Online Flow Time Minimization with Gradually Revealed Jobs

Alexander Lindermayr
Institut für Mathematik, Technische Universität Berlin, Germany

Guido Schäfer
Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Jens Schlöter
Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Leen Stougie
Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Abstract

We consider the problem of online preemptive scheduling on a single machine to minimize the total flow time. In clairvoyant scheduling, where job processing times are revealed upon arrival, the Shortest Remaining Processing Time (SRPT) algorithm is optimal. In practice, however, exact processing times are often unknown. At the opposite extreme, non-clairvoyant scheduling, in which processing times are revealed only upon completion, suffers from strong lower bounds on the competitive ratio. This motivates the study of intermediate information models. We introduce a new model in which processing times are revealed gradually during execution. Each job consists of a sequence of operations, and the processing time of an operation becomes known only after the preceding one completes. This models many scheduling scenarios that arise in computing systems.

Our main result is a deterministic $O(m^2)$-competitive algorithm, where m is the maximum number of operations per job. More specifically, we prove a refined competitive ratio in $O(m_1 \cdot m_2)$, where m_1 and m_2 are instance-dependent parameters describing the operation size structure. Our algorithm and analysis build on recent advancements in robust flow time minimization (SODA '26), where jobs arrive with estimated sizes. However, in our setting we have no bounded estimate on a job's processing time. Thus, we design a highly adaptive algorithm that gradually explores a job's operations while working on them, and groups them into virtual chunks whose size can be well-estimated. This is a crucial ingredient of our result and requires a much more careful analysis compared to the robust setting. We also provide lower bounds showing that our bounds are essentially best possible. For the special case of scheduling with uniform obligatory tests, we show that SRPT at the operation level is 2-competitive, which is best possible.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases optimization, scheduling, online algorithms, competitive analysis, flow time, non-clairvoyance

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.128

Related Version *Full Version*: `https://arxiv.org/abs/2602.12716` [31]

Funding *Alexander Lindermayr*: The main part of this work was done while the author was a research fellow at the Simons Institute for the Theory of Computing for the program on "Algorithmic Foundations for Emerging Computing Technologies" in Fall 2025.
Jens Schlöter: Supported by the Netherlands Organisation for Scientific Research (NWO) through project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)".

Leen Stougie: Supported in part by the Netherlands Organisation for Scientific Research (NWO) through project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)" and Gravitation-project NETWORKS-024.002.003.

1 Introduction

We study the fundamental problem of online preemptive flow time minimization on a single machine: n jobs arrive online over time and must be scheduled with preemption. The objective is to minimize the total flow time, that is, the sum over jobs of their time in the system. This is a central quality-of-service metric with applications, e.g., in networking, cloud computing, and operating systems. A classical result for this problem is that SRPT (Shortest Remaining Processing Time), which always runs the job with the least remaining work, is optimal in this setting [34].

In practice, however, we cannot generally assume that a scheduler has access to the exact processing times p_j of jobs, and therefore cannot implement SRPT directly. Already in the 1990s, this led to the study of *non-clairvoyant* algorithms, which only learn a job's processing time once it has completed. From a theoretical perspective, this model lies at the other extreme of the spectrum and strong lower bounds are known: no non-clairvoyant algorithm can be $o(\log n)$-competitive [32].

A major research direction in online scheduling has therefore been to study intermediate models that bridge the gap between knowing everything (clairvoyance) and knowing nothing (non-clairvoyance), by providing information about a job before it completes [14, 12, 6, 38]. Only recently, two prominent models were introduced to achieve this goal:

- **Predictions.** Each job j arrives with a predicted processing time $\hat{p}_j$ [6]; this is also called *robust* flow time scheduling. Recently, Gupta et al. [24] gave a best-possible $O(\mu)$-competitive algorithm, where $\mu := \mu_1 \cdot \mu_2$ and $\mu_1 = \max_j \hat{p}_j/p_j$ and $\mu_2 = \max_j p_j/\hat{p}_j$ are the maximum overestimation and underestimation factors. A natural critique of this model is that the prediction must be available at arrival, when job and scheduler have not yet interacted. Therefore, it is unclear how such predictions could be obtained in many real-world applications.
- **ε-Clairvoyance.** The downside of predictions is mitigated in the ε-clairvoyant model, where a scheduler learns about a job's processing time once a $(1-\varepsilon)$-fraction of its processing time is done (hence an ε-fraction remains) [38]. Gupta et al. [23] presented a best-possible $\lceil 1/\varepsilon \rceil$-competitive algorithm for all $\varepsilon \in (0, 1]$. While this allows the algorithm to learn about jobs through interaction, the requirement of learning the processing time of a job at a single specific point is a strong assumption that is hard to justify in practice.

To overcome these drawbacks, we introduce a new model in which job information is revealed gradually through execution and that has strong connections to both theory and practice.

From a practical perspective, we move beyond the black-box abstraction and examine what jobs in fundamental computing applications are: programs represented by control-flow graphs [1, 2, 3]. In system design and architecture, it is well established that programs exhibit distinct phase behavior, executing sequences of basic blocks or operations over time [35]. We model this structure formally using a decision tree for each job. We assume that a scheduler can determine the execution time of any deterministic operation *on a given input* [27]. Initially, it knows the processing time of the root operation (or phase). When the root completes, the program branches based on its result and reveals the next operation, along with its processing time. This repeats until the job completes. For deterministic programs, an offline optimum knows the entire sequence of operations and processing times in advance. See Figure 1 for an illustrative example.

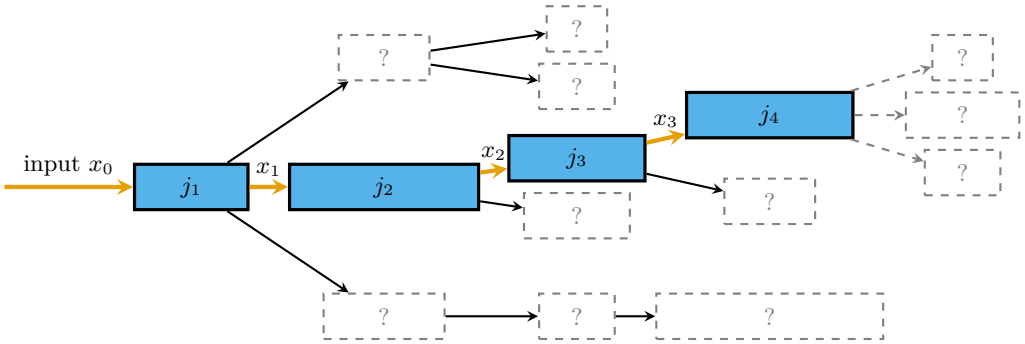

Figure 1 Illustration of the decision tree model. Operations are rectangles with widths proportional to processing time. Edges are labeled with intermediate results x_i that determine the length of the next operation. The blue operations are realized operations. Currently, the job is at operation j_4. Future operations are unknown until the preceding intermediate result is computed.

Formally, we consider the following *operation flow time scheduling problem*. A job j is composed of m *operations* $j_1, \dots, j_m$ with operation processing times $p_{j_1}, \dots, p_{j_m}$ and job processing time $p_j = \sum_{\ell=1}^{m} p_{j_\ell}$. When job j arrives at time r_j, we say that j and its first operation j_1 become active and known. Once the first operation is completed, the next operation j_2 becomes active and known to the scheduler, and so on, until all operations are completed and the job completes. We give a precise definition later. It is helpful to think of operations not as single instructions, but as small, linear subprograms that need to be completed before the overall program branches.

From a theoretical perspective, special cases of our operation flow time scheduling problem have deep connections to existing models and results in online flow time scheduling literature:

- If all operations have lengths in $\{0, 1\}$, a classic result of Motwani, Phillips, and Torng is that running jobs to completion is m-competitive and best-possible, because an optimal solution can finish at most m short jobs while the algorithm works on a long job [32].
- If the operation lengths of each job are monotone non-decreasing, we show that executing SRPT on the operation level (i.e., at any time schedule the job with the shortest remaining *operation* processing time), which we call *Operations-SRPT*, is m-competitive, and this is best-possible (cf. Section 3).
- If the length of the first operation, which is available at job arrival, underestimates p_j by at most a factor of m, we can treat p_{j_1} as a prediction for p_j and obtain a best-possible $O(m)$-competitive algorithm [6, 7, 24]. However, for general instances the length of the first operation can be an arbitrarily bad estimate of the total job size.

We capture all of these aspects in a single algorithm and analyze it for general instances.

1.1 Our Results and Techniques

Our first main result is a bounded competitive ratio in terms of the number of operations m.

▶ **Theorem 1.** *For scheduling with m operations, there exists a deterministic $O(m^2)$-competitive algorithm for minimizing the total flow time on a single machine.*

Our algorithm is highly adaptive and carefully explores the operation structure of jobs while optimistically scheduling them. At a high level, we treat the first operation length p_{j_1} as a "prediction" of p_j and start to run the robust algorithm of [24] using this prediction. In contrast to the robust setting [24, 6, 7], we have no guarantee that p_{j_1} is a good

approximation of p_j as it might underestimate p_j by an arbitrarily large factor. However, as long as subsequent operations of j have length at most p_{j_1}, the prediction p_{j_1} underestimates the total processing time of the revealed part of job j by at most a factor of m, so we can keep running the robust algorithm and retain the best-possible m-competitive guarantee. If an operation j_i with processing time much larger than p_{j_1} becomes active, we diverge from the robust algorithm and we stop processing j. We then effectively treat j as re-arriving but starting from operation i. We call the operations $j_1, \ldots, j_{i-1}$ a *chunk*.

Since the chunks of a job have increasing predictions, handling those has a strong connection to the special case of instances with monotone non-decreasing operation processing times, for which we show that Operations-SRPT is m-competitive.

▶ **Theorem 2.** *For scheduling with m operations, Operations-SRPT is m-competitive for minimizing the total flow time on a single machine if $p_{j_1} \leq \ldots \leq p_{j_m}$ for all jobs j.*

These two ingredients suggest the $O(m^2)$ bound: we lose a factor m by underestimating the size of a chunk by a factor of at most m as the chunk can consist of that many operations, and another factor m because each job may be split into up to m chunks, treated as m virtual jobs. In fact, by defining m_1 as the maximum number of chunks per job and m_2 as the maximum number of operations per chunk, we can show a refined bound, which implies Theorem 1; we give precise definitions in Section 5. Importantly, m_1 and m_2 depend only on the instance, and not on our algorithm.

▶ **Theorem 3.** *For scheduling with m operations, there exists a deterministic $O(m_1 \cdot m_2)$-competitive algorithm for minimizing the total flow time on a single machine.*

Our analysis uses the strong technique of dual fitting, also used in [24]. However, we need a more intricate dual program to cope with the chunks in our algorithm. Further, our more adaptive algorithm requires several new ideas and very careful adaptations in the analysis.

In terms of the number of operations m, we can only show a (randomized) lower bound of $\Omega(m)$, so Theorem 1 is not tight. However, in terms of m_1 and m_2, Theorem 3 is tight in the following sense. On the one hand, if $m_2 = 1$, so each chunk is composed of one operation, the result by [32] shows that every deterministic algorithm has a competitive ratio of at least m_1 (cf. Theorem 26), and we show that every randomized algorithm has a competitive ratio of at least $\Omega(m_1)$ (cf. Theorem 27). On the other hand, if $m_1 = 1$, the lower bound for robust flow time implies a competitive ratio of $\Omega(m_2)$ [7].

Finally, we look at the case of two operations per job ($m = 2$). This recovers *scheduling with obligatory tests* [20], where a mandatory test determines a job's processing time (the test is the first operation, and the job is the second). Dogeas et al. [20] consider total completion time minimization and show that the best possible deterministic competitive ratio is between $\sqrt{2}$ and 1.861, with an improved upper bound of 1.585 for uniform-length tests ($p_{j_1} = p$). Our Theorem 1 and lower bounds imply that, for total flow time, the best possible deterministic competitive ratio for scheduling with obligatory tests lies between 2 and 672; we show it is 2 for uniform-length tests.

▶ **Theorem 4.** *For scheduling with obligatory uniform-length tests, Operations-SRPT is 2-competitive for minimizing the total flow time on a single machine.*

Our analysis of Operations-SRPT for this special case is inspired by Smith's inductive SRPT analysis [36]. While SRPT always processes the smallest job in the system, the same does not hold for Operations-SRPT on the operation level: If we consider all operations that belong to active jobs (even those operations that are not yet active), then Operations-SRPT

does not necessarily process the smallest such operation, because it might not be active yet. To compensate for this crucial difference to SRPT, we apply a variant of the inductive SRPT analysis only to a subset of operations for which we can guarantee that they are processed in order of their remaining processing times. We then carefully complement this with volume-based arguments to argue about all operations.

Our results for scheduling with obligatory tests are in contrast to the model of scheduling with *optional* tests [21], for which we show in the full version [31] of this paper that no deterministic algorithm has a constant competitive ratio, even for uniform-length tests. The main difference to the model with obligatory tests is that the total processing volume is not equal for all schedules anymore and instead depends on the tests that an algorithm decides to execute, which an adversary can exploit.

1.2 Further Related Work

For non-clairvoyant scheduling on a single machine, Becchetti and Leonardi [11] presented an $O(\log n)$-competitive randomized algorithm, matching the lower bound of [32]; their deterministic lower bound of $\Omega(n^{1/3})$ remained unmatched until today. Bender et al. [14] proposed a model where $\lfloor \log_2 p_j \rfloor$ is revealed to an algorithm when job j arrives, for which Becchetti et al. [12] gave an $O(1)$-competitive algorithm. Building on these results, Azar, Leonardi, and Touitou assumed that a job j arrives with a predicted processing time $\hat{p}_j$ with prediction error $\mu = (\max_j \hat{p}_j/p_j) \cdot (\max_j p_j/\hat{p}_j)$, and presented algorithms with competitive ratios $O(\mu^2)$ [6] and $O(\mu \log \mu)$ [7]. Gupta et al. [24] very recently improved this to $O(\mu)$, which is best-possible [7]. Under $(1+\varepsilon)$-speed augmentation, a simple non-clairvoyant algorithm can be shown to be $O(1/\varepsilon)$-competitive [26].

The weighted objective is in terms of guarantees much harder, even in the clairvoyant setting: Bansal and Chan [9] showed that no online algorithm can be $O(1)$-competitive. The currently best-known algorithms are due to Azar and Touitou [8], building up on earlier works [10, 18]. Under $(1+\varepsilon)$-speed augmentation, there are several $O(1)$-competitive algorithms [10, 13]. In the offline setting, unlike the unweighted problem, it is NP-hard [28], and a PTAS for minimizing the weighted flow time has been established only very recently [4].

Our problem can be considered a special case of scheduling with *online precedence constraints* and *weighted* jobs. In scheduling with precedence constraints, we are given a directed acyclic graph $G = (J, E)$ that uses the jobs as vertices and formulates precedence constraints between the jobs, i.e., a job j can only be processed once all j' with $(j', j) \in E$ have been completed. In the online variant, the job set and precedence constraint graph are initially unknown to the scheduler, and a job is only revealed to the scheduler once all predecessors have been completed. This scheduling model has for example been studied for makespan [5] and for weighted total completion time minimization, where no deterministic algorithm has a competitive ratio better than $\Omega(n)$ but $\mathcal{O}(1)$-competitive algorithms are possible in learning-augmented settings [29]. The model captures our operation flow time scheduling problem: A job j with m operations in our model can be represented by introducing (i) a single job j'_i for each operation j_i of j, (ii) introducing precedence constraints (j'_i, j'_{i+1}) for all $1 \leq i < m$, and (iii) using weights $w_{j'_m} = 1$ and $w_{j'_i} = 0$ for all $i < m$. To the best of our knowledge, online precedence constraints have not yet been studied for the total flow time objective.

As outlined before, our model is related to non-clairvoyant flow time minimization with untrusted predictions [24, 6, 7]. Untrusted predictions have also been studied for the special case of total completion time minimization (see, e.g., [33, 37, 25, 16, 17, 19, 22, 30, 15]).

1.3 Organization

In the next section, we introduce definitions and notation. In Section 3 we introduce Operations-SRPT and prove Theorem 2. In Section 4 we analyze this algorithm for scheduling with obligatory uniform-length tests. Finally, in Section 5 we present our algorithm for the general case and prove Theorems 1 and 3. The lower bounds are deferred to the appendix and all other missing proofs can be found in the full version [31].

2 Notation and Preliminaries

We give a formal definition of our *operation flow time scheduling problem*. There are n jobs, denoted $1, \dots, n$, that arrive online over time. Each job j arrives at an integer release date r_j, has an integer processing time $p_j \geq 1$, and is composed of m operations $j_1, \dots, j_m$, each with an integer operation processing time $p_{j_i} \geq 0$ such that $p_j = \sum_{i=1}^{m} p_{j_i}$. We call j_i a *stage-i* operation of job j. The more general setting where jobs have *at most* m operations can be easily simulated using zero-length operations. During each integer interval $[t, t+1]$, an algorithm can process at most one job for one unit of processing. The *completion time* C_j of job j is the first point in time when it has received a total of p_j processing and its *flow time* is $F_j := C_j - r_j$. The objective is to minimize $\sum_j F_j$.

Let ALG and OPT denote the total flow time of an algorithm and the optimal solution, respectively. An algorithm is *c-competitive* if, for any instance, $\text{ALG} \leq c \cdot \text{OPT}$. We say that a job is ***active*** at time t if $r_j \leq t < C_j$. An algorithm is *locally c-competitive* if at any time t, it holds that $|J(t)| \leq c \cdot |J^*(t)|$, where $J(t)$ denotes the set of active jobs in the algorithm's schedule at time t and $J^*(t)$ in the optimal solution [34]. Since $\sum_j F_j = \sum_{t \geq 0} |J(t)|$, every locally c-competitive algorithm is also c-competitive.

Let $y_j(t)$ denote the cumulative processing of job j until time t. For each job j, we call the operation of j which is next to be processed the ***active operation***; formally, operation j_i is active at time t if $j \in J(t)$ and $\sum_{i'=1}^{i-1} p_{j_{i'}} \leq y_j(t) < \sum_{i'=1}^{i} p_{j_{i'}}$. Since there is exactly one operation active at time t if and only if j is active at time t, we slightly overload notation and use $J(t)$ to denote the set of active operations (if clear from the context). Moreover we say that an operation j_i is an ***alive operation*** if it has not been completed until time t; formally, operation j_i is alive at time t if $j \in J(t)$ and $y_j(t) < \sum_{i'=1}^{i} p_{j_{i'}}$. Note the difference between active and alive operations; in particular, at most one operation j_i of job j is active at a time t, while all operations succeeding j_i (including) are alive. At any time t, the algorithm knows the processing times of all active operations but the processing times of the remaining alive operations remain unknown. We remark that the algorithm also does not know the value of m upfront. Finally, we use $p_j(t) = p_j - y_j(t)$ for the remaining processing time of a job j at time t and $p_{j_i}(t)$ for the remaining processing time of operation j_i at time t. We denote the corresponding quantities in a fixed optimal solution as $p^*_j(t)$ and $p^*_{j_i}(t)$.

3 Operations-SRPT

We start by studying the arguably most natural algorithm for scheduling with m operations, which we call *Operations-SRPT*: we simply run SRPT on the set of active operations.

> **Operations-SRPT:** At any time t, schedule the active operation $q \in J(t)$ with the shortest remaining operation processing time $\min_{q \in J(t)} p_q(t)$.

We break ties in favor of the smaller operation index, i.e., we prefer j_k over j'_ℓ if $k < \ell$; if $k = \ell$ we prefer j if $j < j'$. We show that Operations-SRPT is m-competitive for *monotone non-decreasing* operation processing times, which is best-possible (cf. Theorem 26).

▶ **Theorem 2.** *For scheduling with m operations, Operations-SRPT is m-competitive for minimizing the total flow time on a single machine if $p_{j_1} \leq \ldots \leq p_{j_m}$ for all jobs j.*

Proof. We show that Operations-SRPT is locally m-competitive. Fix a time t. We apply that SRPT is locally 1-competitive [34] to a *virtual operations instance* J_o: all operations of job j are released and active at time r_j and can be processed in any order. Let $J_o(t)$ and $J_o^*(t)$ denote the set of active operations at time t in SRPT's schedule and in an optimal schedule for J_o, respectively. Consider the schedule of SRPT for J_o, which treats each operation as an independent job. Since $p_{j_1} \leq \ldots \leq p_{j_m}$ and $j_1, \ldots, j_m$ are available at time r_j for each job j, SRPT would schedule j's operations in order $j_1, \ldots, j_m$. In particular, note that the schedule of SRPT on J_o is equivalent to the schedule of Operations-SRPT on J under the same tie breaking. Using this and that SRPT is locally 1-competitive on J_o, we have $|J(t)| = |J_o(t)| \leq 1 \cdot |J_o^*(t)|$. Finally, since an optimal solution for instance J is also feasible for instance J_o and for every active job in instance J at time t there can be at most m active operations (per job) in instance J_o at time t, we conclude $|J_o^*(t)| \leq m \cdot |J^*(t)|$. Combining both inequalities implies $|J(t)| \leq m \cdot |J^*(t)|$, which concludes the proof. ◀

Moreover, we show in the full version [31] that monotone non-decreasing operation processing times exactly characterize this algorithm in the following sense.

▶ **Theorem 5.** *Operations-SRPT is $\Omega(\log n)$-competitive if $m = 2$ and $p_{j_1} > p_{j_2}$ for all j.*

In the next section, however, we will see that for $m = 2$ Operations-SRPT is constant-competitive under the additional assumption that all stage-1 operations have equal size.

4 Two Operations: Scheduling with Uniform Obligatory Tests

We consider the special case of $m = 2$ and uniform-length stage-1 operations, i.e., $p_{j_1} = p$ for a common integer $p \in \mathbb{N}_+$. This corresponds to scheduling with uniform-length obligatory tests [20]. Our goal is to prove Theorem 4.

▶ **Theorem 4.** *For scheduling with obligatory uniform-length tests, Operations-SRPT is 2-competitive for minimizing the total flow time on a single machine.*

We first classify the jobs based on the size of their stage-2 operation.

▶ **Definition 6** (Job types). *A job $j \in J$ is of* type-A *if $p_{j_2} \geq p$ and of* type-B *if $p_{j_2} < p$.*

This ties into the concept of chunks (cf. Section 1.1): The operations of a type-B job j can be considered a single chunk, since p_{j_1} is a 2-approximation of the total size p_j of the job. For type-A jobs this is not the case, so each operation can be considered an individual chunk.

Next, we state an important property of Operations-SRPT that holds for our special case provided that we break ties in favor of stage-1 operations. We defer the proof to the full version [31].

▶ **Observation 7.** *At any time t, at most one job has a remaining processing time of $< p$.*

Before we move to the competitive analysis, we introduce more notation. If j is a type-γ job, then we also say that j_1 and j_2 are type-γ operations. For a type $\gamma \in \{A, B\}$ and an operation stage $\ell \in \{1, 2\}$, let $Q^*_{\ell\gamma}(t)$ denote the set of alive *stage-ℓ type-γ operations* at time t in an optimal solution. We will also argue about job *volumes*. For a set of operations or jobs H that are alive at point in time t in the optimal solution, let $\mathrm{vol}^*_t(H) = \sum_{q \in H} p^*_q(t)$. Similarly, we define $\mathrm{vol}_t(H) = \sum_{q \in H} p_q(t)$ for a set of operations of jobs H. Moreover, we define $\mathrm{vol}_t = \mathrm{vol}_t(J(t))$ and $\mathrm{vol}^*_t = \mathrm{vol}^*_t(J(t))$.

Our goal is to show Operations-SRPT is locally 2-competitive, which implies Theorem 4.

▶ **Lemma 8.** *Let τ be an arbitrary point in time. Then it holds that $|J(\tau)| \leq 2 \cdot |J^*(\tau)|$.*

Our proof of Lemma 8 is inspired by the optimality proof for SRPT by Smith [36]. At time τ, he considers the remaining volume of the (at most) $|J^*(\tau)|$-largest active jobs in the algorithm's schedule, denoted by $\mathrm{vol}_\tau(L(|J^*(\tau)|, t))$, and shows that it is at least the total remaining volume in the optimal solution $\mathrm{vol}^*_\tau(J^*(\tau)) = \mathrm{vol}^*_\tau$. Since $\mathrm{vol}_\tau = \mathrm{vol}^*_\tau$, this implies $\mathrm{vol}^*_\tau = \mathrm{vol}_\tau \geq \mathrm{vol}_\tau(L(|J^*(\tau)|, \tau)) \geq \mathrm{vol}^*_\tau = \mathrm{vol}_\tau$, meaning that the $|J^*(\tau)|$-largest active jobs at time τ in the algorithm's schedule contain all of the algorithm's remaining volume vol_τ. This can only be if $|J(\tau)| \leq |J^*(\tau)|$.

Inspired by this, we (i) prove a similar volume invariant as in [36] but only for *stage*-2 *type-A operations* and (ii) show that this weaker volume invariant implies Lemma 8. To this end, let $L_{2A}(x, t)$ denote the set of the x-largest stage-2 type-A operations alive at time t in Operations-SRPT's schedule and let $\mathrm{vol}_\tau(L_{2A}(x, t))$ denote the total remaining volume of $L_{2A}(x, t)$ at time t. We would like to prove the following inequality, which states that the total remaining volume of the $|J^*_{2A}(t)|$-largest stage-2 type-A operations alive at τ in Operations-SRPT's schedule is at least as large as the remaining volume of all stage-2 type-A operations alive at time τ in the optimal solution:

$$\mathrm{vol}_\tau(L_{2A}(|Q^*_{2A}(\tau)|, \tau)) \geq \mathrm{vol}^*_\tau(Q^*_{2A}(\tau)) \ . \tag{1}$$

In the next lemma, we show that this volume invariant indeed implies Lemma 8. To illustrate the proof idea, assume for now that the optimal solution does not have alive stage-1 type-A operations at time τ. Using $\mathrm{vol}_\tau = \mathrm{vol}^*_\tau$, Inequality (1) implies $R := \mathrm{vol}_\tau - \mathrm{vol}_\tau(L_{2A}(|Q^*_{2A}(\tau)|, \tau)) \leq \mathrm{vol}^*_\tau - \mathrm{vol}^*_\tau(Q_{2A}(\tau)) =: R^*$, i.e., when ignoring the $|Q^*_{2A}(\tau)|$-largest stage-2 type-A operations in both schedules, the remaining volume of the algorithm at time τ is at most as large as the remaining volume in the optimal solution. Since we assume $Q^*_{1A}(\tau) = \emptyset$, the optimal solution needs at least $R^*/(2p)$ alive type-B jobs at time τ to fill the volume R^*, as all those jobs have size at most $2p$. However, the algorithm has at most one job with a remaining volume smaller than p (cf. Observation 7). Hence, the algorithm can fit at most $R/p \leq R^*/p$ jobs into the volume R, which implies $|J(\tau)| \leq 2 \cdot |J^*(\tau)|$. the full version [31], we turn this idea into a full proof that takes stage-1 type-A operations into account.

▶ **Lemma 9.** *If $\mathrm{vol}_\tau(L_{2A}(|Q^*_{2A}(\tau)|, \tau)) \geq \mathrm{vol}^*_\tau(Q^*_{2A}(\tau))$, then $|J(\tau)| \leq 2 \cdot |J^*(\tau)|$.*

Given Lemma 9, it only remains to prove that the volume invariant (1) holds. A common approach (see e.g. [36, 10, 6]) for this is to prove that

$$\mathrm{vol}_t(L_{2A}(|Q^*_{2A}(t) \cap Q^*_{2A}(\tau)|, t)) \geq \mathrm{vol}^*_t(Q^*_{2A}(t) \cap Q^*_{2A}(\tau)) \tag{2}$$

holds at any time $0 \leq t \leq \tau$ via induction, which then directly implies (1) for $t = \tau$.

A main difference between our volume invariant and previous approaches is that our invariant is not based on jobs but on *subsets* of operations, which leads to the following technical but crucial difference: Even if there are at most $|Q^*_{2A}(t) \cap Q^*_{2A}(\tau)|$ active operations at time t in Operations-SRPT's schedule, we can have that $\mathrm{vol}_t(L_{2A}(|Q^*_{2A}(t) \cap Q^*_{2A}(\tau)|, t))$ does *not* cover the total remaining volume at time t as there might be active stage-1 or type-B operations. On the technical level, this is a crucial difference to previous approaches that consider volume invariants based on *all* jobs. We address this difference in the full version [31] by using a carefully chosen time t' for the base case of our inductive proof.

5 Algorithm for General Instances

In this section, we give our results for general instances and prove Theorems 1 and 3. If the operation sizes are non-decreasing, then Operations-SRPT is $O(m)$-competitive (cf. Theorem 2). If p_{j_1} is a good approximation of p_j, specifically if $p_{j_1} \geq \frac{1}{m} p_j$, then we can treat p_{j_1} as a prediction and are $O(m)$-competitive [6, 24]. But what if neither applies?

Our solution carefully combines these two approaches: we first find the largest index i such that $\lfloor \log_2 p_{j_\ell} \rfloor \leq \lfloor \log_2 p_{j_1} \rfloor$ for all $\ell \in \{1, \dots, i\}$. We call the resulting set of consecutive operations $\{j_1, \dots, j_i\}$ a *chunk*. We treat this chunk as a virtual job (as in Operations-SRPT) and essentially run the algorithm of [24] on it. When the chunk is completed, we compute the next chunk of the job and repeat. We formalize these ideas using the concept of *classes*.

▶ **Definition 10** (Class of an operation). *The class of operation q is defined as $k_q := \lfloor \log_2 p_q \rfloor$.*

5.1 The Algorithm

Our algorithm maintains the following objects.

- We maintain a *current class* class_j for every job. When a job arrives, we set $\mathsf{class}_j \leftarrow k_{j_1}$.
- We maintain a queue J^{full} of jobs, which is sorted by class_j. Whenever a job arrives, we insert it into J^{full}. Let front be the job with the smallest current class in J^{full}; break ties first in favor of the job with the smallest active operation, and remaining ties arbitrarily.
- We maintain a stack J^{part} of jobs. Let top be the job on top of J^{part}.

At any integer time t, we execute the following steps.

1. While $|J^{\text{full}}| \geq |J(t)|/4$ *and* the front has a strictly smaller current class than top, remove front from J^{full} and push it to J^{part}.
2. Process $j = \mathsf{top}$ during $[t, t+1]$.
3. If j completed, remove it from J^{part}. Otherwise, let q be the active operation of j at time $t+1$. If $k_q \geq \mathsf{class}_j + 1$, remove j from J^{part}, update $\mathsf{class}_j \leftarrow k_q$, and insert j into J^{full}.

Note that in Step 3, j will only be moved back to J^{full} if q just became active at time $t+1$, because otherwise the condition can never be true. For the analysis, we use $J^{\text{full}}(t)$, $J^{\text{part}}(t)$, $\mathsf{top}(t)$, $\mathsf{front}(t)$, and $\mathsf{class}_j(t)$ to denote the state of the algorithm at time t. Also, we write $k(t) := \mathsf{class}_{\mathsf{top}(t)}(t)$ for the current class of the job that is processed during $[t, t+1]$.

5.2 Chunks

We first analyze the structure of when the algorithm moves jobs between J^{full} and J^{part}. This is exactly the structure of chunks of a job, which we formally define as follows.

▶ **Definition 11** (Chunks). *Consider a job j with m operations $j_1, \dots, j_m$. We partition the operations of j into a sequence of* chunks *$c_1, \dots, c_{d_j}$ as follows:*

- *c_1 is the maximal prefix of operations such that all operations $j_i \in c_1$ are of class $\leq k$, where k is the class of j_1.*
- *For $h > 1$, the chunk c_h is the maximal prefix of the operations $\{j_1, \dots, j_m\} \setminus \bigcup_{h'<h} c_{h'}$ s.t. all operations $j_i \in c_h$ are of class $\leq k$, where k is the class of the first operation in c_h.*

We use $p_c = \sum_{q \in c} p_q$ to refer to the processing time of c. The class *k_c of chunk c is the maximum class of an operation in c, which is the class of the first operation in c.*

See Figure 2 for an illustration of a job decomposed into chunks. The definition of chunks directly gives that the classes of the chunks of a job j are strictly increasing.

▶ **Fact 12.** *Let $c_1, \dots, c_{d_j}$ be the chunks of a job j. Then, $k_{c_1} < \dots < k_{c_{d_j}}$.*

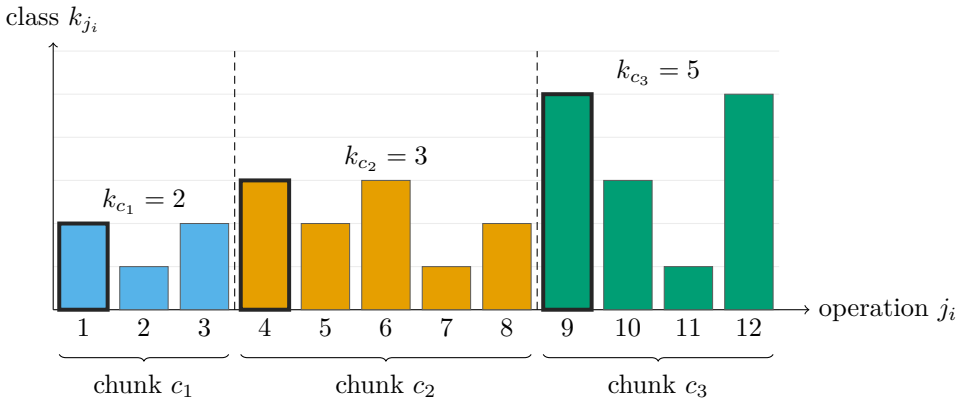

Figure 2 Chunk structure of a job with 12 operations and 3 chunks. Each bar is an operation.

In the schedule of the algorithm, let $c_j(t)$ denote the chunk that contains the active operation $o_j(t)$ of an active job j at time t. We say that this is the *active chunk* at time t. Let r_c denote the first time when chunk c is active, i.e., the time it becomes active. A chunk c is *alive* at time t if its job $j(c)$ is active at t. Similar to operations, each active chunk is alive but not every alive chunk is active. We observe that the description of the algorithm ensures that chunks are exactly the entities of consecutive operations that the algorithm works on before moving a job back to J^{full}. The following observation follows from the definition of the algorithm, we defer the proof to the full version [31].

▶ **Observation 13.** *Consider a job j with ℓ chunks $c_1, \ldots, c_\ell$. The algorithm inserts j into J^{full} at time t if and only if $t \in \{r_{c_1}, \ldots, r_{c_\ell}\}$. Define $r_{c_\ell+1} = C_j$ for the completion time C_j of job j. Each interval $I_i = [r_{c_i}, r_{c_{i+1}}]$ with $i \in \{1, \ldots, \ell\}$ satisfies the following properties:*
1. *Job j is moved to J^{part} exactly once during I_i.*
2. $\mathsf{class}_j(t) = k_{c_i}$ *for all* $t \in [r_{c_i}, r_{c_{i+1}})$.

Since for every job $j \in J(t)$ there exists exactly one active chunk $c_j(t)$, we will without loss of generality refer to $J(t)$ as the set of active chunks at time t. Similarly, we use $J^{\text{full}}(t)$ and $J^{\text{part}}(t)$ for the set of active chunks of jobs in $J^{\text{full}}(t)$ and $J^{\text{part}}(t)$, respectively.

5.3 Instance Parameters m_1 and m_2

Based on the chunk structure of an instance, we derive two parameters m_1 and m_2 that describe the maximal number of chunks in a job and the maximal chunk size, respectively. Our analysis and performance guarantees will be based on these instance parameters.

▶ **Definition 14** (m_1 and m_2)**.** *We define $m_1 := \max_{j \in J} d_j$ as the maximum number of chunks belonging to a single job, where d_j is the number of chunks of job j, and we define $m_2 := \max_{c \in C} |c|$ as the size (number of operations) of the largest chunk.*

The class of a chunk c together with its length/size gives us an approximation of its processing time p_c. Hence, the class of a chunk can be used to approximate its size.

▶ **Observation 15.** *Let c be a chunk of class k_c of some job j, then $p_c = \sum_{q \in c} p_q \leq m_2 \cdot 2^{k_c+1} \leq 2 \cdot m_2 \cdot p_{q_1}$ where q_1 is the first operation in c.*

Proof. Since q_1 is of class k_c, we have $p_{q_1} \geq 2^{k_c}$. All $q \in c \setminus \{q_1\}$ are of a class $\leq k_c$ by Definition 11. Hence, $p_q \leq 2^{k_c+1} \leq 2 \cdot p_{q_1}$ for all $q \in c \setminus \{q_1\}$. In conclusion, $\sum_{q \in c} p_q \leq \min\{m_2 \cdot 2^{k_c+1}, 2 \cdot m_2 \cdot p_{q_1}\}$. ◀

5.4 Local Competitive Analysis and the Local Chunk LP

Fix a time τ. Our goal is to prove $|J(\tau)| \leq O(m_1 \cdot m_2) \cdot |J^*(\tau)|$, which implies local competitiveness. Similarly to our analysis of Operations-SRPT in Section 3, we will show this bound indirectly by comparing $|J(\tau)|$ against the number of chunks that are alive in the optimal solution, denoted by $|J_c^*(\tau)|$. Since each job in $J^*(\tau)$ is composed of at most m_1 chunks, we conclude that

$$|J_c^*(\tau)| \leq m_1 \cdot |J^*(\tau)|. \tag{3}$$

Our goal is to prove the following lemma, which will together with (3) imply Theorem 3.

▶ **Lemma 16.** *It holds that* $|J(\tau)| \leq O(m_2) \cdot |J_c^*(\tau)|$.

Our proof relies on a linear programming relaxation for the problem of minimizing the number of alive chunks at time τ. It follows the same structure as the LP in [24] for minimizing the number of alive jobs at time τ. Let C denote the set of all chunks. We denote with $j(c)$ the job to which chunk c belongs. For a set $S \subseteq C$ of chunks, define the excess at time τ as $e(S) := \max(0, p(S) - (\tau - \ell_S))$ where $\ell_S = \min_{c \in S} r_{j(c)}$ and $p(S) = \sum_{c \in S} p_c$. For each chunk c, let x_c be a variable indicating whether chunk c is alive at time τ. While these quantities describe the state at time τ, we do not add τ to the notation to keep it light. The local chunk LP for time τ can be written as follows.

$$\begin{aligned}
\text{(Chunk-LP}(\tau)\text{)} \quad \min \quad & \sum_{c \in C : \tau \geq r_{j(c)}} x_c & \\
\text{s.t.} \quad & \sum_{c \in S} \min(p_c, e(S)) \cdot x_c \geq e(S) & \forall S \subseteq C \\
& x_c \geq 0 & \forall c \in C
\end{aligned}$$

Intuitively, the LP operates on a surrogate instance J_c that treats each chunk $c \in C$ as an individual job with processing time p_c and release date $r_{j(c)}$. That is, all chunks c that belong to job $j(c)$ are released at the same time and can be processed in any order. For the objective of minimizing the number of alive chunks at time τ, this is a relaxation; the idea is similar to the analysis of Operations-SRPT in Section 3. It is easy to verify that the covering constraints of the LP are valid constraints for the problem of minimizing the number of alive jobs at time τ, which implies the following lemma, whose proof is deferred to the full version [31].

▶ **Lemma 17.** *The optimal objective value of (Chunk-LP*(τ)*) is at most* $|J_c^*(\tau)|$.

In order to compare $|J(\tau)|$ to the optimal objective value of (Chunk-LP(τ)), we use a dual fitting analysis. The dual of (Chunk-LP(τ)) with variables y_S can be written as follows:

$$\begin{aligned}
\text{(Chunk-DP}(\tau)\text{)} \quad \max \quad & \sum_{S} e(S) \cdot y_S & \\
\text{s.t.} \quad & \sum_{S : c \in S} \min(p_c, e(S)) \cdot y_S \leq 1 & \forall c \in C \colon r_{j(c)} \leq \tau \\
& y_S \geq 0 & \forall S \subseteq C
\end{aligned}$$

5.5 Dual Fitting

To prove Lemma 16 via dual fitting, we construct a feasible solution to (Chunk-DP(τ)) with an objective value of $\Omega(|J(\tau)|/m_2)$. A crucial step is identifying subsets $S \subseteq C$ of chunks with a large excess $e(S)$, as the corresponding dual variable y_S may have a large contribution to the objective. In Section 5.5.1, we identify such sets and lower bound their excess. Then, we construct a dual solution, with objective value $\Omega(|J(\tau)|/m_2)$, and prove its feasibility.

5.5.1 Excess Lemmas

A natural strategy, which is also used in [24], for proving excess bounds on a *job level*, is to identify a set of jobs J' and an interval $I = [\ell, \tau]$ such that (i) all jobs in J' are released during I, (ii) the algorithm only works on jobs in J' during I, and (iii) the set J' has a significant remaining volume $p_{J'}(\tau) := \sum_{j \in J'} p_j(\tau)$. Such J' and I then imply for the set of *jobs* J' an excess lower bound $e(J') := p(J') - (\tau - \ell) \geq p_{J'}(\tau)$.

To prove excess bounds for *sets of chunks* S, we want to follow this strategy but are facing a new challenge: The algorithm treats each chunk $c \in S$ as its own job with release date r_c. However, to bound the excess $e(S)$, we need an interval I such that all *jobs* $j(c)$ with $c \in S$ are released during I. Hence, we have to handle the disconnect between r_c and $r_{j(c)}$.

In the following, we provide proofs for excess bounds that take this challenge into account. To this end, let $F(k) \subseteq J(\tau)$ denote the set of full chunks of class k active at time τ. Call class k *crucial* if $F(k) \neq \emptyset$.

▶ **Definition 18** ($t_{>k}$, $t_{\geq k}$, $S_{\leq k}$ and $S_{<k}$). *Let $t_{>k}$ (respectively $t_{\geq k}$) be the last time before τ when the algorithm processed a chunk of class $> k$ (resp. $\geq k$). Let $S_{\leq k}$ (resp. $S_{<k}$) denote the set of chunks c with the following properties.*

1. *c is of a class $\leq k$ (resp. $< k$), and*
2. *$j(c)$, the job to which c belongs, is released during $[t_{>k} + 1, \tau]$ (resp. $[t_{\geq k} + 1, \tau]$).*

To prove excess bounds for the sets $S_{<k}$ and $S_{\leq k}$, we rely on the following lemma, which formulates a crucial property of the algorithm regarding the chunk c that is processed at a time t. In particular, such a chunk has the smallest class among all active chunks with possibly a single exception. We defer the proof to the full version [31].

▶ **Lemma 19.** *Let c be the chunk of class $k(t)$ processed during $[t, t+1]$.*

1. *If there exists a chunk c' of class $< k(t)$ in $J(t)$, then there cannot be another chunk of class $\leq k(t)$ in $J(t) \setminus \{c, c'\}$.*
2. *If there exists another chunk $c' \neq c$ of class $k(t)$ in $J(t)$, then there cannot be any chunk of class $< k(t)$ in $J(t)$.*

▶ **Lemma 20** (Excess Lemma 1). *If class k is crucial, $e(S_{\leq k}) \geq \sum_{k' \leq k} \sum_{c \in F(k')} p_c - 2m_2 2^{k+1}$.*

Proof. We first establish that at time $t_{>k}$ there can be at most one active chunk of class $\leq k$. Let c be such a chunk belonging to a job $j = j(c)$ (if it exists; otherwise the claim is trivial). By definition of $t_{>k}$, the algorithm processes a chunk c' of class $> k$ during $[t_{>k}, t_{>k} + 1]$. This immediately implies that c is full at $t_{>k}$. By Lemma 19 and the existence of c, there cannot exist any additional active chunk of class $< k$ at time $t_{>k}$. Hence, c is the only (full or partial) active chunk of class $\leq k$ at time $t_{>k}$.

We next claim that all chunks of class $\leq k$ that do not belong to job j and that became active during $[t_{>k} + 1, \tau]$ must belong to jobs released after $t_{>k}$. To see this, let c' be a chunk of class $\leq k$ that became active during $[t_{>k} + 1, \tau]$ that belongs to a job j' released

before $t_{>k}+1$. Thus, there must exist an earlier chunk c'' of j' that is active at time $t_{>k}$. By Fact 12, c'' must also be of class $\leq k$. This is a contradiction, since c is the only such chunk at time $t_{>k}$. Thus, except chunks of job j, all chunks c of class $\leq k$ that became active during $[t_{>k}+1,\tau]$ belong to jobs $j(c)$ that are released during $[t_{>k}+1,\tau]$. By definition, all of these chunks belong to $S_{\leq k}$.

We have established that there can be at most one chunk c of class $\leq k$ that is full at time $t_{>k}$, which is not part of $S_{\leq k}$. Let $j=j(c)$ denote the job of this chunk. By choice of $t_{>k}$, the algorithm only works on chunks in $S_{\leq k}$ and on chunks belonging to j during $[t_{>k}+1,\tau]$. Let Δ denote the amount of time during $[t_{>k}+1,\tau]$ which the algorithm spends working on j. Then, by definition of excess,

$$e(S_{\leq k})+\Delta \geq \sum_{k'\leq k}\sum_{c\in F(k')} p_c \ .$$

It remains to bound Δ. Note that whenever the algorithm works on job j during $[t_{>k}+1,\tau]$, then it either works on the chunk c of class $\leq k$ that was full at time $t_{>k}$ or on later chunks that become active due to the completion of c. Let $c_1 \leq \ldots \leq c_d$ denote these chunks indexed in the order they are processed. Since all of these chunks are processed during $[t_{>k}+1,\tau]$, they have to be of class $\leq k$. Furthermore, by Fact 12, the classes of these chunks are strictly increasing. Since each c_i is composed of at most m_2 operations, the total volume of c_i is at most $m_2\cdot 2^{k_{c_i}+1}$. We bound Δ by summing over the volume of all these c_i's:

$$\Delta \leq \sum_{i=1}^{d} m_2\cdot 2^{k_{c_i}+1} \leq m_2\cdot\sum_{i=0}^{k} 2^{i+1} \leq 2\cdot m_2\cdot 2^{k+1}.$$

We can conclude with the excess bound

$$e(S_{\leq k})+2\cdot m_2\cdot 2^{k+1} \geq \sum_{k'\leq k}\sum_{c\in F(k')} p_c \ . \qquad \blacktriangleleft$$

In the full version [31], we prove a second excess lemma for the sets $S_{<k}$ by using similar ideas.

▶ **Lemma 21** (Excess Lemma 2). *If class k is crucial, then $e(S_{<k}) \geq \sum_{k'<k}\sum_{c\in F(k')} p_c$.*

5.5.2 Construction and Analysis of the Dual Solution

We construct our dual solution based on the sets $S_{<k}$ and $S_{\leq k}$ as defined in the previous section. For every crucial class k, we define

- $y_{S_{<k}} := \frac{|F(k)|}{3m_2\cdot e(S_{<k})}$ if $|F(k)|<6m_2$,
- $y_{S_{\leq k}} := \frac{1}{m_2 2^k}$ if $|F(k)|\geq 6m_2$,

and for all other sets S we define $y_S := 0$.

5.5.2.1 Dual Objective Value

Next, we lower bound the objective value of the dual solution in terms of $\frac{J(\tau)}{m_2}$.

▶ **Lemma 22.** *We have $\sum_S e(S)y_S \geq \frac{|J(\tau)|}{12m_2}-\frac{1}{3m_2}$.*

Proof. For every class k with $|F(k)|<6m_2$ we get a contribution equal to

$$e(S_{<k})y_{S_{<k}} = e(S_{<k})\cdot\frac{|F(k)|}{3m_2\cdot e(S_{<k})} = \frac{1}{3m_2}|F(k)|.$$

For every class k with $|F(k)| \geq 6m_2$, we have

$$e(S_{\leq k})y_{S_{\leq k}} \geq \left(\sum_{k' \leq k}\sum_{c \in F(k')} p_c - 2m_2 2^{k+1}\right)\frac{1}{m_2 2^k} \geq \left(\sum_{c \in F(k)} p_c - 2m_2 2^{k+1}\right)\frac{1}{m_2 2^k}$$
$$\geq \frac{2^k}{m_2 2^k}|F(k)| - 4 \geq \frac{1}{m_2}|F(k)| - \frac{4}{6m_2}|F(k)| = \frac{1}{3m_2}|F(k)|.$$

Here, the first inequality uses Lemma 20. Summing over all crucial classes, we obtain $\sum_S e(S)y_S \geq \frac{|J^{\text{full}}(\tau)|}{3m_2}$. The final inequality uses that $|J^{\text{full}}(\tau)| \geq \frac{|J(\tau)|}{4} - 1$, which is a property of the algorithm (cf. the full version [31] or [24] for a proof). ◀

5.5.2.2 Dual Feasibility

Finally, we argue about the feasibility of the dual solution. Fix a chunk c^* of class k^* with $r_{j(c^*)} \leq \tau$. We show that our dual variables violate the constraint of c^* only by some constant factor, which implies that scaling our variables by that factor yields a feasible dual solution.

By definition of our dual variables, the only sets S that can have $c^* \in S$ and $y_S > 0$ are sets $S_{\leq k}$ and $S_{<k}$ with $k \geq k^*$. Let $K_0 = \{k \geq k^* \mid c^* \in S_{<k}\}$ and $K_1 = \{k \geq k^* \mid c^* \in S_{\leq k}\}$.

First, we analyze the contribution of sets $S_{\leq k}$ with $k \in K_1$:

▶ **Lemma 23.** $\sum_{k \in K_1} \min(p_{c^*}, e(S_{\leq k})) \cdot y_{S_{\leq k}} < 4$.

Proof. For every $k \in K_1$, we have $y_{S_{\leq k}} = \frac{1}{m_2 2^k}$ if $|F(k)| \geq 6m_2$ and $y_{S_{\leq k}} = 0$ otherwise. Since c^* is of class k^*, we have $p_{c^*} \leq m_2 2^{k^*+1}$. Thus,

$$\sum_{k \in K_1} \min(p_{c^*}, e(S_{\leq k})) \cdot y_{S_{\leq k}} \leq \sum_{k \geq k^*} p_{c^*} \cdot \frac{1}{m_2 2^k} \leq \sum_{k \geq k^*} \frac{m_2 2^{k^*+1}}{m_2 2^k} < 4\ .$$ ◀

Next, we analyze the contribution of sets $S_{<k}$ with $k \in K_0$.

▶ **Lemma 24.** $\sum_{k \in K_0} \min(p_{c^*}, e(S_{<k})) \cdot y_{S_{<k}} \leq 10$.

Proof. To this end, for all $0 \leq \ell \leq \Delta := \lfloor \log_2(3m_2) \rfloor$, let $T_\ell = \{k \in K_0 \mid 2^\ell \leq |F(k)| < 2^{\ell+1}\}$. For each $k \in T_\ell$, by definition $y_{S_{<k}} = \frac{|F(k)|}{3m_2 \cdot e(S_{<k})}$ if $|F(k)| < 6m_2$ and $y_{S_{<k}} = 0$ otherwise. Thus,

$$\min(e(S_{<k}), p_{c^*}) \cdot y_{S_{<k}} = \min(e(S_{<k}), p_{c^*}) \cdot \frac{|F(k)|}{3m_2 \cdot e(S_{<k})}$$
$$\leq \min(e(S_{<k}), p_{c^*}) \cdot \frac{2^{\ell+1}}{3m_2 \cdot e(S_{<k})} = \frac{2^{\ell+1}}{3m_2} \min\left(1, \frac{p_{c^*}}{e(S_{<k})}\right)\ . \quad (4)$$

Now, assume that $T_\ell \neq \emptyset$ and let $k_1 < k_2 < \ldots < k_{|T_\ell|}$ denote the classes of T_ℓ. All those classes are crucial. Hence, we can apply Lemma 21 to all of the corresponding sets $S_{<k_i}$. For each $i \geq 2$, this implies:

$$e(S_{<k_i}) \geq \sum_{k' < k_i}\sum_{c \in F(k')} p_c \geq \sum_{c \in F(k_{i-1})} p_c \geq |F(k_{i-1})| \cdot 2^{k_{i-1}} \geq 2^\ell 2^{k_{i-1}}\ . \quad (5)$$

Hence,

$$\sum_{k \in T_\ell} \min(e(S_{<k}), p_{c^*}) y_{S_{<k}} \leq \frac{2^{\ell+1}}{3m_2} \sum_{k \in T_\ell} \min\left(1, \frac{p_{c^*}}{e(S_{<k})}\right)$$
$$\leq \frac{2^{\ell+1}}{3m_2}\left(1 + \frac{m_2 2^{k^*+1}}{2^\ell} \sum_{i \geq 2} \frac{1}{2^{k_{i-1}}}\right) \leq \frac{2^{\ell+1}}{3m_2}\left(1 + \frac{m_2 2^{k^*+1}}{2^\ell} \frac{2}{2^{k_1}}\right)\ ,$$

where the first inequality uses (4), the second inequality uses (5) and that $\min(1, p_{c^*}/e(S_{<k_1})) \leq 1$, the third inequality uses that $p_{c^*} \leq m_2 2^{k^*+1}$, and the fourth inequality uses $\sum_{i\geq 2} \frac{1}{2^{k_i-1}} \leq 2 \cdot 2^{-k_1}$.

For each ℓ with $T_\ell \neq \emptyset$, let $k(\ell)$ denote its smallest class. Since $y_{S_{<k}} = 0$ for all k with $|F(k)| \geq 6m_2$, we can finally sum over all indices ℓ with $T_\ell \neq \emptyset$ to verify the dual constraint

$$\begin{aligned}\sum_{k\in K_0} \min(e(S_{<k}), p_{c^*}) \cdot y_{S_{<k}} &= \sum_{\ell=0}^{\Delta} \sum_{k\in T_\ell} \min(e(S_{<k}), p_{c^*}) \cdot y_{S_{<k}} \\ &\leq \sum_{\substack{\ell=0 \\ T_\ell\neq\emptyset}}^{\Delta} \frac{2^{\ell+1}}{3m_2}\left(1 + \frac{m_2 2^{k^*+1}}{2^\ell}\frac{2}{2^{k(\ell)}}\right) \\ &= \sum_{\substack{\ell=0 \\ T_\ell\neq\emptyset}}^{\Delta} \frac{2^{\ell+1}}{3m_2} + \sum_{\substack{\ell=0 \\ T_\ell\neq\emptyset}}^{\Delta} \frac{2\cdot 2^{k^*+1}}{3}\frac{2}{2^{k(\ell)}} \ .\end{aligned}$$

For the first sum, we have $\sum_{\ell=0}^{\Delta} \frac{2^{\ell+1}}{3m_2} \leq \frac{4}{3m_2} 2^{\log_2(3m_2)} = 4$. For the second sum, since $T_0, \ldots, T_\Delta$ partition all crucial classes $\geq k^*$, all $k(0), \ldots, k(\Delta)$ are pairwise distinct and at least k^*. Hence, we get $\sum_{\ell=0}^{\Delta} 2^{-k(\ell)} \leq 2/2^{k^*}$. Thus, the second sum is at most $16/3$. Since $4 + 16/3 < 10$, this completes the proof. ◀

5.5.2.3 Putting Everything Together

We can finally prove Theorem 3, which then implies Theorem 1.

Proof of Theorem 3. Combining Lemma 23 and Lemma 24, we get

$$\sum_{S: c^*\in S} \min(p_{c^*}, e(S)) \cdot y_S \leq 14 \ .$$

Thus, the dual assignment $y_S/14$ for all $S \subseteq C$ is a feasible solution to (Chunk-DP(τ)). By Lemma 22, this dual solution has an objective value of at least $\frac{|J(\tau)|}{168m_2} - \frac{1}{42m_2}$. Hence, Lemma 17 and weak duality give

$$|J(\tau)| \leq 168m_2 \cdot |J_c^*(\tau)| + \frac{1}{42m_2} \ ,$$

where $|J_c^*(\tau)|$ is the number of active chunks in J^* at time τ. This concludes the proof of Lemma 16 and, using Inequality (3), implies $|J(t)| \in \mathcal{O}(m_1 \cdot m_2) \cdot |J^*(t)|$ for any time t. Integrating over time then implies Theorem 3. ◀

6 Conclusion

In this paper, we introduce the *operations flow time scheduling* problem. We present an $O(m^2)$-competitive algorithm as well as a randomized lower bound of $\Omega(m)$. The more fine-grained representation of our algorithms' competitive ratio as $O(m_1 \cdot m_2)$ is asymptotically tight if either $m_1 = 1$ or $m_2 = 1$. It remains an open question whether an $O(m)$-competitive algorithm is possible. Furthermore, it would be interesting to see whether the large constants in the analysis can be improved, in particular for the special case with just two operations.

References

1 Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986. URL: https://www.worldcat.org/oclc/12285707.

2 Frances E. Allen. Control flow analysis. In *Symposium on Compiler Optimization*, pages 1–19. ACM, 1970. doi:10.1145/800028.808479.

3 Frances E. Allen and John Cocke. A program data flow analysis procedure. *Commun. ACM*, 19(3):137–147, 1976. doi:10.1145/360018.360025.

4 Alexander Armbruster, Lars Rohwedder, and Andreas Wiese. A PTAS for minimizing weighted flow time on a single machine. In *STOC*, pages 1335–1344. ACM, 2023. doi:10.1145/3564246.3585146.

5 Yossi Azar and Leah Epstein. On-line scheduling with precedence constraints. *Discret. Appl. Math.*, 119(1-2):169–180, 2002. doi:10.1016/S0166-218X(01)00272-4.

6 Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In *STOC*, pages 1070–1080. ACM, 2021. doi:10.1145/3406325.3451023.

7 Yossi Azar, Stefano Leonardi, and Noam Touitou. Distortion-oblivious algorithms for minimizing flow time. In *SODA*, pages 252–274. SIAM, 2022. doi:10.1137/1.9781611977073.13.

8 Yossi Azar and Noam Touitou. Improved online algorithm for weighted flow time. In *FOCS*, pages 427–437. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00048.

9 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *SODA*, pages 1238–1244. SIAM, 2009. doi:10.1137/1.9781611973068.134.

10 Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms*, 3(4):39, 2007. doi:10.1145/1290672.1290676.

11 Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM*, 51(4):517–539, 2004. doi:10.1145/1008731.1008732.

12 Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Semiclairvoyant scheduling. *Theor. Comput. Sci.*, 324(2-3):325–335, 2004. doi:10.1016/j.tcs.2004.05.023.

13 Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006. doi:10.1016/j.jda.2005.12.001.

14 Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Approximation algorithms for average stretch scheduling. *J. Sched.*, 7(3):195–222, 2004. doi:10.1023/B:JOSH.0000019681.52701.8b.

15 Ziyad Benomar, Romain Cosson, Alexander Lindermayr, and Jens Schlöter. Non-clairvoyant scheduling with progress bars. *CoRR*, abs/2509.19662, 2025. doi:10.48550/arXiv.2509.19662.

16 Ziyad Benomar and Vianney Perchet. Advice querying under budget constraint for online algorithms. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *NeurIPS*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/eda830e16044587b5082a853c4f25a90-Abstract-Conference.html.

17 Ziyad Benomar and Vianney Perchet. Non-clairvoyant scheduling with partial predictions. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 3506–3538. PMLR / OpenReview.net, 2024. URL: https://proceedings.mlr.press/v235/benomar24a.html.

18 Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *STOC*, pages 84–93. ACM, 2001. doi:10.1145/380752.380778.

19 Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Algorithms with prediction portfolios. In *NeurIPS*, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/7f9220f90cc85b0da693643add6618e6-Abstract-Conference.html.

20 Konstantinos Dogeas, Thomas Erlebach, and Ya-Chun Liang. Scheduling with obligatory tests. In *ESA*, volume 308 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.48.

21 Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82(12):3630–3675, 2020. doi:10.1007/s00453-020-00742-2.

22 Marek Eliás, Haim Kaplan, Yishay Mansour, and Shay Moran. Learning-augmented algorithms with explicit predictors. In *NeurIPS*, 2024. URL: http://papers.nips.cc/paper_files/paper/2024/hash/b1a72c79ac7512df8d7b573f38143ac4-Abstract-Conference.html.

23 Anupam Gupta, Haim Kaplan, Alexander Lindermayr, Jens Schlöter, and Sorrachai Yingchareonthawornchai. A little clairvoyance is all you need. In *FOCS*. IEEE, 2025. doi:10.1109/FOCS63196.2025.00010.

24 Anupam Gupta, Amit Kumar, Debmalya Panigrahi, and Zhaozi Wang. An optimal online algorithm for robust flow time scheduling. In *SODA*, pages 1214–1238. SIAM, 2026. doi:10.1137/1.9781611978971.47.

25 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. *ACM Trans. Parallel Comput.*, 10(4):19:1–19:26, 2023. doi:10.1145/3593969.

26 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. doi:10.1145/347476.347479.

27 James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976. doi:10.1145/360248.360252.

28 J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984. doi:10.1016/B978-0-12-566780-7.50020-9.

29 Alexandra Anna Lassota, Alexander Lindermayr, Nicole Megow, and Jens Schlöter. Minimalistic predictions to schedule jobs with online precedence constraints. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 18563–18583. PMLR, 2023. URL: https://proceedings.mlr.press/v202/lassota23a.html.

30 Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. *ACM Trans. Parallel Comput.*, 2025. doi:10.1145/3711872.

31 Alexander Lindermayr, Guido Schäfer, Jens Schlöter, and Leen Stougie. Online flow time minimization with gradually revealed jobs. *CoRR*, abs/2602.12716, 2026. doi:10.48550/arXiv.2602.12716.

32 Rajeev Motwani, Steven J. Phillips, and Eric Torng. Non-clairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, 1994. doi:10.1016/0304-3975(94)90151-1.

33 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *NeurIPS*, pages 9684–9693, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/73a427badebe0e32caa2e1fc7530b7f3-Abstract.html.

34 Linus Schrage. Letter to the editor - A proof of the optimality of the shortest remaining processing time discipline. *Oper. Res.*, 16(3):687–690, 1968. doi:10.1287/opre.16.3.687.

35 Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57. ACM Press, 2002. doi:10.1145/605397.605403.

36 Donald R. Smith. Technical note - A new proof of the optimality of the shortest remaining processing time discipline. *Oper. Res.*, 26(1):197–199, 1978. doi:10.1287/OPRE.26.1.197.

37 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *NeurIPS*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/5bd844f11fa520d54fa5edec06ea2507-Abstract.html.

38 Sorrachai Yingchareonthawornchai and Eric Torng. Delayed-clairvoyant scheduling. In *MAPSP*, pages 198–201, 2017. URL: https://mapsp2017.uni-bremen.de/ProceedingsMAPSP2017.pdf#page=208.

A Lower Bounds

In this section, we prove several lower bounds for the operation flow time scheduling problem. For the sake of convenience, we state our lower bound constructions using rational release dates and processing times. Lower bound instances with integer release dates and processing times can then be obtained via scaling.

For a fixed algorithm and a fixed instance, we denote by $\delta(t)$ the number of unfinished jobs in the algorithm's schedule and by $\delta^*(t)$ the number of unfinished jobs at time t in an optimal solution. To prove lower bounds, we use the following well-known technique, see e.g. [7, 32, 23].

▶ **Proposition 25.** *If for every sufficiently large integer N there exists an instance such that $\delta^*(t) = N$ and $\delta(t) \geq \rho \cdot \delta^*(t)$ for some constant ρ at some time t, then the algorithm has a competitive ratio of at least ρ.*

A.1 A Lower Bound for Deterministic Algorithms

We first present a lower bound for deterministic algorithms. In fact, our lower bound follows directly from a lower bound of Motwani, Philipps, and Torng [32]. They show that every deterministic non-clairvoyant algorithm has a competitive ratio of at least P, where P is the ratio between the largest and smallest processing time. The construction works for jobs with operation sizes 0 and 1 where the smallest job has processing time 1 and the largest job has a processing time of m. Since this implication is immediate, we give the proof of [32] in the language of operations.

▶ **Theorem 26.** *For any integer $m \geq 2$, the competitive ratio of any deterministic algorithm for minimizing the total flow time with m operations is at least m, even if all operations have size 0 or 1 and are monotone non-increasing.*

Proof. Let N be a large integer. At time 0, we release a set J of $N(m+1)$ jobs with $p_{j_1} = 1$ for all $j \in J$. Whenever the algorithm completes an operation, the next operation again has size 1. We assume w.l.o.g. by the integrality of the instance that the algorithm only preempts at integer times. Let s be the time when the algorithm completed the $(m-1)$th operation of N jobs. Let J_1 denote the set of those jobs. At this time, we set the processing time of all hidden operations to 0. Let t be the time when the algorithm completed exactly m jobs, which must be after time s. By construction, $\delta(t) = mN$. Since at time s all jobs have the same remaining total length, we can assume without loss of generality that the algorithm completed all m jobs in J_1 by time t. Hence, it worked for $t - Nm$ units on jobs in $J \setminus J_1$ until time t, and at time t, each of the Nm jobs in $J \setminus J_1$ has remaining length 1. Thus, another solution can finish all jobs in $J \setminus J_1$ until time t, and thus, $\delta^*(t) \leq N$ because only the N jobs in J_1 remain. Then, the lemma follows from Proposition 25. ◀

A.2 A Lower Bound for Randomized Algorithms

We next move to randomized algorithms. We show that randomization does not significantly help to improve the competitive ratio over deterministic algorithms. We emphasize that this lower bound is new and does not appear in [32]. Our proof roughly follows the same framework as the randomized lower bound proofs for related flow time problems in [7, 23] but tailors it to scheduling with m operations.

▶ **Theorem 27.** *The competitive ratio of any randomized algorithm against an oblivious adversary for minimizing the total flow time with m operations is at least $\Omega(m)$.*

We later apply Yao's principle, so fix any deterministic algorithm. We consider a distribution of instances where at time 0 we release $n = \lfloor 2^{m/2} \rfloor$ jobs with integer processing times P_j drawn independently from the geometric distribution with mean 2 at time 0. That is, $P_j = p$ with probability 2^{-p} for every integer $p \geq 1$. Note that $\mathbb{E}[P_j] = 2$. For every job j, we define its operation processing times as follows: $P_{j_\ell} = 1$ for all $1 \leq \ell \leq \min\{m, P_j\}$, $P_{j_\ell} = 0$ for all $P_j + 1 \leq \ell < m$, and $P_{j_m} = \max\{P_j - m + 1, 0\}$.

Let $t = \lfloor 2(n - n^{3/4}) \rfloor$. We first upper bound the expected number of alive jobs at time t in the optimal solution.

▶ **Lemma 28.** *It holds that* $\mathbb{E}[\delta^*(t)] \leq \mathcal{O}(\frac{n^{3/4}}{\log n})$.

Proof. First, note that the total processing time P of all jobs is a sum of n independent random variables with mean $2n$ and variance $2n$. Thus, Chebyshev's inequality gives

$$\Pr[P \geq 2n + n^{3/4}] = \Pr[P \geq 2n + n^{1/2} \cdot n^{1/4}] \leq \mathcal{O}\left(\frac{1}{n^{1/2}}\right) .$$

Moreover, let $b = \frac{\log n}{4}$ and let B be the number of jobs with processing time more than b. Note that B is binomially distributed, because it can be written as the sum of n binary random variables $\mathbb{1}[P_j > b]$ with equal success probabilities $\Pr[P_j > b] = 2^{-b} = n^{-1/4}$. Therefore, $\mathbb{E}[B] = n^{-1/4} \cdot n = n^{3/4}$ and $\mathbb{V}[B] = n^{-1/4}(1 - n^{-1/4})n = O(n^{3/4})$. Thus, Chebyshev's inequality gives

$$\Pr[B \leq \tfrac{1}{2}n^{3/4}] = \Pr[B - \mathbb{E}[B] \leq -\tfrac{1}{2}\mathbb{E}[B]] \leq \frac{\mathbb{V}[B]}{(\frac{1}{2}\mathbb{E}[B])^2} \leq \mathcal{O}\left(\frac{1}{n^{3/4}}\right) .$$

Let $\mathcal{E}_1$ denote the event $P < 2n + n^{3/4}$ and let $\mathcal{E}_2$ denote the event $B > \frac{1}{2}n^{3/4}$. By the union bound, we have

$$\Pr[\mathcal{E}_1 \cap \mathcal{E}_2] \geq 1 - \Pr[\bar{\mathcal{E}}_1] - \Pr[\bar{\mathcal{E}}_2] \geq 1 - \mathcal{O}\left(\frac{1}{n^{1/2}}\right) - \mathcal{O}\left(\frac{1}{n^{3/4}}\right) = 1 - \mathcal{O}\left(\frac{1}{n^{1/2}}\right) .$$

Conditioned on $\mathcal{E}_1 \cap \mathcal{E}_2$, we compute the maximum number of jobs of size at least b that an optimum solution cannot complete until time t as follows

$$\frac{1}{b}(P - t) \leq \frac{1}{b}\left(2n + n^{3/4} - 2(n - n^{3/4}) + 1\right) \leq \mathcal{O}\left(\frac{n^{3/4}}{\log n}\right) .$$

Note that for sufficiently large n it holds that $\mathcal{O}(\frac{n^{3/4}}{\log n}) \leq \frac{1}{2}n^{3/4} \leq B$, i.e., there are many such long jobs. In total, the expected number of alive jobs in an optimum solution at time t is at most

$$\mathbb{E}[\delta^*(t)] \leq \left(1 - \mathcal{O}\left(\frac{1}{n^{1/2}}\right)\right) \cdot \mathcal{O}\left(\frac{n^{3/4}}{\log n}\right) + \mathcal{O}\left(\frac{1}{n^{1/2}}\right) \cdot n = \mathcal{O}\left(\frac{n^{3/4}}{\log n}\right) ,$$

which concludes the proof of the lemma. ◀

▶ **Lemma 29.** $\mathbb{E}[\delta(t)] \geq \Omega(n^{3/4})$.

Proof. We call a job j *short* if it has $P_j \leq m$; otherwise *long*. Let $\mathcal{E}_1$ be the event that all n jobs are short. The probability of a job j being long is $\Pr(P_j > m) = 2^{-(m-1)} = 2/n^2$, and thus, a union bound gives $\Pr(\mathcal{E}_1) = 1 - n \cdot (2/n^2) = 1 - 2/n$.

Let $L := n^{3/4}$, and define $\mathcal{E}_2$ to be the event that the algorithm finishes at most $n - \frac{L}{2}$ jobs during the first $t = \lfloor 2(n - L) \rfloor$ timesteps. Thus, under $\mathcal{E}_2$ we have $\delta(t) \geq \frac{L}{2}$.

In the following, we condition on $\mathcal{E}_1$ and assume that all jobs are short. Note that every active short operation has an operation processing time of 1, hence an algorithm cannot distinguish between all jobs under $\mathcal{E}_1$. To analyze $\mathcal{E}_2$ under $\mathcal{E}_1$, consider the complementary event, and suppose the algorithm finishes $k = n - \frac{L}{2}$ jobs in the first t timesteps. Say these completed jobs are numbered $1, \dots, k$, this means that $\sum_{j=1}^{k} P_j \leq t = \lfloor 2(n-L) \rfloor$. Rephrasing, this means that among the first $N := 2(n-L)$ unbiased coin flips there are at least $n - \frac{L}{2}$ heads. However, the expected number of heads is only $\mathbb{E}[X] = N/2 = n - L$, hence we can show that this only happens with small probability. Indeed, by Chebyshev's inequality, we have

$$\Pr\left[X \geq \mathbb{E}[X] + \frac{L}{2}\right] \leq \frac{\mathbb{V}(X)}{(L/2)^2} \leq \frac{4n}{L^2} = o(1) \ ,$$

because $\mathbb{V}(X) = N \cdot \frac{1}{2}(1 - \frac{1}{2}) = N/4 \leq n$ and $L = n^{3/4}$. Thus, we have $\Pr(\mathcal{E}_2 \mid \mathcal{E}_1) = 1 - o(1)$.

Since also $\Pr(\mathcal{E}_1) = 1 - o(1)$, we conclude $\Pr(\mathcal{E}_1 \cap \mathcal{E}_2) = 1 - o(1)$. Then

$$\mathbb{E}[\delta(t)] \geq \frac{L}{2} \cdot \Pr(\mathcal{E}_1 \cap \mathcal{E}_2) \geq \Omega(L) = \Omega(n^{3/4}) \ ,$$

which proves the lemma. ◀

We can now prove Theorem 27.

Proof of Theorem 27. Lemmas 28 and 29 immediately imply

$$\frac{\mathbb{E}[\delta(t)]}{\mathbb{E}[\delta^*(t)]} \geq \Omega(\log n) = \Omega(m).$$

The theorem now follows by applying Proposition 25 to every realization of our distribution and finally using Yao's principle. ◀

A.3 Stronger Lower Bound for Operations-SRPT

In this section, we show that the competitive ratio of Operations-SRPT is not constant, even if each job has two operations.

▶ **Theorem 5.** *Operations-SRPT is $\Omega(\log n)$-competitive if $m = 2$ and $p_{j_1} > p_{j_2}$ for all j.*

The basic idea of our construction follows from the lower bound that is given in [14] for an approximate version of SRPT. This version of SRPT groups the jobs into classes based on their remaining processing time, i.e., job j is in class k at time t if $p_j(t) \in [2^k, 2^{k+1})$, and always processes some job from the smallest class. However, our lower bound has to take into account that Operations-SRPT is precise w.r.t. the remaining processing times of the active operations. This requires a more involved construction compared to [14].

To construct the lower bound instance, we first observe that if there exists a *critical* point in time t at which Operations-SRPT and OPT satisfy certain properties, then there exists a sequence of job releases after time t that strictly *increases* the local competitive ratio of Operations-SRPT on the instance. We define these critical times as follows. For the remaining section, fix a sufficiently small constant $\varepsilon > 0$. We also assume w.l.o.g. that OPT is running SRPT. Let $J(t)$ and $J^*(t)$ denote the sets of active jobs at time t in the schedule of Operations-SRPT and OPT, respectively.

▶ **Definition 30.** *Fix parameters $M \in \mathbb{R}_+$ and $k \in \mathbb{N}_+$. We refer to a time t as (k, M)-critical if the following three conditions hold:*

1. $|J^*(t)| = 1$ *and* $p_j^*(t) = 2M - \varepsilon$ *for the single job* $j \in J^*(t)$.
2. *There is a job* $i \in J(t)$ *such that* $p_{i_1}(t) = 0$ *and* $p_{i_2}(t) = \frac{M}{2^k} - \varepsilon$.
3. *All active operations in* $J(t)$ *except for* i_2 *have remaining processing time at least* $\frac{M}{2^k}$.

The next lemma shows that if Operation-SRPT reaches a critical time in its schedule, then the lower bound instance can enforce another critical point in time with a worse local competitive ratio.

▶ **Lemma 31.** *Let t be a (k, M)-critical time. Then, an adversary can enforce a $(k+1, M)$-critical time $t' = t + 2\frac{M}{2^k} - \varepsilon$ with $|J(t')| = |J(t)| + 1$ by releasing two jobs during $[t, t')$.*

Proof. Let t and t' be as defined in the lemma, and consider the following job releases during $[t, t')$:

1. At time t, release a job a with $p_{a_1} = \frac{M}{2^k} - \frac{\varepsilon}{2}$ and $p_{a_2} = \frac{M}{2^{k+1}} - \frac{\varepsilon}{2}$.
2. At time $t_b = t + \frac{M}{2^k} + \frac{M}{2^{k+1}} - \varepsilon$, release a job b with $p_{b_1} = \frac{M}{2^{k+1}}$ and $p_{b_2} = 0$.

Next, we argue about the behavior of OPT and Operations-SRPT between t and t':

- First consider the time interval $[t, t_b]$: Since t is (k, M)-critical by assumption, there are exactly two jobs that OPT can process at time t: The job j with $p_j^*(t) = 2M - \varepsilon$ that exists because t is (k, M)-critical by assumption and the job a that is released at t. Since a is the shorter job and no further jobs are released during $[t, t_b)$, OPT will process a and complete it by time t_b.
  Operations-SRPT will first work on the stage-2 operation of the job i with $p_{i_2}(t) = \frac{M}{2^k} - \varepsilon$, which exists by the assumption that t is (k, M)-critical. Since no further jobs are released, Operations-SRPT will run the job to completion.
  The completion of i_2 takes $\frac{M}{2^k} - \varepsilon$ time units. For the remaining $\frac{M}{2^{k+1}}$ time units of the interval, Operations-SRPT reduces the remaining processing time of a's stage-1 operation a_1 to $\frac{M}{2^{k+1}} - \frac{\varepsilon}{2}$.
- Next, consider the interval $[t_b, t']$. At the beginning of this interval, job b is released. Since $t' - t_b = \frac{M}{2^{k+1}}$, the optimal solution has enough time to complete b during $[t_b, t']$. Hence, $J^*(t) = J^*(t')$ and $|J^*(t')| = 1$. Since the single job in $J^*(t) \cap J^*(t')$ is not processed during the interval, we still have $p_j^*(t') = 2M - \varepsilon$ by our assumption that t is (k, M)-critical. Hence, t' satisfies the first condition of Definition 30.
  Starting from time t_b, Operations-SRPT continues to process the stage-1 operation of a, and completes it at time $t' - \frac{\varepsilon}{2}$. After a is completed, we have $p_{a_2}(t' - \frac{\varepsilon}{2}) = \frac{M}{2^{k+1}} - \frac{\varepsilon}{2}$, and $p_{b_1}(t' - \frac{\varepsilon}{2}) = \frac{M}{2^{k+1}}$. Hence, in the interval $[t' - \frac{\varepsilon}{2}, t']$, Operations-SRPT works on the stage-2 operation of job a. At time t', we have $p_{b_1}(t') = \frac{M}{2^{k+1}}$ and $p_{a_2}(t') = \frac{M}{2^{k+1}} - \varepsilon$. The existence of job a implies that t' satisfies the second condition of Definition 30.
  In order to show that t' is $(k+1, M)$-critical, it only remains to show that the third condition of Definition 30 is satisfied, i.e., all jobs apart from a have remaining time at least $\frac{M}{2^{k+1}}$ at time t'. For b, we already argued above that the remaining time is exactly $\frac{M}{2^{k+1}}$. Since we have $J(t') = (J(t) \setminus \{i\}) \cup \{a, b\}$, the assumption that t is (k, M)-critical implies that all jobs in $J(t') \setminus \{a, b\}$ have remaining time at least $\frac{M}{2^k}$. Hence, the condition is satisfied.
  We conclude the proof of the lemma by observing that $J(t') = (J(t) \setminus \{i\}) \cup \{a, b\}$ implies $|J(t')| = |J(t)| + 1$. ◀

By repeatedly applying Lemma 31, an adversary can make the local competitive ratio of Operations-SRPT arbitrarily large, provided that Operations-SRPT reaches a first critical time at some point in its schedule. The next lemma shows that there indeed is an instance that forces Operations-SRPT to reach a critical time.

▶ **Lemma 32.** *Fix parameter $M \in \mathbb{R}_+$ and let $t = M$. There exists an instance that releases two jobs at time 0 and forces time t to be $(1, M)$-critical.*

Proof. Consider the following two job releases at time 0:
1. Release a job j with $p_{j_1} = p_{j_2} = M - \frac{\varepsilon}{2}$.
2. Release a job i with $p_{i_1} = M$ and $p_{i_2} = 0$.

During $[0, M]$ the optimal solution will only work on job i and complete it at time M. This implies $J^*(t) = \{j\}$ and $p_j^*(t) = 2M - \varepsilon$. Hence, $t = M$ satisfies the first condition of Definition 30.

Operations-SRPT, on the other hand, first completes the stage-1 operation of j at time $M - \frac{\varepsilon}{2}$. Then, Operations-SRPT uses the remaining $\frac{\varepsilon}{2}$ time units to work on the stage-2 operation of job j and reduces its remaining time to $M - \varepsilon$. The existence of j implies that t satisfies the second condition of Definition 30. Finally, the fact that $p_{i_1}(t) = M$ implies that the third condition is satisfied as well. Hence, t is $(1, M)$-critical. ◀

Having Lemmas 31 and 32 in place, we are ready to complete our lower bound construction and prove Theorem 5.

Proof of Theorem 5. Fix a large $k^* \in \mathbb{N}$, define $M = 2^{k^*}$, and consider the following instance:
1. At time 0, release two jobs as described in the proof of Lemma 32.
2. For each $k \in \{2, \dots, k^*\}$:
   a. Consider time $t = M + \left(\sum_{k'=2}^{k-1} 2 \cdot \frac{M}{2^{k'}} - \varepsilon\right)$ and $t' = t + 2\frac{M}{2^k} - \varepsilon$.
   b. Release two jobs during $[t, t']$ as described in the proof of Lemma 31.
3. Let $\hat{t} = M + \left(\sum_{k'=2}^{k^*} 2 \cdot \frac{M}{2^{k'}} - \varepsilon\right)$. At each time $t' \in \{\hat{t} + 1 - \varepsilon, \dots, \hat{t} + 1 - \varepsilon + M\}$, release a job j with $p_{j_1} = 1$ and $p_{j_2} = 0$.

We first argue that $|J(\hat{t})| = k^* + 1$ and $|J^*(\hat{t})| = 1$. Using Lemma 32, we get that $|J(M)| = 2$ and $|J^*(M)| = 1$. For each time t' considered in Step 2 of the construction, we can apply Lemma 31. Hence, $|J^*(t')| = 1$, whereas $|J(t')|$ increases by one for each such time t'. This gives us $|J(\hat{t})| = k^* + 1$ and $|J^*(\hat{t})| = 1$.

Furthermore, we can observe that $\hat{t}$ is (k^*, M)-critical. By Definition 30 and since $M = 2^{k^*}$, this implies that there is a single operation in $J(\hat{t})$ with a remaining time of $1 - \varepsilon$, and other active operations have remaining time at least 1. By time $\hat{t}' = \hat{t} + 1 - \varepsilon$, we have $|J(\hat{t}')| = k^*$, $|J^*(\hat{t}')| = 1$, and all operations in $J(\hat{t}')$ have a remaining time at least one. The third step of the construction implies $|J(t)| = k^* + 1$ and $|J^*(t)| = 2$ for each $t \in [\hat{t}', \hat{t}' + M]$.

Since OPT never has more than two active jobs during $[0, \hat{t}' + 2M - \epsilon]$ and completes the last job at time $\hat{t}' + 2M - \epsilon$, we can conclude that $\mathrm{OPT} \leq 2 \cdot (\hat{t} + 1 - \varepsilon + M) + 2M - \epsilon \leq 2 \cdot (2M + 1 - \varepsilon) + 2M \leq 7M$, using that $\hat{t} \leq 2M$. On the other hand, we can lower bound $\mathrm{ALG} \geq k^* \cdot M$ by just considering the cost incurred during $[\hat{t}', \hat{t} + M]$. Hence, $\frac{\mathrm{ALG}}{\mathrm{OPT}} \in \Omega(k^*)$.

By observing that $n = 2 \cdot k^* + M = 2\log_2(M) + M$, we can conclude with $\frac{\mathrm{ALG}}{\mathrm{OPT}} \in \Omega(\log(n))$. ◀