At each normalized level the excess is at most $f\, m(s) r(s)$, where $m(s)$ counts the smaller unfinished jobs and $r(s)$ the jobs owning an excess layer. Splitting that product by the arithmetic–geometric mean inequality and charging the two halves to the layer-cake decomposition of OPT gives $E \leq \sqrt{f} \cdot \mathrm{OPT}$. The charges are disjoint because a job never returns to the pack once it has been preferred, so no layer is charged twice.

Theorem 14 (Robustness)**.** *For every* $\varepsilon \in (0, 1]$ *and every prediction,*

$$\sum_j C_j^{\mathrm{PDRR}} \leq \min\Big\{2/\sqrt{\varepsilon},\ 2 + 1/\sqrt{\varepsilon}\Big\} \cdot \mathrm{OPT},$$

where the second bound is the smaller one exactly when $\varepsilon < \frac{1}{4}$.

Remark 15 (Round-Robin endpoint)**.** *At* $\varepsilon = 1$ *we have* $f = 1$*, the preferred job has no advantage over the pack, and PDRR is Round Robin. The simple bound* $2\sqrt{f}$ *returns the classical value* 2*, while the refined bound returns* 3*; the combined bound is therefore* 2*, as it must be.*

Proposition 16 (Tightness of the leading constant)**.** *For every* $f > 1$ *and every* $\zeta > 0$ *there is an instance and a prediction on which PDRR has ratio at least* $\sqrt{f} + \frac{1}{2} + \frac{1}{2\sqrt{f}} - \zeta$*. Consequently the worst-case robustness ratio of PDRR is at least* $\sqrt{f} + \frac{1}{2} + \frac{1}{2\sqrt{f}}$.

Proof idea. Take m jobs of size 1 and r jobs of size κf with $1/f < \kappa < 1$, predicted in the worst order, all large jobs first. Every job but the first completes with the pack level equal to κ, which makes all pairwise delays explicit. Letting $\kappa \to 1$ and $r \to \infty$ with $m/r \to \sqrt{f}$ balances the two quadratic terms and yields the stated ratio.

For every $f > 1$, the construction above together with Theorem 14 places the worst-case ratio of PDRR, that is, the supremum of $\sum_j C_j^{\mathrm{PDRR}}/\mathrm{OPT}$ over all instances and all predictions, between $\sqrt{f} + \frac{1}{2} + \frac{1}{2\sqrt{f}}$ and $\sqrt{f} + 2$. Hence this ratio is $\sqrt{f} + \Theta(1) = 1/\sqrt{\varepsilon} + \Theta(1)$ as $\varepsilon \to 0$. Thus, the leading constant 1 is best possible for PDRR, and the remaining gap is additive and less than $3/2$.

6 Experiments

Appendix C evaluates PDRR against time sharing and the algorithms of Benomar et al. (2025) on heavy-tailed instances under Gaussian prediction noise, illustrating the consistency–robustness trade-off rather than measuring worst-case ratios. Among methods tuned to the same worst-case guarantee, PDRR has the smallest ratio when the predicted order is accurate, even though it alone sees only the order and not the predicted values, and it stays near the Round Robin value once the order carries no information, where time sharing keeps deteriorating. Raising the permitted robustness from 4 to 8 improves PDRR's measured ratio under a correct order from 1.25 to 1.03, while the baselines improve only from 1.36 and 1.47 to about 1.20.

7 Conclusion

PDRR closes the quadratic consistency–robustness gap that had remained open for non-clairvoyant scheduling on general instances with static predictions. It does so from an ordinal prediction alone, so numerical job-size predictions are not necessary to reach the optimal order. Extending the drift principle to weighted objectives, release dates and parallel machines, improving the leading constant, and establishing a smooth dependence on prediction error (Benomar & Perchet, 2025) are open problems.

AI USE STATEMENT

In this work, we used generative AI tools to implement PDRR within the experimental code released with Benomar et al. (2025); the authors verified the correctness of this implementation. We have not used generative AI tools for the other tasks with required disclosure, which are either not applicable to this work or were carried out by the authors. Additionally, we used generative AI tools to rewrite and edit the text of the paper. We have reviewed all AI-assisted work, and we take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ziyad Benomar and Vianney Perchet. Non-clairvoyant scheduling with partial predictions. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 3506–3538. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/benomar24a.html.

Ziyad Benomar and Vianney Perchet. On tradeoffs in learning-augmented algorithms. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Emtiyaz Khan (eds.), *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 802–810. PMLR, 03–05 May 2025. URL https://proceedings.mlr.press/v258/benomar25a.html.

Ziyad Benomar, Romain Cosson, Alexander Lindermayr, and Jens Schlöter. Non-clairvoyant scheduling with progress bars. In *Advances in Neural Information Processing Systems 38*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/868f2266086530b2c71006ea1908b14a-Abstract-Conference.html.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pp. 89–96, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931805. doi: 10.1145/1102351.1102363. URL https://doi.org/10.1145/1102351.1102363.

Alex Elenter, Spyros Angelopoulos, Christoph Dürr, and Yanni Lefki. Overcoming brittleness in pareto-optimal learning augmented algorithms. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 9329–9357. Curran Associates, Inc., 2024. doi: 10.52202/079017-0296. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/11c6625b0481a7d5625831369f6b7c82-Paper-Conference.pdf.

Marek Eliáš, Haim Kaplan, Yishay Mansour, and Shay Moran. Learning-augmented algorithms with explicit predictors. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient LLM scheduling by learning to rank. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 59006–59029. Curran Associates, Inc., 2024. doi: 10.52202/079017-1882. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/6c8985579293e0209bdaa4f21bb1d237-Paper-Conference.pdf.

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*, SPAA '21, pp. 285–294, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380706. doi: 10.1145/3409964.3461790. URL https://doi.org/10.1145/3409964.3461790.

Ravi Kumar, Manish Purohit, and Zoya Svitkina. Improving online algorithms via ml predictions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 9684–9693, Red Hook, NY, USA, 2018. Curran Associates Inc.

Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, pp. 357–368, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391467. doi: 10.1145/3490148.3538579. URL https://doi.org/10.1145/3490148.3538579.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3296–3305. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/lykouris18a.html.

Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, August 1994. ISSN 0304-3975. doi: 10.1016/0304-3975(94)90151-1. URL https://doi.org/10.1016/0304-3975(94)90151-1.

Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi: https://doi.org/10.1002/nav.3800030106. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106.

Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.