An Optimal Online Algorithm for Robust Flow Time Scheduling

Anupam Gupta* Amit Kumar† Debmalya Panigrahi‡ Zhaozi Wang*

Abstract. The problem of minimizing the total flow time on a single machine is one of the few problems for which we can give an optimal online algorithm: just schedule the job with the shortest remaining processing time (SRPT). However, this requires knowledge of the true running time p_j of each job j. Azar, Leonardi, and Touitou recently asked: what if we are given estimates $\widehat{p}_j$ for each job, such that the multiplicative error between p_j and $\widehat{p}_j$ (called the *distortion*) is at most μ? It is easy to construct examples where no algorithm can be $o(\mu)$ competitive; can we get $O(\mu)$ competitiveness?

We show how to achieve this asymptotically optimal result, improving on the previous best result of $O(\mu \log \mu)$. Moreover, we give a very simple algorithm to get this tight bound of $O(\mu)$; the previous ZigZag algorithm was relatively more involved. Our proof is via a dual-fitting argument based on the idea of a *reduced instance*: we consider an LP relaxation based on knapsack-cover inequalities, and show a solution with a large dual value. Our ideas can also be extended to give alternative dual-fitting arguments for previously analyzed algorithms (like SRPT, ZigZag, and others for minimizing the total flow time, and the Bansal-Dhamdhere algorithm for weighted flow-time minimization).

1 Introduction. Consider the classic online scheduling problem of minimizing the total flow-time for a sequence of jobs on a single machine. Jobs arrive online, where each job j (arriving at some time r_j) reports its size p_j upon arrival. We have a single machine, and hence only one job can be processed at any time. The algorithm needs to decide which job to process at any time t; preemption and resumption of jobs are allowed and lossless. The goal is to minimize the sum of *flow (or response) times* of all jobs $\sum_j F_j$, where the flow-time of job j is defined as $F_j := C_j - r_j$. Here, C_j is the completion time of job j – the first time at which it has received its full processing requirement of p_j. For this basic problem, the simple *Shortest Remaining Processing Time* (SRPT) algorithm is optimal [19, 18]. Formally, if $e_j(t)$ is the *elapsed time* for job j at time t (i.e., the amount of processing it has received till time t), then SRPT runs any job in the set $\arg\min_j(p_j - e_j(t))$ at time t.

However, what if the jobs do not report their true processing time p_j, but only a *prediction* $\widehat{p}_j$? This *robust scheduling* model was proposed and studied in two (surprisingly recent) works by Azar, Leonardi, and Touitou [2, 3] (also see the *semi-clairvoyant* model of [10]). To define the quality of predictions, denote the maximum overestimate and underestimate of job sizes by $\mu_1 := \max_j \widehat{p}_j/p_j$ and $\mu_2 := \max_j p_j/\widehat{p}_j$, respectively; then the *distortion* of the prediction is $\mu := \mu_1 \cdot \mu_2$.

What can we do with predictions? We could use a non-clairvoyant policy (which completely ignores these predictions), but these are known to have a competitive ratio of $\Theta(\log n)$ [9, 16]—we would like to do better if the predictions are close to the true processing times. Alternatively, as a strawman, we

*Computer Science Department, New York University, New York NY. Email: anupam.g@nyu.edu, zw4460@nyu.edu. Supported in part by NSF awards CCF-2224718 and CCF-2422926.

†Department of Computer Science and Engineering, IIT Delhi, New Delhi, India. Email: amitk@cse.iitd.ac.in.

‡Department of Computer Science, Duke University, Durham, NC. Email: debmalya@cs.duke.edu. Supported in part by NSF awards CCF-1955703 and CCF-2329230.

can run the SRPT algorithm on the predictions (i.e., run some job which minimizes $\widehat{p}_j - e_j(t)$ at time t), but it is easy to construct examples where this algorithm performs arbitrarily badly, even when the predictions have nearly perfect quality.

Indeed, the algorithms achieving $f(\mu)$ guarantees are quite sophisticated. The work of [2] considered the setting where the distortion μ is known, and gave an $O(\mu^2)$-competitive algorithm using a new "two-list" algorithm. Subsequently, a follow-up paper by the same authors [3] gave an $O(\mu \log \mu)$-competitive algorithm, ZigZag, for the *distortion-oblivious* setting, where the algorithm is not given the value of μ, thereby improving the competitiveness in a more challenging setting. Moreover, they showed an $\Omega(\mu)$ lower bound for (randomized) algorithms, even when the distortion is known and one-sided (e.g., all predictions are under-estimates). They left open the question of getting a tight bound of $O(\mu)$ for the distortion-oblivious setting—in fact, such a bound was not known even for the distortion-aware setting.

1.1 Our Results and Techniques. Our main result resolves this question:

THEOREM 1.1 (Main Theorem). *There is a deterministic algorithm for the problem of minimizing the total flow-time in the robust scheduling model that is $O(\mu)$-competitive in the distortion-oblivious setting.*

The Algorithm. Our algorithm, which we call BALANCE, is considerably simpler than the previous ZigZag algorithm from [3], which was highly non-trivial. Indeed, our algorithm builds on intuitions from recent algorithms for robust scheduling, such as ZigZag and "two-list" [2, 3], but avoids many of their complexities without sacrificing the quality.

The idea is simple: BALANCE is essentially a "cautious" version of the *Shortest Job First* (SJF) algorithm. We group the jobs into classes, where class k contains jobs with predicted size $\widehat{p}_j \approx 2^k$. Jobs are considered *partial* if they have received non-zero processing; otherwise, they are called *full*. Now, if there is a partial job in the smallest class among all jobs in the system, then we simply run it. If the smallest class contains only full jobs, then we run any job from this class if the number of full jobs is at least a quarter of all the jobs currently in the system. Otherwise, we run a partial job of the smallest class only among partial jobs.

The Analysis. The idea behind the analysis is to focus on the full jobs. By our invariant, these are a constant fraction of all jobs, and hence accounting for them is enough to account for all jobs up to this constant. (One exception is when the total queue size is tiny, but then we can charge to the total volume of all jobs.) Focusing on full jobs makes sense, because sizes of full jobs are the only things we have a good handle on; the remaining sizes of the partial jobs could be very far (multiplicatively) from any estimates we have for them.

Suppose we want to account for the full jobs of class k that are in the system at some time $t^\star$. It would be convenient if we could identify an interval $[t', t^\star]$ where the algorithm only processed jobs of class at most k, and there were no jobs of class at most k in the system at time t'. In this case, we can claim that any feasible solution, including an optimal one, must have a certain amount of "excess" work generated in this interval (namely, the total processing time of the jobs of class at most k released in this interval, minus the interval length) jobs at time $t^\star$ as the algorithm. Furthermore, this excess can be related to the number of full jobs of class k, since these have the largest remaining processing time among all jobs of class at most k.

We show a structural property that almost achieves this. Namely, we show that there is an interval $[t', t^\star]$ such that the algorithm only processes jobs of class at most k in this interval, and there is *at most one* job of class at most k that was released before t' and possibly processed in this time interval. This extra job is not a concern when the number of full jobs in class k at time $t^\star$ is large (at least $\Theta(\mu)$), but when this number is small, we need a more careful analysis.

To develop a formal framework for our analysis, we introduce our second ingredient—dual fitting

using a knapsack-cover based LP relaxation for total flow-time. For each interval I of time of length L, if some set S of jobs arrives within that interval, the LP imposes the constraint that at most L units of the processing of S can be done in I; in other words, it "chooses" which jobs from S are still alive at the end of the interval, so that their processing time is at least the excess $\sum_{j \in S} p_j - L$ of this interval I. This is a covering knapsack problem, and we add in the knapsack cover inequalities to avoid large integrality gaps [11]; such an LP was used by, e.g., Bansal and Pruhs [8].

Our analysis now proceeds via dual fitting on this LP relaxation. Instead of maintaining a global dual solution across time, we view the goal of the dual fitting exercise as ensuring that the dual objective at any time $t^\star$ can account for the full jobs in the system at time $t^\star$. This leads to a useful concept we call a *reduced instance* (for time $t^\star$). The idea is simple: we can reduce job sizes as long as it does not change the behavior of the algorithm until time $t^\star$, since it only makes the optimal solution cheaper while not changing the algorithm's cost. Therefore, dual fitting on the reduced instance is also valid on the actual instance to account for the number of full jobs in the system at time $t^\star$. The notion of reduced instances is more than a technical convenience, they turn out to be essential in settings where job sizes are imprecise. Indeed, consider an interval $I := [t', t^\star]$ where several jobs waiting at time t' get processed in I. To lower bound the excess of this interval, one needs to show that a larger volume of jobs gets released inside I. However, if job sizes are off by a factor μ, this argument based on estimated volumes does not work. Reduced instances severely restrict settings where such events can happen and thus, enable us to identify intervals with large excess even in the prediction model.

We see our dual fitting framework as one of the major contributions of this paper. Our framework can be used to give alternative dual-fitting proofs of other classic results in online flow-time minimization—as other examples, we give analyses of SRPT and weighted flow-time minimization algorithm of Bansal and Dhamdhere [7] in the appendix. (While we do not give them here, dual-fitting can also give new proofs for DL, ZigZag, and "two-stack", all of which were proposed by [2, 3], and we hope it will be useful for future research.)

1.2 Related Work. The classical problem of minimizing total flow-time in the online setting has been well-studied. When job sizes are revealed upon arrival, the Shortest Remaining Processing Time (SRPT) algorithm is known to be optimal. In the non-clairvoyant setting, where job sizes are revealed only on completion, randomized algorithms can achieve $O(\log n)$ competitive ratio in expectation, but this is asymptotically tight [16, 9]. In the semi-clairvoyant setting, where job sizes are known up to the nearest power of 2, [10] gave a constant-competitive algorithm. This model was generalized by [2] where the declared size of an arriving job could have distortion μ. They gave an $O(\mu^2)$-competitive algorithm, and subsequently [3] improved this result to $O(\mu \log \mu)$ even in the distortion oblivious setting. They also showed that any randomized algorithm in the distortion oblivious setting is $\Omega(\mu)$ competitive.

Stronger lower bounds are known for minimizing the weighted flow-time online (WtdFlow) on a single machine. [6] showed a lower bound of $\widetilde{\Omega}(\min(\sqrt{\log W}, \sqrt{\log \log P}))$ competitiveness for deterministic algorithms; here W is the ratio of the maximum weight to the minimum weight, and P is the corresponding ratio of processing times. [12] gave an $O(\log^2 P)$-competitive algorithm and [7] gave an $O(\log W)$-competitive algorithm for WtdFlow. Subsequently, [4] improved these results to $O(\log(\min(P, D, W)))$, where D is the ratio of the maximum to the minimum density of a job. In the prediction model, [2] gave an $O(\mu^2 \log W)$ and an $O(\mu^3 \log(\mu \min(P, D)))$ competitive algorithm when the parameter μ is known. For the distortion-oblivious setting, no corresponding results are known.

This line of work also connects to the growing area of algorithms with predictions, which aim to combine machine learning forecasts with online decision-making. In the context of scheduling, prediction-augmented algorithms have been studied for minimizing total completion time [17, 13], speed scaling [5], and load balancing [14, 15, 1], among others.

Paper Outline: In Section 2 we give the LP relaxation for FlowTime (which readily extends to WtdFlow). To warm up, we give an analysis of SRPT in the distortion-free setting in Section 3 to give more intuition for the definition of the dual variables. We then present our algorithm for FlowTime, BALANCE, in Section 4, and give its dual-fitting analysis in Section 5. We show the versatility of our analysis technique by giving a different $O(\log W)$-competitiveness proof for the [7] algorithm in Section 6.

1.3 Preliminaries and Notation. We consider the online single machine scheduling setting where jobs arrive over time. Each arriving job j has two parameters associated with it: release time r_j and processing size p_j. We assume w.l.o.g. that both of these quantities are integers. The time horizon is divided into unit-sized *time slots*, each starting at a non-negative integer. A (preemptive) schedule $\mathcal{S}$ specifies the job that gets processed in each time slot $[t, t+1]$ such that (i) a job j is processed in a time slot $[t, t+1]$ only if $t \geq r_j$, and (ii) the total number of time slots in which a job gets processed is equal to p_j.

The flow-time of a job j (in a particular schedule $\mathcal{S}$) is the difference between its completion time C_j and release time r_j. The total flow-time problem (FlowTime) seeks to minimize the total flow-time of the jobs. Given a schedule $\mathcal{S}$, a job j is *alive* at a time t if $t \in [r_j, C_j)$. We shall use $A(t)$ to denote the set of alive jobs at time t. Note that the total flow-time of the jobs is also equal to $\sum_{t \geq 0} |A(t)|$.

Now we define FlowPred, the flow-time problem in the robust scheduling model, where the setting and the objective are same as those in FlowTime. However, each arriving job j only declares an estimate $\widehat{p}_j$ on its actual processing time. There are underlying parameters μ_1 and μ_2 such that the actual size p_j satisfies:

$$\widehat{p}_j/\mu_1 \leq p_j \leq \mu_2\, \widehat{p}_j.$$

We shall use $\mu := \mu_1 \cdot \mu_2$ to denote the *distortion* factor. It is important to note that the online algorithm is not aware of the parameters μ_1, μ_2 or μ.

In the weighed flow-time problem (WtdFlow), each arriving job j also specifies a weight w_j and the goal is to maintain a schedule that minimizes $\sum_j w_j(C_j - r_j)$.

2 A Covering Relaxation for Total Flow Time. Natural time-indexed LP relaxations for the total flow time problem are known to have large gaps, so we consider a stronger relaxation by adding in knapsack-cover-type constraints; such LPs have been considered before, e.g., in [8].

The variables for the LP are $x_{j,t}$ which are intended to be indicators for the event that job j has been released and is unfinished at time t. The intuition for the relaxation is as follows: for time t, let the total processing requirement of a set of jobs S released before t be denoted $p(S) := \sum_{j \in S} p_j$. Let $\ell_S := \min_{j \in S} r_j$ be the earliest release time of a job in S. Consider the interval $[\ell_S, t]$: the length of this interval is $L_S := t - \ell_S$, so at least $e(S,t) := \max(0, p(S) - L_S)$ amount of processing remains to be performed on the jobs in S. Hence, the unfinished jobs in S must add up to at least $e(S,t)$: this is like the *covering knapsack* problem, where we must cover L_S requirement by objects of size $\{p_j\}_{j \in S}$. The natural LP for this problem also has an unbounded integrality gap, but one can strengthen the LP using the knapsack-cover inequalities. We emulate the same idea here.

Formally, the LP is as follows: we have variables $x_{j,t}$ for each job j and time $t \geq r_j$; recall that job j is released at time r_j. For a job $j \in S$, let $p_j(S,t)$ denote $\min(p_j, e(S,t))$. Then the (exponentially large) LP is given by:

(LP-flow)
$$\min \textstyle\sum_j \sum_{t \geq r_j} x_{j,t}$$

(2.1)
$$\textstyle\sum_{j \in S} \min(p_j, e(S,t))\, x_{j,t} \geq e(S,t) \qquad \forall S, t$$
$$x_{j,t} \geq 0.$$

The dual linear program is

$$\text{(DP-flow)} \qquad \max \textstyle\sum_{S,t} e(S,t)\, y_{S,t}$$

$$\text{(2.2)} \qquad \textstyle\sum_{S:j\in S} \min(p_j, e(S,t))\, y_{S,t} \le 1 \qquad \forall j,t \text{ with } r_j \le t$$

$$y_{S,t} \ge 0.$$

FACT 2.1. *The optimal value of the primal (LP-flow) is at most that of the optimal total flow time. By weak duality, any dual feasible solution gives a lower bound on the optimal flow time.*

In Section 7 we show a lower bound of 2 on the integrality gap of the LP; for the rest of this discussion, we will focus on showing an upper bound on the integrality gap. We will not solve (LP-flow) directly, but use the *dual fitting* approach to show that the cost of the algorithm in question can be related to feasible duals. In particular, observe that the LP constraints decouple across time: i.e., there are no constraints that link the primal $x_{j,t}$ variables, or the dual $y_{S,t}$ variables, across time. Hence, for each timestep t^*, we set dual variables y_{S,t^*} corresponding to just t^*, satisfying the dual constraints (approximately) with objective function value close to $|A(t^*)|$, the number of unfinished jobs in the algorithm's at time t^*.

3 A Dual Fitting Analysis for SRPT. To help get a better understanding of our LP relaxation, we give a dual fitting analysis of SRPT in the distortion-free setting, and show that it is constant-competitive. While we know that SRPT is optimal for total flow time, this exercise will show the power of our LP, and also develop some tools we will use in the following sections.

We give some notation first. For a $t \in \mathbb{Z}_+$ and job j, let $p_j(t)$ denote the remaining processing of j at time t. Recall that $A(t)$ denotes the set of alive jobs at time t (including those jobs which have arrived at time t); and integrality of jobs sizes and time slots implies that $p_j(t) \ge 1$ for any $j \in A(t)$. For each time $t \in \mathbb{Z}_+$, let $j(t)$ be the job processed during the slot $[t, t+1]$, and let $q(t)$ be the remaining processing time of this job $j(t)$ at time t, i.e., $q(t) := p_{j(t)}(t)$. By the SRPT rule, $j(t) \in \arg\min_{j\in A(t)} p_j(t)$, and hence $q(t) \le p_j(t)$ for all $j \in A(t)$. By standard arguments, we assume that the machine never idles, otherwise we can argue separately for each maximal busy period, so that $j(t)$ and $q(t)$ are well-defined.

3.1 Defining the Duals. We now define dual variables $y_{S,t}$ corresponding to (DP-flow). Fix a time $t^\star$. We shall define dual variables $y_{S,t^\star}$ for suitable subsets S such that (i) the constraints (2.2) corresponding to time $t^\star$ are approximately satisfied, and (ii) the dual objective corresponding to $t^\star$, i.e., $\sum_S e(S,t^\star) y_{S,t^\star}$ is $\Omega(|A(t^\star)|)$. Since the time $t^\star$ is fixed, we shall remove references to $t^\star$ in the notations $e(S,t^\star)$ and $y_{S,t^\star}$, and just use $e(S)$ and y_S instead. We also assume that $A(t^\star)$ is non-empty, otherwise we do not need to construct dual variables corresponding to $t^\star$. To define the duals:

1. For any $x \ge 0$, let $\tau(x)$ be the earliest time before $t^\star$ such that the algorithm only processes jobs of remaining size at most x in the interval $[\tau(x), t^\star]$. I.e., $q(t') \le x$ for all $t' \in [\tau(x), t^\star]$. Define

$$S(x) = \{\text{jobs of size at most } x \text{ released during } [\tau(x), t^\star]\}.$$

2. To avoid excessive notation, let the jobs in $A(t^\star)$ be called $\{1, 2, \ldots, |A(t^\star)|\}$, with remaining processing times $p_1(t^\star) \le p_2(t^\star) \le \ldots$. We partition these jobs into buckets as follows. For an integer $k \ge 0$, let B_k be the set of jobs $j \in A(t^\star)$ of "class" k, i.e., for which $p_j(t^\star) \in [2^k, 2^{k+1})$. Suppose bucket B_k is non-empty: let job $j_k \in B_k$ be the job with the *largest* remaining processing time in this bucket; i.e., $j_k = \arg\max_{j\in B_k} p_j(t^\star)$. Define the set of jobs for which we will set

non-zero duals:[1]

$$S_k := S(p_{j_k}(t^\star)) = \{\text{jobs of size at most } p_{j_k}(t^\star) \text{ released during } [\tau(p_{j_k}(t^\star)), t^\star]\}.$$

3. There are two cases, depending on the size of bucket B_k:

   (D1) If $|B_k| = 1$ and k is the not the first non-empty bucket (we arrange buckets in increasing order of k), set $y_{S_k} = 1/e(S_k)$.

   (D2) If $|B_k| \geq 2$, set $y_{S_k} = 1/2^k$.

For case (D2), we show that the excess of the set S_k (with respect to time $t^\star$) is positive, whereas in (D1), we need to define a suitable subset with positive excess; in both cases we need to show the duals are large enough to give good objective function value, while satisfying (approximate) dual feasibility.

3.2 Lower Bounds on the Excess. The following "excess" lemma is crucial to the dual fitting analysis. The notation used here roughly parallels that used in the definition of the duals, with small changes to avoid excess notation.

LEMMA 3.1 (Excess Lemma). *For any index $i \geq 1$, let τ_i denote $\tau(p_i(t^\star))$. Let R_i be the set of jobs of size at most $p_i(t^\star)$ released during the time interval $I_i := [\tau_i, t^\star]$. Let $A_i(t^\star)$ be the jobs in $A(t^\star)$ of remaining size at most $p_i(t^\star)$. Then there is at most one job in $A_i(t^\star) \setminus R_i$, and moreover*

$$e(R_i) \geq \sum_{j \in A_i(t^\star)} p_j(t^\star) - p_i(t^\star). \tag{3.1}$$

Proof. Fix an index $i \geq 1$. For sake of brevity let p^* denote $p_i(t^\star)$. Call a job j *early* if it is released before time τ_i (i.e., $r_j < \tau_i$) and its remaining processing time at time τ_i is $p_j(\tau_i) \leq p^*$. We claim that there can be at most one early job. Indeed, suppose there is some other such job j'. Since only one of these jobs can be processed in the period $[\tau_i - 1, \tau_i]$, either $p_j(\tau_i - 1) \leq p^*$ or $p_{j'}(\tau_i - 1) \leq p^*$. But then the remaining size of the job processed at time $\tau_i - 1$ would be at most p^*, contradicting the definition of τ_i. Thus there can be at most one early job. Moreover, this early job (if it exists) can be the only job in $A(t^\star) \setminus R_i$ that has size at most p^*, hence proving the first claim of the lemma. To bound the excess of the set R_i, consider two cases:

1. Suppose there is an early job j_0 whose remaining size $p_{j_0}(\tau_i)$ becomes equal to p^* at time τ_i. In this case, the jobs in $A_i(t^\star)$ are exactly the jobs in $S'_i := \{j_0\} \cup R_i$ which have not yet finished. In particular,

   $$\textstyle\sum_{j \in A_i(t^\star)} p_j(t^\star) = \sum_{j \in S'_i} p_j(t^\star). \tag{3.2}$$

   Let α be the amount of processing received by the early job j_0 in $[\tau_i, t^\star]$. Furthermore, any job $j \in R_i$ receives $p_j - p_j(t^\star)$ amount of processing. Since the total amount of processing in the interval is $t^\star - \tau_i$,

   $$t^\star - \tau_i = \sum_{j \in R_i} (p_j - p_j(t^\star)) + p_{j_0}(\tau_i) - p_{j_0}(t^\star). \tag{3.3}$$

[1] We assume that the sets S_k are distinct, else we can sum up the dual values corresponding to the identical sets. Moreover, these sets are defined separately for each time $t^\star$; we do not explicitly write $S_{kt^\star}$ instead of S_k to reduce notation.

Rearranging, using the definition of excess and that all jobs in R_i arrived in $[\tau_i, t^\star]$, we get:

$$e(R_i) \geq \Big(\sum_{j\in R_i} p_j\Big) - (t^\star - \tau_i) \overset{(3.3)}{=} \Big(\sum_{j\in S'_i} p_j(t^\star)\Big) - p_{j_0}(\tau_i)$$
$$= \Big(\sum_{j\in S'_i} p_j(t^\star)\Big) - p^* \overset{(3.2)}{=} \Big(\sum_{j\in A_i(t^\star)} p_j(t^\star)\Big) - p^*.$$

2. The other case is when there is no job whose size reduces to p^* at time τ_i: now there is no early job and $A_i(t^\star) = R_i$. In this case, some job of size at most p^* must have arrived at time τ_i, and belongs to R_i. Moreover, all the processing during $[\tau_i, t^\star]$ is performed on the jobs in R_i, and the excess is precisely $\sum_{j\in R_i} p_j(t^\star) = \sum_{j\in A_i(t^\star)} p_j(t^\star)$.

This completes the proof of the lemma. ∎

Observe that the excess lemma gives a non-zero lower bound for all the sets R_i, except when $|A_i(t^\star)| = 1$ in which case the excess could be zero. This shows that the duals in case (D1) are finite; it also explains the exception in the dual definition for the first bucket being of unit size. We now use the excess lemma to show that desired properties of the dual solution.

3.3 Proving Constant Competitiveness. We now use the dual variables defined in Subsection 3.1 to show that SRPT is 11-competitive; while this constant can be optimized further, we do not pursue this, since the analysis of SRPT is just to give intuition for the main contribution, which is the analysis of Balance. Again, we fix a time $t^\star$ and only analyse the dual variables corresponding to this time; we also suppress references to $t^\star$ in $e(S, t^\star)$ and $y_{S,t^\star}$, and refer to them as $e(S)$ and y_S.

The first step of our proof is to show that the dual objective can approximately account for the $|A(t^\star)|$.

Lemma 3.2 (Large Dual Value). $\sum_S y_S\, e(S) \geq \frac{|A(t^\star)|-1}{2}$.

Proof. For all buckets B_k corresponding to case (D1) except the first non-empty bucket, $y_{S_k} = 1/e(S_k)$, and hence the total contribution equals the number of such buckets minus 1.

For any bucket B_k corresponding to case (D2), Lemma 3.1 shows that the excess $e(S_k)$ is at least the remaining size of all jobs other than the largest job in the bucket; hence this is at least $(|B_k| - 1)\, 2^k$. Since the dual value $y_{S_k} = 1/2^k$, the dual contribution of this bucket is

$$e(S_k)\ y_{S_k} \geq (|B_k| - 1) \geq {}^{|B_k|}/_2.$$

Summing these bounds over all buckets, we get the claim. ∎

Lemma 3.3 (Dual Feasibility). *$y/5$ satisfies the dual feasibility constraints.*

Proof. Again we fix a time $t^\star$ and only consider dual constraints (2.2) corresponding to $t^\star$. Consider a job j with $p_j \in [2^k, 2^{k+1})$. By construction, job j only participates in sets $S_{k'}$ for $k' \geq k$. Let I_1 be the set of indices $k' \geq k$ for which case (D1) occurs, and I_2 be those for which case (D2) occurs. We start with the easier case:

$$\sum_{k'\in I_2} \min(p_j, e(S_{k'})) \cdot y_{S_{k'}} \leq \sum_{k'\in I_2} p_j \cdot {}^1/_{2^{k'}} \leq \sum_{k'\in I_2} \frac{2^{k+1}}{2^{k'}}.$$

For the indices in I_1, observe that if $k' \in I_1$ is not the smallest index in I_1, and k'' is the index preceding k' in I_1, then Lemma 3.1 shows that the excess $e(S_{k'})$ is at least the total remaining time of

the jobs in $B_{k''}$, and hence at least $2^{k''}$. Thus,

$$\sum_{k' \in I_1} \min(p_j, e(S_{k'})) \cdot y_{S_{k'}} \leq e(S_{x_k}) \cdot y_{S_{x_k}} + \sum_{k' > k, k' \in I_1} p_j \cdot y_{S_{k'}}$$
$$\leq 1 + \sum_{k'' \in I_1} \frac{2^{k+1}}{2^{k''}}.$$

Now summing the two bounds above and using that all indices in $I_1 \cup I_2$ are distinct and at least k, we get a geometric sum, and a total contribution of at most 5. Hence, scaling the dual variables down by 5 gives us feasibility, and completes the proof. ∎

We now put together all the results above to show:

THEOREM 3.4. *The integrality gap of (LP-flow) is at most* 11.

Proof. Consider an instance $\mathcal{I}$ and let $\mathsf{LPOpt}(\mathcal{I})$ denote the optimal value of (LP-flow) on the instance $\mathcal{I}$. Let T be the set of times where $|A(t)| \geq 1$; Lemma 3.2 and Lemma 3.3 show that

$$\sum_{t \in T} \frac{|A(t)| - 1}{2} \leq \sum_{S,t} y(S,t) \cdot e(S,t) \leq 5 \cdot \mathsf{LPOpt}(\mathcal{I}). \tag{3.4}$$

In other words, $\sum_{t \in T} |A(t)| - \sum_j p_j \leq 10 \cdot \mathsf{LPOpt}(\mathcal{I})$. We observe that $\mathsf{LPOpt}(\mathcal{I})$ is at least $\sum_j p_j$: indeed, for each time $t < r_j + p_j$, considering the singleton set $S = \{j\}$ shows that $x_{j,t} \geq 1$. Hence, $\sum_j p_j \leq \mathsf{LPOpt}(\mathcal{I})$. Combining with the above calculation, we have

$$\sum_t |A(t)| \leq 10 \cdot \mathsf{LPOpt}(\mathcal{I}) + \sum_j p_j \leq 11 \cdot \mathsf{LPOpt}(\mathcal{I}).$$

This proves the desired result. ∎

To showcase the clean ideas behind the approach, we did not optimize the constants above. In the next section, we give our new algorithm for flow-time minimization, which achieves an $O(\mu)$-competitive bound in the robust scheduling setting with unknown distortion μ.

4 A New Algorithm for Minimizing Flow-Time. In this section, we give an $O(\mu)$-competitive algorithm BALANCE for instances of the $\mathsf{FlowPred}$ problem. Recall that in this setting, when a job j is released, we are given an estimate. We denote this estimate $\widehat{p}_j$ and the actual size p_j. We let μ_1, μ_2 denote the maximum overestimate and underestimate of job sizes, i.e., $\mu_1 := \max_j \widehat{p}_j / p_j$ and $\mu_2 := \max_j p_j / \widehat{p}_j$. We also denote $\mu := \mu_1 \mu_2$.

A job is said to be in *class* k if $2^k \leq \widehat{p}_j < 2^{k+1}$. For simplicity, we assume that all job sizes and estimates are integers and that all jobs are released at integer timesteps. The algorithm BALANCE operates in unit timesteps, i.e., at time t, a job is selected for processing for unit time until time $t+1$.

4.1 The Algorithm BALANCE**.** Let $A(t)$ denote the set of jobs that have been released before (and including) time t, but have not been completed by BALANCE at time t. We call these *active* jobs at time t. The jobs in $A(t)$ are in two categories, those that have already been partially processed and the rest that have not been processed at all. We call the first category *partial* jobs and the second category *full* jobs. The latter is denoted $A^{\mathsf{full}}(t) \subset A(t)$.

The algorithm maintains a *priority queue* on the full jobs and a *stack* on the partial jobs. We refer to these as the *full queue* (Queue) and the *partial stack* (Stack) respectively.

1. The priority order in Queue is by decreasing job class, i.e., the smallest job classes are at the front of the queue. The order of jobs in the same job class is arbitrary.

2. The order of jobs in Stack is determined by stack operations, i.e., the algorithm will always add and remove jobs at the top of the stack. We will show later that the partial stack will maintain a strict order of increasing job class, i.e., any job is of a strictly lower class than the job that is below it in the stack.

The job in front of Queue is denoted front(Queue), and the job on top of Stack by top(Stack).

At time t, first any newly arriving jobs are inserted in Queue according to the priority order. Since these jobs are full jobs, they belong to both $A^{\mathsf{full}}(t)$ and $A(t)$.

Next, BALANCE decides the job to process in this timestep t. If $A(t)$ is empty, then there is nothing to process; in the rest of the description, we assume that $A(t)$ is not empty. If one of the following conditions hold:

(a) Stack is empty, or

(b) the job front(Queue) has strictly smaller class than the job top(Stack) and

$$|A^{\mathsf{full}}(t)| \geq \frac{|A(t)|}{4}, \tag{4.1}$$

then the algorithm moves the job front(Queue) from Queue to Stack. This job is now partial, and hence will not appear in $A^{\mathsf{full}}(t+1)$. Moreover, this job becomes top(Stack) after the move.

Next, the algorithm processes the job top(Stack) for unit time in timestep t. At the end of the timestep, if the job that was just processed is finished, then it is removed from the set of active jobs for the next timestep, i.e., from $A(t+1)$. Correspondingly, the job is also removed from the partial stack Stack and the next job in the stack is exposed and becomes top(Stack) (or else the partial stack is empty).

5 The Analysis of BALANCE. We now show that BALANCE is $O(\mu)$-competitive. For a fixed time $t^\star$, let $\mathsf{opt}_{t^\star}(\mathcal{I})$ be the minimum number of jobs waiting at time $t^\star$ in any schedule for $\mathcal{I}$—note that unlike total flow-time objective, where we minimize $\sum_t |A(t)|$, we are only interested in minimizing $|A(t^\star)|$ here. Our goal is to show that for any time $t^\star$, the number of remaining jobs in BALANCE at time $t^\star$ is within $O(\mu)$ factor of $\mathsf{opt}_{t^\star}(\mathcal{I})$. The main technical tool here is to define a *reduced instance* $\mathcal{I}^{\mathsf{red}}(t^\star)$ of $\mathcal{I}$ with the following properties: (i) the behavior of BALANCE (i.e., jobs processed in each slot) till time $t^\star$ in the instances $\mathcal{I}$ and $\mathcal{I}^{\mathsf{red}}(t^\star)$ remains the same, and (ii) $\mathsf{opt}_{t^\star}(\mathcal{I}^{\mathsf{red}}(t^\star)) \leq \mathsf{opt}_{t^\star}(\mathcal{I})$.

Since the reduced instance $\mathcal{I}^{\mathsf{red}}(t^\star)$ may vary as we change $t^\star$, we cannot use the LP relaxation (LP-flow) because this relaxation treats all times uniformly. Instead, we define a new LP relaxation (LP-flow($t^\star$)) for each time $t^\star$—the constraints in this LP relaxation are a subset of those in (LP-flow), and the objective function only considers time $t^\star$.

The intuition for the construction of reduced instances is as follows: at time $t^\star$, we wish to account for all the jobs in $A^{\mathsf{full}}(t)$. For each job class k, let $A^{\mathsf{full}}_k(t)$ be the jobs of class k in $A^{\mathsf{full}}(t)$. Constructing a dual solution that accounts for $|A^{\mathsf{full}}_k(t)|$ when this quantity is large enough turns out to be relatively easy. The problematic case happens when $|A^{\mathsf{full}}_k(t)|$ is $O(1)$ (and this is where we need the truncation operation in the knapsack cover inequalities). The reduced instance tries to reduce job sizes in order to merge jobs belonging to different job classes into one; this must be done without changing the resulting schedule constructed by BALANCE. In case some job class k still has constant number of jobs, we show that one can charge to the fact that even jobs of lower classes could not be completed by time $t^\star$. We begin by proving some important properties of the BALANCE algorithm.

5.1 Properties of BALANCE**.** The first property is that jobs in Stack are arranged in strictly increasing order of their class, with jobs further up the stack having smaller classes.

CLAIM 5.1. *At any time t, the partial stack* Stack *is strictly monotone in job class, i.e. any job in the stack is of a strictly smaller class than the job immediately below it in the stack.*

Proof. We show that the claim holds inductively over time. Initially, the partial stack is empty and hence the claim holds trivially. Now suppose the claim holds at some time t. The only way the claim can be violated is if the contents of the partial stack Stack is changed. This can happen in two ways. First, suppose a job completes processing and is removed from $A(t)$ and hence from Stack. In this case, the claim continues to hold by the inductive hypothesis. Next, suppose a new job is added to Stack. Note that when this happens, the newly added job is of a strictly smaller class than the job previously at the top of Stack (or Stack is empty). Therefore, the claim continues to hold in this case as well. ∎

Next, we show that if a full job becomes partial, it must be a smallest class full job.

CLAIM 5.2. *If a job j moves from* Queue *to* Stack *at time t, it belongs to the smallest class in $A(t)$.*

Proof. This follows directly from the fact that a job j is moved from Queue to Stack only when it is in $\mathsf{front}(\mathsf{Queue})$ and has strictly smaller class than $\top(\mathsf{Stack})$. Since Queue is a priority queue based on job classes and the jobs in Stack are arranged in increasing order of class from top to bottom (Claim 5.1), the desired result follows. ∎

Claim 5.1 ensures that jobs of a smaller class than that of $\mathsf{top}(\mathsf{Stack})$ cannot be in Stack. But can Queue have jobs of class smaller than that of $\mathsf{top}(\mathsf{Stack})$? The next claim bounds the number of such jobs in $A(t)$, i.e., in both Stack and Queue. In this and subsequent proofs, we shall often refer to the first timestep when a partial job was processed. For a job j, we denote this timestep $\mathsf{first}(j)$.

CLAIM 5.3. *Suppose a job j of class k is processed at a timestep t. If there is a job of a strictly smaller class than k in $A(t)$, then there cannot be another job of class at most k in $A(t)$.*

Proof. First, note that by Claim 5.1, if there are any jobs of class $\leq k$ in $A(t)$, they must be full jobs. Now, let j_1 be a job of class $< k$ in $A^{\mathsf{full}}(t)$. Note that the condition (4.1) is not satisfied at time t, else j_1 should have been processed instead of j in timestep t. This implies that j cannot be the only job in Stack, since otherwise (4.1) holds because A^{full} is not empty. Let j' be the job immediately below j in Stack.

For the sake of contradiction, assume that there is a job j_2 (different from j_1) of class $\leq k$ in $A^{\mathsf{full}}(t)$. By Claim 5.1, job j' is of a strictly larger class than k and hence larger than the class of j, j_1, j_2. Let J' be the jobs in $A(t)$ that are released after time $t' := \mathsf{first}(j')$. We claim that all these jobs must be full jobs at time t with the exception of j. This is because if any of these jobs becomes a partial job, then it would be placed on top of Stack at some time after t', i.e., it would be above job j' in Stack. Since this job is no longer in Stack at time t, it must have been completed by then. But, such a job is not in $A(t)$ and therefore, not in J'.

Since j' was moved from Queue to Stack at time t', it follows that (4.1) was satisfied before the move. (Note that even if Stack was empty before the move, (4.1) holds trivially.) Now, we compute the change on the LHS and RHS of this inequality in the interval $[t', t]$. The change on the LHS is $|J'| - 2$: we have $|J'| - 1$ due to jobs in $A(t)$ arriving between t' and t that are full at time t with the exception of j, and -1 is due to the move of j' from Queue to Stack at time t'. In the same interval, the change in RHS is $|J'|/4$ due to jobs in $A(t)$ arriving between t' and t. Since $j, j_1, j_2 \in J'$, we have $|J'| \geq 3$. But, this would mean that (4.1) continues to hold at time t, which contradicts the fact that j was processed at time t and not j_1 which is of a smaller class. ∎

5.2 Local LP and the Reduced Instance. Next, we define the notion of *local LP* and *reduced instance*. The idea of the local LP is something we have already seen for SRPT: the linear program (LP-flow) decouples over time, and hence it suffices to set duals for each timestep separately. The local LP is merely a formalization of this idea. Indeed, consider a fixed time $t^\star$ (and hence, remove all subscripts involving $t^\star$).

The LP relaxation (LP-flow($t^\star$)) for the problem of minimizing number of alive jobs at time $t^\star$ is as follows: here we only consider jobs released by time $t^\star$. We have variables x_j for each job j denoting whether j is alive at time $t^\star$. The constraint (5.1) is the same as (2.1) except that we only restrict to time $t^\star$, and hence, S is a subset of jobs with release date at most $t^\star$.

$$\text{(LP-flow}(t^\star)) \qquad \min \textstyle\sum_{j:r_j \le t^\star} x_j$$

$$\textstyle\sum_{j \in S} \min(p_j, e(S,t^\star))\, x_j \ge e(S,t^\star) \qquad \forall S \tag{5.1}$$

$$x_j \ge 0$$

The corresponding dual is:

$$\text{(DP-flow}(t^\star)) \qquad \max \textstyle\sum_S e(S,t^\star)\, y_S$$

$$\textstyle\sum_{S: j \in S} \min(p_j, e(S,t^\star))\, y_S \le 1 \qquad \forall j \text{ with } r_j \le t^\star \tag{5.2}$$

$$y_S \ge 0$$

Henceforth, our goal is to show that the local dual above has a feasible solution, with value comparable to $|A(t^\star)|$; we drop references to $t^\star$ in $e(S,t^\star)$, and just say $e(S)$.

5.2.1 Reduced Instances. Next, we define the notion of a *reduced instance* $\mathcal{I}^{\mathsf{red}}(t^\star)$ (since $t^\star$ is fixed, we shall shorten this notation to $\mathcal{I}^{\mathsf{red}}$). Let J be the set of jobs in the original instance $\mathcal{I}$ and $J(r_j \le t^\star)$ be the jobs released by time $t^\star$. The set of jobs J^{red} in $\mathcal{I}^{\mathsf{red}}$ is given by $J(r_j \le t^\star)$ and their release dates remain unchanged. The processing time of a job in the two instances may not be same: let $q(t)$ denote the job class being processed in the timeslot $[t, t+1]$ by Balance on the original instance $\mathcal{I}$. For each job $j \in \mathcal{I}^{\mathsf{red}}$, we use p_j and p_j^{red} to denote its size in $\mathcal{I}$ and $\mathcal{I}^{\mathsf{red}}$ respectively. We set $p_j^{\mathsf{red}} := \min(p_j, \max_{t \in [r_j, t^\star)} 2^{q(t)+1})$. In other words, we reduce the job class of j to the maximum job class that gets processed during $[r_j, t^\star)$. Note that $p_j^{\mathsf{red}} \le p_j$ for all j, and $p_j^{\mathsf{red}} = p_j$ if $j \notin A^{\mathsf{full}}(t^\star)$—indeed, if $j \notin A^{\mathsf{full}}(t^\star)$, it must have been processed on at least one time slot during $[r_j, t^\star)$. Therefore, $\max_{t \in [r_j, t^\star)} q(t)$ is at least $\mathsf{class}(j)$. We shall use $\mathsf{class}^{\mathsf{red}}(j)$ to denote its job class in $\mathcal{I}^{\mathsf{red}}$. Since $p_j^{\mathsf{red}} \le p_j$ for all $j \in J^{\mathsf{red}}$, the following fact is easy to see:

Fact 5.4. $\mathsf{opt}_{t^\star}(\mathcal{I}^{red}) \le \mathsf{opt}_{t^\star}(\mathcal{I})$.

We now show the more interesting direction: the behavior of Balance does not change. Note that Balance uses a tie breaking rule: in the priority Queue, jobs of the same class can be arranged arbitrarily (in the partial job stack Stack, we know that the no two jobs have the same job class and hence, no tie breaking rule is needed there). The following result now compares the run of Balance on the two instances:

Lemma 5.5. *Let $\mathcal{S}$ be a schedule constructed by* Balance *when run on $\mathcal{I}$. Then there is a suitable tie breaking rule (that may depend on $\mathcal{S}$) such that when* Balance *is run on $\mathcal{I}^{red}$, the resulting schedule $\mathcal{S}^{red}$ is identical to $\mathcal{S}$ till time $t^\star$, i.e., for every time $t \in [0, t^\star)$, both the schedules process the same job in the slot $[t, t+1]$. In particular, the sets of full jobs alive at time $t^\star$ in the two schedules is the same.*

Proof. We prove by induction on time $t \le t^\star$ that the schedule $\mathcal{S}$ and $\mathcal{S}^{\mathsf{red}}$ are identical till time t. The hypothesis is clearly true for $t = 0$. Assume that it holds at time t also.

Let $A(t')$ and $A^{\mathsf{red}}(t;)$ denote the alive jobs at a time t' in $\mathcal{S}$ and $\mathcal{S}^{\mathsf{red}}$ respectively. Define $A^{\mathsf{full}}(t')$ and $A^{\mathsf{full,red}}(t')$ similarly. By induction hypothesis, $A(t) = A^{\mathsf{red}}(t)$ and $A^{\mathsf{full}}(t) = A^{\mathsf{full,red}}(t')$. Let Stack and $\mathsf{Stack}^{\mathsf{red}}$ denote the stacks in the two schedules respectively (and similarly define Queue and $\mathsf{Queue}^{\mathsf{red}}$). We first note a useful fact:

Claim 5.6. *Suppose a job $j \in A^{\mathsf{full}}(t)$ moves to Stack at time t. Then j also moves to Stack^{red} at time t in the schedule $\mathcal{S}^{red}$; and $p_j^{red} = p_j$. Similarly, the jobs at top of Stack and Stack^{red} at time t are the same and have the same sizes in the two instances.*

Proof of Claim 5.6. Suppose $j \in A^{\mathsf{full}}(t)$ moves to Stack at time t. Note that j gets scheduled at time t and hence, $q(t) = \mathsf{class}(j)$. Since j gets (partially) scheduled in $\mathcal{S}$, $p_j^{\mathsf{red}} = p_j$ and it has the minimum job class among the jobs in $A^{\mathsf{full}}(t)$ (Claim 5.2). We claim that j also has minimum job class in $A^{\mathsf{full,red}}(t)$. Indeed, consider a job $j' \in A^{\mathsf{full}}(t)$. Since $t \in [r_{j'}, t^\star)$ and $q(t) = \mathsf{class}(j)$, $p_{j'}^{\mathsf{red}} \geq p_j$. Thus, j also has the minimum reduced job size among all the jobs in $\mathsf{Queue}^{\mathsf{red}}$ at time t.

Now, j moves to Stack in $\mathcal{S}$ because one of the following reasons: (i) Stack is empty; or (ii) Condition (4.1) is satisfied. Since $\mathsf{Stack} = \mathsf{Stack}^{\mathsf{red}}$ at time t, and all partially processed jobs have the same sizes in the two schedules respectively, the same case would happen in $\mathcal{S}^{\mathsf{red}}$ as well. Thus, j would move to $\mathsf{Stack}^{\mathsf{red}}$ in schedule $\mathcal{S}^{\mathsf{red}}$ also (in case there are multiple jobs with job class equal to that of j in $A^{\mathsf{full,red}}(t)$, we break ties in favor of j, i.e., by breaking ties we can ensure that j is at the front of $\mathsf{Queue}^{\mathsf{red}}$. This proves the first part of the claim.

For the second part, observe that Stack and $\mathsf{Stack}^{\mathsf{red}}$ are identical at time t (even after a job moves from Stack to Queue). Since $p_j^{\mathsf{red}} = j$ for each $j \in \mathsf{Stack}$, both the stacks would have the same job on the top. ∎

We now continue with the proof of Lemma 5.5. Claim 5.6 now directly proves the induction hypothesis for $t + 1$. Indeed, if a job $j \in A^{\mathsf{full}}(t)$ moves to Stack at time t, then Claim 5.6 shows that j also moves to $\mathsf{Stack}^{\mathsf{red}}$. The claim also shows that the same job remains at the top of the two respective stacks, and hence, is processed at time t in both the schedules. ∎

Our goal is to now show that $|A^{\mathsf{full,red}}(t^\star)|$ (which is same as $|A(t^\star)|$) is within a small factor of $\mathsf{opt}_{t^\star}(\mathcal{I}^{\mathsf{red}})$. For sake of ease of notation, we assume our instance is reduced, and use $\mathcal{I}$ to also refer to the reduced instance $\mathcal{I}^{\mathsf{red}}$. In other words, we assume that the job sizes in $\mathcal{I}$ satisfy the following:

Reduction Property (RP): Every job $j \in A^{\mathsf{full}}(t^\star)$ has the property that

$$p_j \leq \max_{t \in [r_j, t^\star)} 2^{q(t)+1}, \tag{RP}$$

where $q(t)$ is the job class being processed at time t.

5.3 Dual Fitting for Local LP. In this section, we show that $|A(t^\star)|$ is within $O(\mu)$ factor of $\mathsf{opt}_{t^\star}(\mathcal{I})$—here $A(t^\star)$ is the set of alive jobs when the algorithm Balance is run on the instance $\mathcal{I}$. The proof is by constructing a dual feasible solution to (DP-flow($t^\star$)) and whose objective function value is close to $|A(t^\star)|$.

We are now ready to give the main structural properties that allow us to define suitable dual variables. Let $A_k^{\text{full}}(t^\star)$ denote the jobs in $A^{\mathsf{full}}(t^\star)$ of class k. In Claims 5.7 and 5.8, we identify the sets S of jobs for which we define dual variables in (DP-flow($t^\star$)). The next claim holds for any $A_k^{\text{full}}(t^\star)$, but it will be useful when $A_k^{\text{full}}(t^\star)$ has few jobs.

Claim 5.7. *Consider a class k such that $A_k^{\text{full}}(t^\star)$ is non-empty. Then there exists a time $t_0 \leq t^\star$*

such that the following property holds: Let S be the jobs of class $< k$ released during $[t_0+1, t^\star]$. Then

$$e(S) \geq \sum_{k'<k} \sum_{j \in A^{\text{full}}_{k'}(t^\star)} p_j.$$

Proof. Let t_0 be the last timestep before t when a job j of class $\geq k$ is processed. (If there is no such timestep, then $t_0 = 0$.) During $[t_0+1, t]$, we only process jobs of class $< k$. By the reduction property (RP), any job in $A^{\mathsf{full}}(t^\star)$ that is released during $[t_0+1, t]$ must have job class $< k$. Thus, if $j \in A^{\text{full}}_k(t^\star)$ (recall that this set has been assumed to be non-empty), then $r_j \leq t_0$. Hence, $j \in A^{\text{full}}_k(t_0)$ as well. Now Claim 5.3 implies that there is no job of class $< k$ in $A(t_0)$. Therefore any job of class $< k$ processed during $[t_0+1, t^\star]$ or in $A(t^\star)$ must have been released during this interval, and hence, belongs to the set S. The desired claim follows. ∎

Observe that the reduction property (RP) allowed us to argue in the proof above that the job $j \in A^{\text{full}}_k(t^\star)$ is also alive at time t_0. Without this property, it could have happened that j was released inside $[t_0+1, t^\star]$. Now one argument in this case would have been the following: suppose j is released inside this interval. Now we know from Claim 5.3 that there can be at most one job j' of class less than k that is in $A(t_0)$ and gets processed during $[t_0+1, t^\star]$. Can we *cancel* the contribution of j' to $e(S)$ by the size of j (note that j belongs to a higher job class)? This indeed can be done in the setting where we know job sizes precisely (with some minor changes in the definition of the set S). But when there is uncertainty (up to a factor μ) in the job sizes, such cancellation may not occur. That is why we need a more subtle argument using reduced instances.

The next claim also holds for any $A^{\text{full}}_k(t^\star)$, but it will be useful when $A^{\text{full}}_k(t^\star)$ has many jobs.

Claim 5.8. *Consider a class k such that $A^{\text{full}}_k(t^\star)$ is non-empty. Then there exists a time $t_0 \leq t$ such that the following property holds: Let S be the set of jobs of class $\leq k$ released during $[t_0+1, t]$. Then*

$$e(S) \geq \sum_{k' \leq k} \sum_{j \in A^{\text{full}}_{k'}(t^\star)} p_j - \mu_2 \cdot 2^{k+1}.$$

Proof. Let t_0 be the last timestep before t when a job j of class $> k$ is processed. (If there is no such timestep, then $t_0 = 0$.) During $[t_0+1, t]$, we only process jobs of class $\leq k$. Claim 5.3 shows that at most one job of class $\leq k$ can be present in $A(t_0)$. This job has estimated size at most 2^{k+1} and hence, actual size at most $\mu_2 \cdot 2^{k+1}$. Thus,

$$e(S) \geq \sum_{k' \leq k} \sum_{j \in A^{\text{full}}_{k'}(t^\star)} p_j - \mu_2 \cdot 2^{k+1}.$$

∎

5.3.1 Setting the Dual Variables to obtain $O(\mu)$ guarantee. We now define the dual variables corresponding to (DP-flow($t^\star$)); again we focus on a fixed time $t^\star$. Let $F(k)$ denote $A^{\text{full}}_k(t^\star)$ and call a class k *crucial* if $F(k)$ is non-empty. Let I be the set of all crucial classes. For each class $k \in I$, let $S_0(k), S_1(k)$ be the sets given by Claims 5.7 and 5.8 respectively. For ease of notation, we pretend that the sets $S_0(k)$ and $S_1(k)$ are all distinct (this will not affect the correctness of our analysis, but we will need to define y_S for suitable subsets by summing this quantity over all $S_0(k)$ or $S_1(k)$ that are equal to S).

For each (crucial) class $k \in I$, we set the dual variables as follows:

(C1) $|F(k)| < 3\mu$: we set $y_{S_0(k)} := \frac{|F(k)|}{3\mu \cdot e(S_0(k))}$.

(C2) $|F(k)| \geq 3\mu$: we set $y_{S_1(k)} := \frac{1}{\mu_2 \cdot 2^k}$.

All other dual variables y_S are set to 0. Let I_0 and I_1 denote the indices $k \in I$ satisfying conditions in case (C1) and case (C2) respectively. We now bound the dual objective value of this solution.

LEMMA 5.9 (High Dual Objective). $\sum_S e(S)\, y_S \geq \frac{|A^{\mathsf{full}}(t^\star)|}{3\mu} \geq \frac{|A(t^\star)|/4 - 1}{3\mu}$.

Proof. We split the sum $\sum_S e(S)\, y_S$ into contributions from the classes in I_0 and I_1 respectively. First consider the indices in I_0. Using (C1), we have:

$$\sum_{k \in I_0} e(S_0(k))\, y_{S_0(k)} = \sum_{k \in I_0} e(S_0(k)) \cdot \frac{|F(k)|}{3\mu \cdot e(S_0(k))} = \sum_{k \in I_0} \frac{|F(k)|}{3\mu}. \tag{5.3}$$

Next consider the set I_1. For an index $k \in I_1$, we have

$$\begin{aligned} e(S_1(k)) &\overset{\text{(Claim 5.7)}}{\geq} \sum_{k' \leq k} \sum_{j \in F(k')} p_j - \mu_2 \cdot 2^{k+1} \geq \sum_{j \in F(k)} p_j - \mu_2 \cdot 2^{k+1} \\ &\geq |F(k)| \cdot \frac{2^{k^*}}{\mu_1} - \mu_2 \cdot 2^{k+1} = (|F(k)| - 2\mu) \cdot \frac{2^{k^*}}{\mu_1}. \end{aligned}$$

Since $|F(k)| \geq 3\mu$, the expression on the r.h.s. is at least $\frac{|F(k)|}{3} \cdot \frac{2^{k^*}}{\mu_1}$. Therefore,

$$\sum_{k \in I_1} e(S_1(k))\, y_{S_1(k)} \geq \sum_{k \in I_1} \frac{|F(k)|}{3} \cdot \frac{2^{k^*}}{\mu_1} \cdot \frac{1}{\mu_2 \cdot 2^{k^*}} \geq \sum_{k \in I_1} \frac{|F(k)|}{3\mu}. \tag{5.4}$$

Combining (5.3) and (5.4), we get the first inequality in the lemma. The second inequality follows from the following claim that relates $|A^{\mathsf{full}}(t)|$ and $|A(t)|$:

CLAIM 5.10. *For any time t during* BALANCE*,*

$$|A^{\mathsf{full}}(t)| \geq \frac{|A(t)|}{4} - 1. \tag{5.5}$$

Proof of Claim 5.10. The proof is by induction on t. It is trivially true at $t = 0$. Suppose it is true at time t. If a job arrival happens at time t, then $|A^{\mathsf{full}}(t)|$ increases by 1, but $|A(t)|/4$ increases by $1/4$ only. Thus, (5.5) continues to hold. Now if we transfer a job $j \in A^{\mathsf{full}}(t)$ to Stack, it must be the case that (4.1) holds before this transfer. After this transfer, $|A^{\mathsf{full}}(t)|$ decreases by 1, but $|A(t)|$ does not change. Hence, (5.5) is satisfied.

Finally, assume that we process a job j during $[t, t+1]$ – $A^{\mathsf{full}}(t)$ does not change because j is a partial job. Now, $|A(t+1)|$ is either $|A(t)|$ or $|A(t)| - 1$ (in case j finishes). Thus, (5.5) still holds at time $t+1$. This proves the induction hypothesis. ∎

This completes the proof of Lemma 5.9. ∎

It remains to establish (approximate) dual feasibility of the constraints (5.2). Fix a job j^* of class k^* and assume $r_{j^*} \leq t^\star$. We split the sum in the l.h.s. of (5.2) for the job j^* into two parts corresponding to the index sets I_0 and I_1 respectively.

Moreover, by Claims 5.7 and 5.8, for any class k, all jobs in the sets $S_0(k)$ and $S_1(k)$ are of class at most k. Hence, we need only consider classes $k \in I$ that satisfy $k \geq k^*$. More precisely, define $I_0(j^*) := \{k \geq k^* : k \in I_0, j^* \in S_0(k)\}$, and analogously, $I_1(j^*)$. We first bound the contribution from indices in $I_0(j^*)$:

Claim 5.11 (Dual Feasibility-I). $\sum_{k\in I_0(j^*)} \min(e(S_0(k)), p_{j^*})\, y_{S_0(k)} < 10$.

Proof. Define $T_\ell := \{k \in I_0(j^*) : |F(k)| \in [2^\ell, 2^{\ell+1})\}$ to be the classes with approximately 2^ℓ full jobs in them. Since each class $k \in I_0(j^*)$ satisfies $|F(k)| < 3\mu$, the set T_ℓ can be non-empty only when when $\ell \le \Gamma := \lfloor \log_2(3\mu) \rfloor$. Now using the definition of $y_{S_0(k)} = \frac{|F(k)|}{3\mu \cdot e(S_0(k))}$, we get

$$\begin{aligned}
\sum_{k\in I_0(j^*)} \min(e(S_0(k)), p_j)\, y_{S_0(k)} &= \sum_{k\in I_0(j^*)} \min(e(S_0(k)), p_j) \cdot \frac{|F(k)|}{3\mu \cdot e(S_0(k))} \\
&\le \sum_{\ell=0}^{\Gamma} \sum_{k\in T_\ell} \min(e(S_0(k)), p_j) \cdot \frac{2^{\ell+1}}{3\mu \cdot e(S_0(k))} \\
&\le \sum_{\ell=0}^{\Gamma} \frac{2^{\ell+1}}{3\mu} \cdot \sum_{k\in T_\ell} \min\left(1, \frac{p_j}{e(S_0(k))}\right) \\
&\le \sum_{\ell=0}^{\Gamma} \frac{2^{\ell+1}}{3\mu} \cdot \underbrace{\sum_{k\in T_\ell} \min\left(1, \frac{2^{k^*+1}\cdot\mu_2}{\sum_{k'<k}\sum_{j\in F(k')} p_j}\right)}_{(\star)}, \qquad (5.6)
\end{aligned}$$

where (5.6) follows from Claim 5.7 and the fact that a job of class k^* is of size at most $\mu_2 \cdot 2^{k^*+1}$. Let us focus on the term $(\star)$ for some ℓ, and let the indices in T_ℓ arranged in increasing order be $(k_1, k_2, \ldots, k_{|T_\ell|})$. Then $(\star)$ equals

$$\sum_{i=1}^{|T_\ell|} \min\left(1, \frac{\mu_2 \cdot 2^{k^*+1}}{\sum_{k'<k_i}\sum_{j\in F(k')} p_j}\right) \le 1 + \sum_{i=2}^{|T_\ell|} \frac{\mu_2 \cdot 2^{k^*+1}}{\sum_{j\in F(k_{i-1})} p_j},$$

where we bounded the first term in the summation by 1, and the rest by the second term inside the min expression (and further reduced the denominator in each of these terms). Note that for each $k \in T_\ell$, $\sum_{j\in F(k)} p_j \ge |F(k)|2^k/\mu_1 \ge 2^k \cdot 2^\ell/\mu_1$. Hence, we get:

$$(\star) \le 1 + \sum_{i=2}^{|T_\ell|} \frac{\mu_2 \cdot 2^{k^*+1}}{2^\ell \cdot (2^{k_{i-1}}/\mu_1)} = 1 + \mu \cdot \frac{2^{k^*+1}}{2^\ell} \cdot \sum_{i=2}^{|T_\ell|} \frac{1}{2^{k_{i-1}}} \le 1 + \mu \cdot \frac{2^{k^*+2}}{2^\ell} \cdot \frac{1}{2^{k_1}}. \qquad (5.7)$$

Let $K_\ell \ge k$ be the first class contained in T_ℓ (i.e., k_1 in the above inequality), then we can use this change of notation to substitute (5.7) into (5.6) to get

$$\begin{aligned}
\sum_{\ell=0}^{\Gamma} \frac{2^{\ell+1}}{3\mu} \cdot \sum_{k\in T_\ell} \min\left(1, \frac{2^{k^*+1}\cdot\mu_2}{\sum_{k'<k}\sum_{j\in F(k')} p_j}\right) &\le \sum_{\ell=0}^{\Gamma} \mathbf{1}_{(|T_\ell|\neq\varnothing)} \cdot \frac{2^{\ell+1}}{3\mu} \cdot \left(1 + \mu \cdot \frac{2^{k^*+2}}{2^\ell} \cdot \frac{1}{2^{K_\ell}}\right) \\
&\le \sum_{\ell=0}^{\Gamma} \frac{2^{\ell+1}}{3\mu} + \frac{2^{k^*+3}}{3} \cdot \underbrace{\sum_{\ell=0}^{\Gamma} \left(\mathbf{1}_{(|T_\ell|\neq\varnothing)} \cdot \frac{1}{2^{K_\ell}}\right)}_{(\dagger)}. \qquad (5.8)
\end{aligned}$$

The classes K_ℓ for different values of ℓ are all different (because each index k can lie in only one of the sets T_ℓ), and each of these is at least k^*. Hence, the sum $(\dagger)$ is at most $2/2^{k^*}$. In turn, the entire sum in the r.h.s. of (5.8) is at most

$$\frac{1}{3} \cdot \left(\frac{2^{\Gamma+2}}{\mu} + \frac{2^{k^*+3}\cdot 2}{2^{k^*}}\right) < 10,$$

since $2^\Gamma < 3\mu$. This proves the desired result. ∎

Finally, we consider the classes $k \in I_1(j^*)$.

Claim 5.12 (Dual Feasibility-II).
$\sum_{k \in I_1(j^*)} \min(e(S_1(k)), p_{j^*})\, y_{S_1(k)} < 4$.

Proof. For any $k \in I_1(j^*)$, we know that $y_{S_1(k)} = \frac{1}{\mu_2 \cdot 2^k}$. Therefore,

$$\sum_{k \in I_1(j^*)} \min(e(S_1(k)), p_{j^*})\, y_{S_1(k)} \le \sum_{k \ge k^*} p_{j^*} \cdot \frac{1}{\mu_2 \cdot 2^k} \le \sum_{k \ge k^*} \mu_2 \cdot 2^{k^*+1} \cdot \frac{1}{\mu_2 \cdot 2^k} < 4.$$

∎

Combining Claims 5.11 and 5.12 shows that the dual variables are approximately feasible:

Lemma 5.13. *For any job j^* with $r_{j^*} \le t^\star$, $\sum_{S : j^* \in S} \min(e(S), p_{j^*})\, y_S < 14$.*

5.3.2 Putting it together. Now we combine the results in the previous sections to exhibit guarantees for the Balance algorithm. Consider an instance $\mathcal{I}$ of flow-time minimization problem. Let T be the set of timesteps t where Balance processes a job. Note that $T = \sum_j p_j$. Fix a time $t^\star \in T$. We construct the reduced instance $\mathcal{I}^{\mathsf{red}}$ corresponding to $t^\star$. Let y_S be the dual solution constructed in Subsection 5.3.1. Lemma 5.13 shows that $y_S/14$ is a feasible solution to (DP-flow($t^\star$)). Combining this result with Lemma 5.9, we get (recall that $\mathsf{opt}_{t^\star}(\mathcal{I}^{\mathsf{red}})$ is the optimal solution value to the instance $\mathcal{I}^{\mathsf{red}}$ and $A^{\mathsf{red}}(t^\star)$ is the number of alive jobs at time $t^\star$ when Balance is run on $\mathcal{I}^{\mathsf{red}}$):

$$|A^{\mathsf{red}}(t^\star)| \le 4 + O(\mu) \cdot \mathsf{opt}_{t^\star}(\mathcal{I}^{\mathsf{red}}).$$

Fact 5.4 and Lemma 5.5 imply that

$$|A(t^\star)| \le 4 + O(\mu) \cdot \mathsf{opt}_{t^\star}(\mathcal{I}).$$

Summing over all $t^\star \in T$, we get

$$\sum_t |A(t)| \le 4 \sum_j p_j + O(\mu) \cdot \sum_t \mathsf{opt}_t(\mathcal{I}).$$

Finally observe that $\sum_t |A(t)|$ is the total flow-time incurred by Balance, and $\mathsf{opt}_t(\mathcal{I})$ is at most the number of alive jobs at time t in the optimal schedule for the FlowTime problem on $\mathcal{I}$. Combining these observations, we get:

Theorem 5.14. *The algorithm* Balance *is $O(\mu)$-competitive for the problem of minimizing total flow-time in the prediction model.*

6 Dual Fitting for Weighted Flow-Time Scheduling. In this section, we show that the dual fitting approach can be used to analyze the $O(\log W)$-competitive algorithm of [7]. Consider an instance $\mathcal{I}$ of the weighted flow-time problem (WtdFlow), where each job j has weight w_j. By rescaling and rounding, we assume the following: each weight is a power of 2, and there are $K = O(\log W)$ distinct weights $1 = W_1 < W_2 < \ldots < W_K = W$. We use *weight class k* to refer to jobs of weight W_k. Henceforth, the goal is to show that the algorithm in [7], which we call WtdSRPT, is $O(K)$-competitive.

The WtdSRPT algorithm proceeds as follows. For each time t and weight W_k, let $A_k(t)$ be the jobs of weight W_k alive at time t. Let $k(t)$ be the index $k \in [K]$ for which the total weight of the jobs in $A_k(t)$ is maximized, where we break ties in favor of higher indices. The algorithm selects a job in $A_{k(t)}(t)$ with the smallest remaining processing time. In other words, it runs SRPT on the weight class with highest total weight among the jobs in $A(t)$.

The LP relaxation is identical to (LP-flow), except with objective function $\sum_{j,t \geq r_j} w_j x_{j,t}$. We again fix a time $t^\star$ and show that the total weight of the jobs in $A(t)$ in WtdSRPT is at most some factor of that in $\mathsf{opt}_{t^\star}(\mathcal{I})$. Here $\mathsf{opt}_{t^\star}(\mathcal{I})$ denotes the minimum weight remaining at time $t^\star$ in any schedule for $\mathcal{I}$. For this, we again define the local LP relaxation as in (LP-flow($t^\star$)) with the appropriate change in the objective function. The dual LP becomes:

$$\text{(DP-wtdflow}(t^\star)) \qquad \max \textstyle\sum_S e(S, t^\star)\, y_S$$

$$\text{(6.1)} \qquad \textstyle\sum_{S: j \in S} \min(p_j, e(S, t^\star))\, y_S \leq w_j \qquad \forall j \text{ with } r_j \leq t^\star$$

$$y_S \geq 0.$$

As earlier, we will use $e(S)$ instead of $e(S, t^\star)$ for brevity.

6.1 The Weighted Excess Lemma. In order to define the duals, we follow the same template as for Balance in Section 5, and for SRPT in Section 3. We fix a time $t^\star$ and first show suitable subsets of jobs whose excess is large. To get intution for the setting of dual variables, recall that we defined sets S in the unweighted setting with large excess using the following method: given a size q, we found the earliest time t_0 such that the algorithm processed jobs of size at most q during $[t_0, t^\star]$. In the weighted case, it is natural to replace the size q by the *density* d.[2] However, since the algorithm does not treat jobs of equal density d from various weight classes in a uniform manner, we should be careful. Recall that the algorithm first selects the weight class which has highest remaining weight, and then chooses the job with least remaining time from it. To capture this in our dual fitting framework, we introduce the following ideas:

(i) For each class k, we think of each job as a rectangle of width W_k and arrange these job rectangles in descending order of their remaining size at time $t^\star$. In some sense, all jobs whose intervals hit a particular vertical line (which we call *position*) are treated equally by the algorithm. (See Figure 6.1.)

(ii) Given a density d, we identify K values $\pi_1, \ldots, \pi_K$, which roughly correspond to the remaining processing time of jobs hitting a suitable vertical line. The set S contains jobs released during a time interval $[t_0, t^\star]$ where t_0 satisfies the following condition: for each time t in this interval, if the algorithm process a job from weight class k during $[t, t+1]$, then the remaining time of this job is at most π_k.

We now make these ideas more precise.

6.1.1 Positions and Anchors. Consider the jobs $j_1, j_2, \ldots, j_\ell$ in $A_k(t^\star)$, i.e., alive jobs in class k at time $t^\star$, ordered in non-increasing order of remaining processing times at time $t^\star$, breaking ties in favor of jobs released earlier. (See Figure 6.1.)

- Define the *rank* of the i^{th} largest job j_i to be i.
- Define $\mathsf{firstpos}(j_i) := (i-1)W_k$ and $\mathsf{endpos}(j_i) := iW_k$.
- Job j_i occupies *positions* in the interval $\mathsf{Interval}(j_i) := ((i-1)W_k, iW_k]$ of width W_k.
- Let $\mathsf{After}(j_i)$ be the jobs appearing after j_i in this sequence, i.e., $\{j_{i+1}, j_{i+2}, \ldots\}$.

All these definitions are with respect to alive jobs and remaining sizes at time $t^\star$, although we do not explicitly mention $t^\star$ in the notation. Since weights are powers of 2, the intervals $\mathsf{Interval}(j)$ and $\mathsf{Interval}(j')$ for any two jobs j, j' are either disjoint, or one is contained inside the other.

[2] Recall that density of a job at time t is $w_j/p_j(t)$, the weight of the job, divided by its remaining size at time t; the analogue of choosing jobs of size at most q would be to choose jobs with density *at least* some threshold.

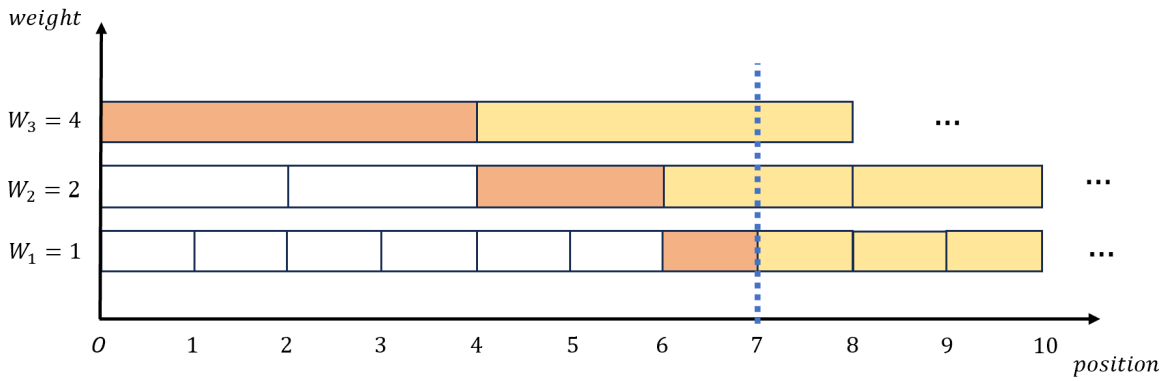

Figure 6.1: The geometric representation of jobs in $A(t)$, where each row corresponds to a weight class; jobs to the right in each row have smaller remaining sizes. The anchor corresponding to position $p = 7$ is given by the orange boxes, and the yellow boxes show $\cup_k \mathsf{After}(j^k)$.

For simplicity, we assume that there exist an infinite number of "virtual" jobs for each weight class k, each with zero remaining processing time. This ensures that for any position $p > 0$, there is a unique job j of each weight class for which $p \in \mathsf{Interval}(j)$. We now give a key definition:

Definition 6.1 (Anchor). *For a position $p \geq 0$, define $\mathsf{Anc}(p)$ to be the sequence of jobs $(j^1, \ldots, j^K)$, one of each weight class $k \in [K]$, where j^k is the job in $A_k(t^\star)$ of largest rank having $\mathsf{endpos}(j^k) \leq p$.*

Observe the following property of anchors:

Claim 6.2. *For any position p, the anchor $\mathsf{Anc}(p) := (j^1, j^2, \ldots, j^K)$ satisfies the following condition: for all pairs k, k' with $1 \leq k < k' \leq K$:*

$$\mathsf{endpos}(j^{k'}) \leq \mathsf{endpos}(j^k) \leq \mathsf{endpos}(j^{k'}) + W_{k'}. \tag{6.2}$$

For some time t, let $k(t)$ and $q(t)$ be the weight class and the remaining processing time of the job being processed by WtdSRPT at time t. We now give the "Excess Lemma" identifying sets with large excess.

Lemma 6.3 (Excess Lemma for Weighted Flow). *Let $\mathsf{Anc}(p) = (j^1, j^2, \ldots, j^K)$ be the anchor for a position p, and let $\pi_k := p_{j^k}(t^\star)$ be the remaining processing time of job j^k at time $t^\star$. Let t_0 be the earliest time such that the following condition is satisfied for each time $t \in [t_0, t^\star]$ and each $k \in [K]$: if $k(t) = k$, then $q(t) \leq \pi_k$. Let R be all the jobs released in the time period $[t_0, t^\star]$. Then*

$$e(R) \geq \sum_{k=1}^{K} \sum_{j \in \mathsf{After}(j^k)} p_j(t^\star).$$

Proof. Let $T(k)$ denote the total weight of the jobs in $A_k(t_0 - 1)$. We first claim that $T(k)$ cannot be too large for any $k \in [K]$.

Claim 6.4. *$T(k) \leq \mathsf{endpos}(j^k)$ for all $k \in [K]$.*

Proof. Consider the job j_0 processed at time $t_0 - 1$, and let k_0 be its weight class. By the definition of the algorithm, $T(k) \leq T(k_0)$ for all k, and the inequality is strict for $k > k_0$. Moreover, job j_0 was the right-most job in its sequence at time $t_0 - 1$, and had the least remaining size in its weight class, which at time $t_0 - 1$ was more than π_{k_0} by the choice of t_0. Suppose it had rank z at that time—then

all $z-1$ jobs in its weight class to its left at that time had remaining size at time t_0 strictly larger than π_{k_0}. They were not processed in the interval $[t_0, t^\star]$, so they lie to the left of job j^{k_0} (which has remaining size π_{k_0}) at time $t^\star$. Hence $\mathsf{endpos}(j^{k_0})$ at time $t^\star$ is at least $z \cdot W_{k_0} = T(k)$.

For any other weight class $k \neq k_0$, we do a case analysis depending on if it is smaller than k_0:

1. $k < k_0$: we know from Claim 6.2 that $\mathsf{endpos}(j^k) \geq \mathsf{endpos}(j^{k_0})$, which gives us
$$T(k) \leq T(k_0) \leq \mathsf{endpos}(j^{k_0}) \leq \mathsf{endpos}(j^k).$$
2. $k > k_0$: we know from Claim 6.2 that $\mathsf{endpos}(j^k) \geq \mathsf{endpos}(j^{k_0}) - W_k$. Now we get
$$T(k) < T(k_0) \leq \mathsf{endpos}(j^{k_0}) \leq \mathsf{endpos}(j^k) + W_k;$$
note the strict inequality at the beginning. Since $T(k)$ and $\mathsf{endpos}(j^k)$ are both integer multiples of W_k, we have that $T(k) \leq \mathsf{endpos}(j^k)$. ∎

Let O^k be the "old" jobs in $A_k(t_0 - 1)$ that receive processing during $[t_0, t^\star]$. We claim that if $j \in O^k$, then $p_j(t^\star) < \pi_k$. Indeed, any job j processed in this interval (say in slot $[t, t+1]$ has $p_j(t) \leq \pi_k$ and hence $p_j(t^\star) \leq p_j(t+1) \leq \pi_k - 1$. Thus, all jobs in O^k must have rank strictly greater than that of j^k (or have finished).

Let ℓ_k be the rank of j^k at time $t^\star$. Claim 6.4 implies that $A_k(t_0 - 1)$ has *at most* ℓ_k jobs; moreover, we just argued that $|O^k|$ of these jobs appear after j^k at time $t^\star$. Hence at least $|O^k|$ jobs of weight class k and size at least π_k must have been released during $[t_0, t^\star]$. Let I^k be the timesteps in $[t_0, t^\star]$ when the algorithm processes a job of weight class k, and let R^k be the subset of the newly released jobs R of weight class k. Then we have

$$\begin{aligned} e(R) = \sum_{k \in [K]} \Big(\sum_{j \in R^k} p_j - |I^k| \Big) &\geq \sum_{k \in [K]} \Big(\sum_{j \in R^k} p_j(t^\star) - \sum_{j \in O^k} (p_j(t_0) - p_j(t^\star)) \Big) \\ &= \sum_{k \in [K]} \Big(\sum_{j \in R^k \cup O^k} p_j(t^\star) - \sum_{j \in O^k} p_j(t_0) \Big), \end{aligned}$$

where the inequality uses that each timestep in I^k processes a job from R^k or O^k. Finally, for any job $j \in O^k$, we have $p_j(t_0) \leq \pi_k$; moreover, there are at least $|O^k|$ jobs of weight class k and size at least π_k released in this interval (which appear *before* j^k in the ranking at time $t^\star$, and hence are not in $\mathsf{After}(j^k)$). The latter $|O^k|$ terms contribute at least π_k each to the first summation over $R^k \cup O^k$, and compensate for the subtraction of the terms (which are at most π_k) in the summation over O^k. The remaining expression is a superset of those in $\mathsf{After}(j^k)$, which gives:

$$e(R) \geq \sum_k \sum_{j \in \mathsf{After}(j^k)} p_j(t^\star).$$

This proves the desired result. ∎

6.2 Setting Dual Variables. Recall that we are setting dual variables for (DP-wtdflow($t^\star$)). Let I_k be the time slots in $[0, t^\star]$ where a job of weight class k gets processed. As in Subsection 5.2.1, we assume w.l.o.g. that the input $\mathcal{I}$ satisfies the following reduction property:

Reduction Property (RPwtd): For every job j with weight $w_j = W_k$, release date $r_j \leq t^\star$ and remaining processing $p_j(t^\star) = p_j$ (i.e., it is "full"), we have the property that

$$p_j \leq \max_{t \in I_k \cap [r_j, t^\star)} q(t), \tag{RPwtd}$$

where $q(t)$ is the remaining job size being processed at time t.[3]

Now we give a crucial definition of the *density* of a position (at time $t^\star$).

Definition 6.5 (Density). *Given a position p and weight class k, define $j(p,k)$ to be the unique job in $A_k(t^\star)$ such that $\mathsf{Interval}(j(p,k))$ contains p. Define the density $\mathsf{den}(p)$ of a position p as*

$$\mathsf{den}(p) := \min_{k=1}^{K} \frac{W_k}{p_{j(p,k)}(t^\star)}.$$

Definition 6.6 (Density Block). *For integer $d \in \mathbb{Z}$, define $\mathsf{Block}(d)$ as the set of positions p for which $\mathsf{den}(p) \in [2^d, 2^{d+1})$.*

Since the jobs in each weight class are arranged in non-increasing order of remaining processing times, $\mathsf{den}(p)$ is non-decreasing as a function of p; it is finite if at least one job $j(p,k), k \in [K]$ is not virtual. The monotonicity of $\mathsf{den}(p)$ with respect to p implies that $\mathsf{Block}(d)$ is an interval. Moreover, if $p \in \mathbb{Z}$, the density of any position in the unit interval $[p, p+1]$ is constant, and hence any non-empty density block $\mathsf{Block}(d)$ contains at least one $\mathsf{Interval}(j)$ for some job j with $w_j = 1$.

We define several parameters for $\mathsf{Block}(d)$:

Definition 6.7 (Parameters for $\mathsf{Block}(d)$). *Given an integer d:*

1. *Let k_d be the largest weight class k such that $\mathsf{Block}(d)$ contains $\mathsf{Interval}(j)$ with $j \in A_k(t^\star)$; by the observation above, this is well-defined.*
2. *Let j_d be the lowest ranked (i.e., leftmost) job of class k_d such that $\mathsf{Interval}(j_d) \subseteq \mathsf{Block}(d)$.*
3. *Finally, let g_d denote the position $\mathsf{firstpos}(j_d)$.*

Now we are ready to define the dual variables. For each non-negative d such that $\mathsf{Block}(d)$ is non-empty, apply Lemma 6.3 with position g_d, and let S_d be the set (called R in the lemma statement) returned by it. Define

$$y_{S_d} := \frac{|\mathsf{Block}(d)|}{e(S_d)}, \tag{6.3}$$

where $|\mathsf{Block}(d)|$ refers to the length of the interval $\mathsf{Block}(d)$. We now show that $e(S_d) > 0$, and so y_{S_d} is well-defined.

Claim 6.8. *Given a non-negative integer d such that $\mathsf{Block}(d)$ is non-empty, $e(S_d) \geq \frac{|\mathsf{Block}(d)|}{2^{d+2}}$.*

Proof. Let the first and the last positions in $\mathsf{Block}(d)$ be f_d and ℓ_d respectively; i.e., $\mathsf{Block}(d) = (f_d, \ell_d]$. Recall that $g_d = \mathsf{firstpos}(j_d)$. (See Figure 6.2.) We make the following observations:

(i) The gap between f_d and g_d is less than W_{k_d}. Since $\mathsf{Block}(d)$ contains $\mathsf{Interval}(j_d)$, it follows that the interval $[g_d, \ell_d]$ has length at least half of $|\mathsf{Block}(d)|$.

(ii) For every weight class $k \leq k_d$, the job $j^k \in \mathsf{Anc}(g_d)$ ends at position g_d, hence every position $p \geq g_d$ has a job of weight class k in $\mathsf{After}(j^k)$ crossing that position.

(iii) Moreover, for each position $p \geq g_d$, there exists a job j_p of weight class $k_p \leq k_d$ such that the density of this job j_p, namely $W_{k_p}/p_{j_p}(t^\star)$ is less than 2^{d+1}. In other words, there exists some job of weight class at most k_d which is a witness for this position having density less than 2^{d+1}—this is because k_d was chosen to the largest weight class for which some $\mathsf{Interval}(j)$ of that weight class is contained in $\mathsf{Block}(d)$.

[3] In Subsection 5.2.1, $q(t)$ was the size *class* being processed, whereas here $q(t)$ is the actual remaining size of the job processed at time t.

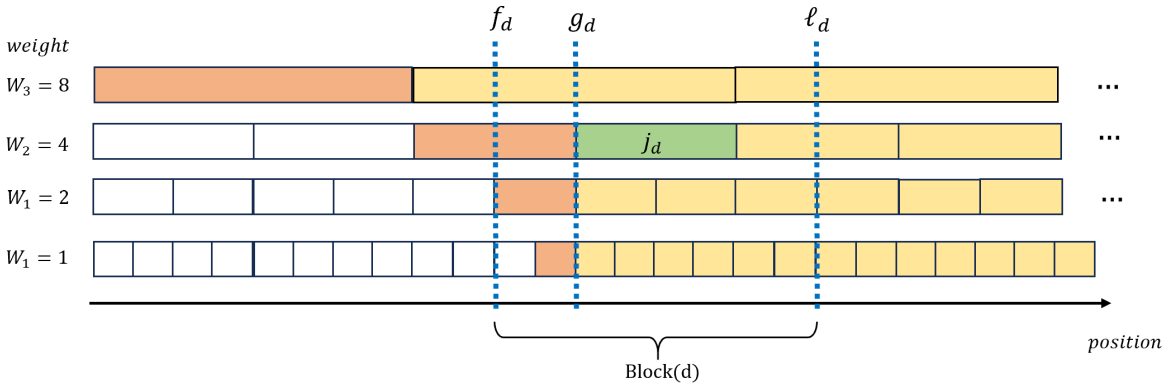

Figure 6.2: Proof of Claim 6.8: given $\mathsf{Block}(d)$, the green block is job j_d, the orange jobs are $\mathsf{Anc}(g_d)$, and the yellow/green blocks are $\mathsf{After}(j^k)$. Due to divisibility, each weight class $k \le k_d$ has its anchor job ending at g_d.

Now we can use these facts (and that $W_{k_p} \ge 1$) to get:

$$e(S_d) \overset{(6.3)}{\ge} \sum_k \sum_{j \in \mathsf{After}(j^k)} p_j(t^\star) \overset{\text{(iii)}}{\ge} \int_{p \ge g_d} \frac{W_{k_p}}{2^{d+1}} \overset{\text{(i)}}{\ge} \frac{|\mathsf{Block}(d)|/2}{2^{d+1}},$$

which is what we claimed. ∎

6.2.1 Large Dual Objective. We now show that the dual values can account for the weighted flow-time incurred by the algorithm:

Lemma 6.9. $\sum_S e(S) \cdot y_S \ge 1/K \cdot \sum_{j \in A(t)} w_j$.

Proof. Let D be the set of densities for which $\mathsf{Block}(d)$ is non-empty, and hence we define duals.

$$\sum_S e(S) \cdot y_S = \sum_{d \in D} e(S_d) \cdot y_{S_d} = \sum_{d \in D} e(S_d) \cdot \frac{|\mathsf{Block}(d)|}{e(S_d)} = \sum_{d \in D} |\mathsf{Block}(d)|.$$

Observe that $\sum_{d \in D} |\mathsf{Block}(d)|$ is equal to the total weight of the weight class with most weight; in turn, that is at least the total weight in the system at time $t^\star$ divided by K, which proves the desired result. ∎

6.2.2 Approximate Dual Feasibility. In this section, we show that constraints (DP-wtdflow($t^\star$)) are approximately satisfied.

Lemma 6.10 (Approximate Dual Feasibility). *Consider a job j^* with $r_{j^*} \le t^\star$. Then,*

$$\sum_{S : j^* \in S} \min\{p_{j^*}, e(S)\} \cdot y_S = \sum_{d \in D : j^* \in S_d} \min\{p_{j^*}, e(S_d)\} \cdot y_{S_d} \le 11 w_{j^*}.$$

Proof. Suppose the job j^* is of weight class k^*, and that its initial density is $\frac{W_{k^*}}{p_{j^*}} \in [2^{d^*}, 2^{d^*+1})$. We break the analysis into two cases::

1. Density classes $d \le d^*$: here we have

$$\sum_{d \le d^* : j^* \in S_d} \min\{p_{j^*}, e(S_d)\} \cdot y_{S_d} \le p_{j^*} \cdot \sum_{d \le d^* : j^* \in S_d} y_{S_d} \le \frac{W_{k^*}}{2^{d^*}} \sum_{d \le d^*} \cdot 2^{d+2} = 8 W_{k^*},$$

where we used Claim 6.8 and the definition of y_{S_d} in the last inequality.

2. The remaining density classes $d > d^*$: for these, we have

$$\sum_{d>d^*:j^*\in S_d} \min\{p_{j^*}, e(S_d)\} \cdot y_{S_d} \le \sum_{d>d^*:j^*\in S_d} e(S_d) \cdot y_{S_d} = \sum_{d>d^*:j^*\in S_d} |\mathsf{Block}(d)|.$$

We claim that this summation is at most $3W_{k^*}$. For a given density class $d > d^*$ with $j^* \in S_d$, let j_d be the "defining job" with starting time g_d, and let its anchor $\mathsf{Anc}(g_d)$ be $(j_d^1, \ldots, j_d^K)$. Now, the reduction property (RPwtd) implies that the original processing size of any job of class k in S_d is at most $p_{j_d^k}(t^\star)$ for any $k \in [K]$, and hence the initial density of such a job is at most the density of j_d^k at time $t^\star$.

(i) Consider the jobs in the sequence $A_{k^*}(t^\star)$: the density of these jobs increases as their rank increases. Let j' be the first job in this sequence with density at least 2^{d^*+1}. We claim that job $j^* \in S_d$ only if $g_d \in \mathsf{Interval}(j')$. Indeed, we show that neither of the other two possibilities: (i) $g_d < \mathsf{firstpos}(j')$ or (ii) $g_d > \mathsf{endpos}(j')$ can occur.

If case (i) happens, then $d = \mathsf{den}(g_d) \le d^*$ by definition of j'. But this is a contradiction since we have assumed $d > d^*$. If case (ii) happens, then the job $j_d^{k^*}$ in $\mathsf{Anc}(g_d)$ would be j' or a higher ranked job in $A_{k^*}(t^\star)$. Thus, the density of $j_d^{k^*}$ would be at least that of j', i.e., 2^{d^*+1} or higher. But the reduction property would imply that j^* also has density at least 2^{d^*+1}, a contradiction to the definition of d^*.

In summary, g_d must lie in some interval of width at most $\mathsf{endpos}(j') - \mathsf{firstpos}(j') = W_{k^*}$.

(ii) We claim that if $j^* \in S_d$, then $k_d < k^*$, and hence $|\mathsf{Block}(d)| \le 2W_{k_d} \le W_{k^*}$. Indeed, suppose not and $k_d \ge k^*$: then $\mathsf{endpos}(j_d^{k^*}) = g_d$ as well. (See Figure 6.2.) But then any job of weight class k^* in S_d needs to have density class at least d (due to the reduction property), which contradicts the fact that $d^* < d$.

Thus, g_d needs to lie in a range of width W_{k^*}, and since the maximum width of any $|\mathsf{Block}(d)| \le W_{k^*}$, we see that the total length of the positions occupied by $\mathsf{Block}(d)$ where $d > d^*$ and $j^* \in S_d$ is at most $3W_{k^*}$. Finally, since the blocks $\mathsf{Block}(d)$ are disjoint, the sum of their widths is at most $3W_{k^*}$ as well.

Combining the above two cases, we get the desired result. ∎

7 Integrality Gap Lower Bound. In this section, we show the following lower bound on the integrality gap of the LP:

THEOREM 7.1. *For any $\varepsilon > 0$, the integrality gap of the linear program (LP-flow) is at least $2 - \varepsilon$.*

Proof. Consider the following instance $\mathcal{I}$:

1. (type-A jobs) For each $i \in \{0, 1, \ldots, k-1\}$, a job a_i of size 2^{k-i} is released at time $2^k - 2^{k-i}$, and then

2. (type-B jobs) for each $i = 0, 1, \ldots, T-1$, a job b_i of size 1 is released at time $2^k - 1 + i$.

Observe that the optimal algorithm (SRPT) would switch to processing each arriving job as soon as it arrives (where we can break ties in favor of the new jobs, without loss of generality), and the finally finish the remaining jobs at the end. Hence, at all times in the range $[2^k - 1, 2^k - 1 + T]$ we have $k + 1$ jobs alive in the system, giving a total flow time of at least $(k+1)T$.

Now consider the following fractional solution for the period $[0, 2^k - 1 + T]$:

(i) define $x_{b_i, 2^k - 1 + i} = 1$ for each job of type-B, and

(ii) define $x_{a_i,t} = 1$ for each type-A job a_i and each time $t \in [r_{a_i}, 2^k)$, $x_{a_i,t} = 1/2$ for all times $t \in [2^k, 2^k + T)$, and $x_{a_i,t} = 1$ for $[2^k + T, C^*_{a_i}]$, where $C^*_{a_i}$ is its completion time in SRPT.

The total LP value is at most approximately $O(2^k) + (T \cdot (k/2 + 1))$, so as long as the solutions is feasible, we get a gap of at least $2 - \varepsilon$ by setting k and T large enough.

To show feasibility, the only interesting times are $t = 2^k - 1 + i$, and some contiguous interval of jobs $S = \{a_\ell, a_{\ell+1}, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{i-1}, b_i\}$. The total length of this interval is $2^{k-\ell} + i - 1$, whereas the total processing time of jobs in S is $(2 \cdot 2^{k-\ell}) + i - 1$; hence excess of this interval is $2^{k-\ell}$, which is the length of the longest job in this set; hence $\min(p_j, e(S,t)) = p_j$ for all jobs in S. Now the LP constraints simply want that

$$\sum_{j \in S/a_l} p_j x_{j,t} \geq 2^{k-\ell},$$

which is satisfied by setting $x_{j,t} = 1/2$ for all type-A jobs. ∎

8 Summary and Closing Remarks. In this work, we considered the problem of minimizing the total flow time in the robust scheduling model; here we are only given estimates $\widehat{p}_j$ for the true processing times p_j. We give a new, simple algorithm Balance for this setting. If the distortion of the predictions is μ, the algorithm is $O(\mu)$-competitive, which matches the lower bound of [3], and improves on the previous result of $O(\mu \log \mu)$.

In addition to getting a quantitative improvement, our algorithm Balance is conceptually simpler; we feel it abstracts out the essential ideas behind the good performance of the algorithms presented in [2, 3], while avoiding some of the complexity. Our analysis proceeds by dual-fitting, using a strong knapsack-cover-based LP; along the way we introduce the idea of *reduced instances*, where we modify input instances to get more structure, allowing us to set appropriate duals with large objective function value.

Our work completes the story for minimizing flow time robustly, and suggests many other directions of investigation. For example, given the simplicity of our algorithm, can we extend it to get robust algorithms for weighted flow time? Can the LP-based dual-fitting analysis, and the reduction property, be useful for more general models of robust scheduling, e.g., when the distortion in the predicted sizes follows some distribution?

References

[1] C. Argue, A. Frieze, A. Gupta, and C. Seiler, *Learning from a sample in online algorithms*, in NeurIPS, Dec 2022, https://openreview.net/forum?id=KMaI40_UaGw.

[2] Y. Azar, S. Leonardi, and N. Touitou, *Flow time scheduling with uncertain processing time*, in STOC, ACM, 2021, pp. 1070–1080, https://doi.org/10.1145/3406325.3451023, https://doi.org/10.1145/3406325.3451023.

[3] Y. Azar, S. Leonardi, and N. Touitou, *Distortion-oblivious algorithms for minimizing flow time*, in SODA, SIAM, 2022, pp. 252–274, https://doi.org/10.1137/1.9781611977073.13, https://doi.org/10.1137/1.9781611977073.13.

[4] Y. Azar and N. Touitou, *Improved online algorithm for weighted flow time*, in FOCS, IEEE Computer Society, 2018, pp. 427–437, https://doi.org/10.1109/FOCS.2018.00048, https://doi.org/10.1109/FOCS.2018.00048.

[5] É. Bamas, A. Maggiori, L. Rohwedder, and O. Svensson, *Learning augmented energy minimization via speed scaling*, in Advances in Neural Information Processing Systems 33: Annual

Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[6] N. Bansal and H. Chan, *Weighted flow time does not admit o(1)-competitive algorithms*, in SODA, SIAM, 2009, pp. 1238–1244, https://doi.org/10.1137/1.9781611973068.134, https://doi.org/10.1137/1.9781611973068.134.

[7] N. Bansal and K. Dhamdhere, *Minimizing weighted flow time*, ACM Trans. Algorithms, 3 (2007), p. 39, https://doi.org/10.1145/1290672.1290676, https://doi.org/10.1145/1290672.1290676.

[8] N. Bansal and K. Pruhs, *The geometry of scheduling*, SIAM Journal on Computing, 43 (2014), pp. 1684–1698.

[9] L. Becchetti and S. Leonardi, *Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines*, J. ACM, 51 (2004), pp. 517–539, https://doi.org/10.1145/1008731.1008732, https://doi.org/10.1145/1008731.1008732.

[10] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs, *Semi-clairvoyant scheduling*, Theor. Comput. Sci., 324 (2004), pp. 325–335, https://doi.org/10.1016/J.TCS.2004.05.023, https://doi.org/10.1016/j.tcs.2004.05.023.

[11] R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips, *Strengthening integrality gaps for capacitated network design and covering problems*, tech. report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia ..., 1999.

[12] C. Chekuri, S. Khanna, and A. Zhu, *Algorithms for minimizing weighted flow time*, in STOC, ACM, 2001, pp. 84–93, https://doi.org/10.1145/380752.380778, https://doi.org/10.1145/380752.380778.

[13] S. Im, R. Kumar, M. M. Qaem, and M. Purohit, *Non-clairvoyant scheduling with predictions*, in Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), ACM, 2021, pp. 285–294, https://doi.org/10.1145/3409964.3461790, https://doi.org/10.1145/3409964.3461790.

[14] S. Lattanzi, T. Lavastida, B. Moseley, and S. Vassilvitskii, *Online scheduling via learned weights*, in Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2020, pp. 185–197, https://doi.org/10.1137/1.9781611975994.12, https://doi.org/10.1137/1.9781611975994.12.

[15] S. Li and J. Xian, *Online unrelated machine load balancing with predictions revisited*, in Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, M. Meila and T. Zhang, eds., vol. 139 of Proceedings of Machine Learning Research, PMLR, 2021, pp. 6523–6532, http://proceedings.mlr.press/v139/li21w.html.

[16] R. Motwani, S. J. Phillips, and E. Torng, *Non-clairvoyant scheduling*, Theor. Comput. Sci., 130 (1994), pp. 17–47, https://doi.org/10.1016/0304-3975(94)90151-1, https://doi.org/10.1016/0304-3975(94)90151-1.

[17] M. Purohit, Z. Svitkina, and R. Kumar, *Improving online algorithms via ml predictions*, in Advances in Neural Information Processing Systems (NeurIPS), vol. 31, 2018, pp. 9661–9670, https://proceedings.neurips.cc/paper/2018/hash/1e3f63f75a0b32ec3a0ae784b6e11970-Abstract.html.

[18] L. Schrage, *Letter to the editor - A proof of the optimality of the shortest remaining processing time discipline*, Oper. Res., 16 (1968), pp. 687–690, https://doi.org/10.1287/OPRE.16.3.687, https://doi.org/10.1287/opre.16.3.687.

[19] W. E. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly, 3 (1956), pp. 59–66.