Abstract
We consider the classic online scheduling problem of minimizing the total weighted flow time on a single machine with preemptions. Here, each job $j$ has an arbitrary arrival time $r_j$, weight $w_j$ and size $p_j$, and given a schedule its flow time is defined as the duration of time since its arrival until it completes its service requirement. The first non-trivial algorithms with poly-logarithmic competitive ratio for this problem were obtained relatively recently, and it was widely believed that the problem admits a constant factor competitive algorithm. In this paper, we show an $\omega(1)$ lower bound on the competitive ratio of any deterministic online algorithm. Our result is based on a novel gap amplification technique for online algorithms. Starting with a trivial lower bound of 1, we give a procedure to improve the lower bound sequentially, while ensuring at each step that the size of the instance increases relatively modestly.

1 Introduction
We consider the classic problem of scheduling a collection of dynamically arriving jobs so as to minimize the total weighted flow time. Given a schedule, the flow time of a job is defined as the amount of time its spends in the system until it completes its service requirement. Flow time is also referred to as the response time or the sojourn time, and is perhaps the most natural and widely used measure of system performance. In settings where jobs might have varying degrees of importance, it is usually more meaningful to consider the average (or equivalently total) weighted flow time.

In this paper, we consider the problem in an online setting on a single machine with preemptions. In the online setting, the jobs arrive arbitrarily over time and the scheduler learns of job only when it is released. Each job $j$ is determined by its release time $r_j$, processing time $p_j$, which is the amount of time the processor must spend on $j$ until it finishes, and weight $w_j$. The flow time of $j$ is defined as $f_j = c_j - r_j$ where $c_j$ is the completion time of $j$, that is, the earliest time it receives $p_j$ units of service, and the total weighted flow time is $\sum_j w_j f_j$. In the preemptive setting, a job can be interrupted arbitrarily and resumed later from the point of preemption. Most computer systems, such as web servers and operating systems, allow preemption and it is well known, both in theory and practice, that preemption is critical in order to provide a reasonable performance [12].

Minimizing unweighted flow time is quite well understood. It is a folklore result, that the online algorithm Shortest Remaining Processing Time, SRPT, is optimum for minimizing the total unweighted flow time on a single machine. Relatively recently, Leonardi and Raz [14] analyzed SRPT for multiple machines and showed that it has a competitive ratio of $O(\min(\log(n/m), \log \Delta))$. Here $n$ is the total number of jobs, $m$ is the number of machines and $\Delta$ is the ratio of the maximum to minimum job size. They also showed that no online algorithm can have a better competitive ratio up to constant factors. Subsequently, more restricted algorithms with similar guarantees, such that those that do not allow jobs to be migrated among machines [3] or those that dispatch a job to a machine immediately upon arrival [2] have also been obtained.

However, despite much interest not much was known about weighted flow time until recently. The first non-trivial guarantee was obtained by Chekuri, Khanna and Zhu who gave an $O(\log^2 \Delta)$ competitive algorithm for the problem [11]. Strictly speaking, [11] refer to their algorithm as semi-online as it requires an apriori knowledge of $P$. They also showed a lower bound on 1.618 on the competitive ratio of any online algorithm. Subsequently, Bansal and Dhamdhere [5] gave another algorithm with a competitive ratio of $O(\log W)$ where $W$ is the ratio to the maximum to minimum job weight. Given the lack of any non-trivial lower bounds and the incomparable guarantees of [11] and [5], it was widely believed that there should be an $O(1)$ competitive algorithm for the problem [11, 5, 17, 18]. In fact, in a recent survey, [17] (page 4) calls this the most important prob-

---

IBM T. J. Watson Research Center, nikhil@us.ibm.com
†Max-Planck-Institut für Informatik, hlchan@mpi-inf.mpg.de
lem in online scheduling.

In this paper, we show that in fact no algorithm can be c-competitive for any constant c. In particular, we show that any algorithm must have a competitive ratio of \( \Omega(\min\{ \sqrt{\frac{\log W}{\log \log W}}, \sqrt{\frac{\log \log n}{\log \log \log n}} \}) \). We use a gap amplification technique which works roughly as follows: For any \( c \geq 1 \), given an adversary which leads to a lower bound of \( c \) on the competitive ratio, we show how to use it to construct another adversary which leads to a lower bound of \( (c + \Omega(1)) \) on the competitive ratio. Furthermore, the size and weight of the jobs used by the new adversarial strategy are not much larger than those used by the original adversary. By applying this construction repeatedly we obtain the claimed lower bounds. We now give some more intuition about the weighted flow time problem.

**Preliminaries.** It is easily seen that the total weighted flow time for a schedule is identical to the sum over each time step, of the total weight of unfinished jobs at that time\(^4\). Thus to show than an algorithm is c-competitive, it suffices to show that the total weight of unfinished jobs at any time \( t \) under the algorithm is at most \( c \) times that under any offline algorithm. Interestingly, it turns out that this condition is also necessary, and is referred to as local-competitiveness [11, 7]. The reason is that if the weight of unfinished jobs under online algorithm is more than \( c \) times that under the offline algorithm at some time \( t \), then starting at time \( t \), the adversary can release a continuous stream of very small jobs with very high relative weight. This forces both the online and offline algorithms to work on this stream. By making the stream arbitrarily long, the total weighted flow can be made arbitrarily close to \( c \) times that of the offline. A formal argument can be found in proof of Theorem 2.1 below.

The above observation makes the problem much cleaner to argue about. In particular, designing a c-competitive algorithm is equivalent to designing an online algorithm that for any time \( t \), ensures that its unfinished weight is at most \( c \) times that under any possible schedule. Intuitively, to keep the unfinished weight low, any reasonable algorithm must work as much as possible on jobs with high weight to size ratio. However, it is instructive to see why a natural algorithm such as Highest Density First, HDF, performs poorly. HDF, at time works on the job with the highest density, where density of a job is its weight to size ratio. Suppose \( n \) jobs of size 1 and weight 2 each and one job of size \( W \) and weight \( W \) arrive at \( t = 0 \). HDF first works on the small jobs, and then works on the larger job. However, consider scenario at time \( t = n + W - 1 \). At this time, HDF still has \( W \) unfinished weight while the offline algorithm can finish all but one small job by this time, and have only one unfinished weight of 2. This example shows that there is a trade off between working on high density jobs, and at the same time favoring jobs with a relatively low density job but high weight. The algorithms of [11, 5] carefully balance this trade off. At a high level, they try to work on high density jobs as much as possible until there is a low density job whose weight is comparable to the total weight of high density jobs.

The lower bound of 1.618 due to [11], is also based on this trade off between high density and high weight jobs. In order to improve this lower bound to \( \omega(1) \) new ideas are needed. First we need to extend the above trade off to multiple weight classes and size classes, and extend the above trade off to these classes. This is necessary, since the algorithms [11, 5] give good guarantees if the number of weight classes or sizes classes are not too many. To do this, we give a recursive construction, where the number of classes is gradually increased. The crucial step in doing this however, is to ensure that while the adversary introduces jobs in new classes, it should not be possible for the online algorithm to finish work in previous classes, thereby eliminating the gap constructed thus far.

**Related work.** In general, flow time related problems have been studied extensively in the last few years, and we refer the reader to [18] for a relatively recent survey. Here we only state the results directly related to weighted flow time. Becchetti et al. [7] considered the resource augmentation analysis, where the online algorithm is allowed a slightly faster processor than the offline adversary, and showed that for any \( \epsilon > 0 \), HDF is a \((1 + \epsilon)\)-speed, \((1 + 1/\epsilon)\)-competitive for minimizing total weighted flow time. Later [5] gave an algorithm with an identical guarantee in the non-clairvoyant setting. The non-clairvoyant setting was first introduced by [15], and here the online algorithm does not know the size of the job until the job meets its service requirement. In the offline case, the problem was shown to be NP-hard by Lenstra et al. [13]. Chekuri and Khanna [10] gave a quasi-polynomial time approximation scheme. This was later extended to a constant number of machines [4].

A special case when the weight of job is inversely proportional to its size, i.e. \( w_j = 1/p_j \), has been extensively studied. This measure is referred to as stretch,
also known slowdown or normalized response time, and models the fact that users with longer jobs might be willing to wait more for their service. Muthukrishnan et al. [16] showed that SRPT is 2 competitive for minimizing the total stretch on a single machine and O(1) competitive for multiple machines. The multiple machine guarantee was later improved by [11]. Stretch has also been studied in the offline case and an approximation scheme is known for single machine [10, 8].

The \( \ell_p \) norms of flow time and stretch have also been studied [6, 9], which is some sense can be viewed as weighted flow time, where the weight of a job is dynamic and proportional to \( f_j^{-1} \) or \( f_j^{-1}/p_j^2 \). Another related measure is that of minimizing the total weighted completion time which has been extensively studied and culminated in various approximation schemes [1]. Note that completion time is identical to flow time in terms of finding an optimum solution, but substantially easier in terms of approximation.

2 Proof of the Lower Bound
In this section, we show that the competitive ratio of any algorithm is \( \Omega(\min\{\sqrt{\log W}, \sqrt{\log \log \Delta}\}) \).

In section 2.1, we begin by describing the notion of a \( c \)-adversary which is a key building block in our construction. In section 2.2 we describe the procedure to construct a \( (c+1/4) \)-adversary by combining various \( c \)-adversaries, and prove its correctness. Finally, in section 2.3 we show how the results in the previous section together with local-competitiveness imply the desired lower bound.

2.1 The building block
We define a special type of adversary called \( c \)-adversary with parameters \( (P, W, b) \).

**Definition 2.1.** Let \( c, P, W, b \geq 1 \) be some real numbers. A strategy \( S \) to release jobs is a \( c \)-adversary with parameters \( (P, W, b) \) if it satisfies the following properties.

1. Each job has size at least 1. The total size of all jobs is at most \( P \).
2. Each job has weight at least 1. The total weight of all jobs is at most \( W \).
3. Each job has release time at least 0. For any algorithm \( ALG \), there exists a time \( t \leq P \) such that the following two conditions hold.
   - Let \( I \) be the set of jobs released by \( S \) during \([0, t]\). Let \( w_{ALG}(t, I, b) \) be the total weight of unfinished jobs in \( ALG \) at time \( t \) with remaining size at least \( b \). Then \( w_{ALG}(t, I, b) \geq W/(4e)^{4c} \).
   - There exists an algorithm \( OPT \) to process \( I \) during \([0, t]\) such that the total weight of unfinished jobs in \( OPT \) at time \( t \) is \( w_{OPT}(t, I) \) and \( w_{OPT}(t, I) \leq \frac{1}{c} w_{ALG}(t, I, b) \).

To ease the discussion, we denote the above three properties as Property 1, 2 and 3. We say that the algorithm \( ALG \) breaks down with respect to \( S \) at time \( t \), where \( t \) is the time specified by Property 3. We also call the corresponding algorithm \( OPT \) the optimal algorithm. As an example, we can see that simple 1-adversary exists.

**Observation 1.** The strategy \( S \) to release a single job of size 1 and weight 1 at time 0 is a 1-adversary with parameters \((1, 1, 1)\).

**Proof.** Property 1 and 2 are obviously true. For Property 3, we see that any algorithm breaks down at time \( t = 0 \).

When the algorithm \( ALG \) breaks down, the total weight of unfinished jobs in \( ALG \) will remain at least \( c \) times that of \( OPT \) until \( ALG \) processes at least \( b \) unit of work. The reason is that \( w_{ALG}(t, I, b) \) consists of jobs with remaining size at least \( b \) and their total remaining weight will not decrease until \( ALG \) completes some of these jobs. Thus we say that \( ALG \) has \( b \) units of pending work once it breaks down. The pending work will be useful later in the recursive construction to ensure that online algorithm cannot complete too many jobs from this set at some future time. We observe that the amount of pending work can be increased to any amount by scaling up the size of all jobs, as stated in the following observation.

**Observation 2.** Assume there exists a \( c \)-adversary with parameters \( (P, W, b) \). We can scale up the size of each job by \( \alpha \) times, for any \( \alpha \geq 1 \), to obtain a \( c \)-adversary with parameters \( (\alpha P, \alpha W, \alpha b) \).

2.2 Constructing a \( (c + \frac{1}{4}) \)-adversary
Assume that there exists a \( c \)-adversary with parameters \( (P, W, 1) \). This section shows how to use the \( c \)-adversary to construct a \( (c + \frac{1}{4}) \)-adversary with suitable parameters.

**Lemma 2.1.** Let \( c \geq 1 \) be a multiple of \( \frac{1}{4} \). Assume there exists a \( c \)-adversary with parameters \( (P, W, 1) \). Then we can construct a \( (c + \frac{1}{4}) \)-adversary with parameters \( ((4P)^k, 2kW, 1) \), where \( k = 4(c+1)/(4c)^{4c} \).

**Proof.** To ease the discussion, for any subset \( I \) of jobs and algorithm \( A \), we use \( w_A(t, I, b) \) to denote the total weight of unfinished jobs in \( A \) at time \( t \) that have remaining size at least \( b \). We use \( w_A(t, I) \) to denote
Let ALG be any algorithm. Note that the strategy $S$ will be adaptive and its operation will depend on the execution of ALG. $S$ consists of three steps.

1. At time 0, $S$ invokes a $c$-adversary $S_1$ with parameters $((4P)^{k-1}P, W, (4P)^{k-1})$. Such an adversary exists by applying Observation 2 with $a = (4P)^{k-1}$. Whenever $S_1$ releases a job, $S$ releases the same job with same size and weight. Let $t_1$ be the time that ALG breaks down with respect to $S_1$ and let $I_1$ be the set of jobs released by $S_1$. Let OPT be the optimal algorithm to process $I_1$. Note that by the definition of a $c$-adversary, we have

$$w_{\text{ALG}}(t_1, I_1, (4P)^{k-1}) \geq \frac{W}{(4c)^{k-1c}} \quad \text{and} \quad w_{\text{ALG}}(t_1, I_1, (4P)^{k-1}) \geq c \cdot w_{\text{OPT}}(t_1, I_1)$$

Recall that $w_{\text{ALG}}(t_1, I_1, (4P)^{k-1})$ is the total weight of unfinished jobs at time $t$ that have a remaining size of at least $(4P)^{k-1}$. In particular this implies that ALG has $(4P)^{k-1}$ units of pending work belonging to $I_1$ at time $t_1$. $S$ stops the operation of $S_1$ starting at time $t_1$.

2. We repeat the above process for $i = 2, \ldots, k$ as follows. Let $t_i-1$ be the time when ALG first breaks down with respect to $S_{i-1}$. The strategy $S$ immediately invokes a $c$-adversary $S_i$ with parameters $((4P)^{k-1}P, W, (4P)^{k-1})$ at time $t_i-1$. Let $t_i \geq t_i-1$ be the time that ALG breaks down with respect to $S_i$. Note that $t_i \leq t_i-1 + (4P)^{k-1}P$ by Property 3. Let $I_i$ be the set of jobs released by $S_i$ and OPT be the optimal algorithm to process $I_i$. We have

$$w_{\text{ALG}}(t_i, I_i, (4P)^{k-1}) \geq \frac{W}{(4c)^{k-1c}} \quad \text{and} \quad w_{\text{ALG}}(t_i, I_i, (4P)^{k-1}) \geq c \cdot w_{\text{OPT}}(t_i, I_i)$$

$S$ stops the operation of $S_i$ at time $t_i$. Note that the last $c$-adversary invoked by $S$ is $S_k$, which is a $c$-adversary with parameters $(P, W, 1)$.

3. Let Size denote the total remaining size of all unfinished jobs in ALG at time $t_k$. Let Weight denote $\sum_{i=1}^{k} w_{\text{ALG}}(t_i, I_i, (4P)^{k-1})$. Note that for each $i$ such that $1 \leq i \leq k$, by Property 3 we have that $w_{\text{ALG}}(t_i, I_i, (4P)^{k-1}) \geq W/(4c)^{k-1c}$, and hence

$$\text{Weight} \geq kW/(4c)^{k-1c}$$

At time $t_k$, $S$ releases a final job $J$ such that the $p(J) = \text{Size}$ and $w(J) = \frac{1}{y}\text{Weight}$ for some $y > 1$, where we will set $y = c + 1$ later. This completes the description of $S$. We let $I = I_1 \cup \ldots \cup I_k \cup \{J\}$.

We now show that $S$ defined above is a $(c + \frac{1}{4})$-adversary with parameters $((4P)^{k-1}2kW, 1)$, where $k = 4(c+1)(4c)^{k-1}$. We first observe that all the jobs released by $S$ have size at least 1. The total size of jobs released by $S_i$ is at most $(4P)^{k-1}P$. The size of $J$ is at most the total size of all the previously released jobs. Thus, the total size of all jobs released by $S$ is at most

$$(4P)^{k-1}P + (4P)^{k-2}P + \ldots + P) \times 2 \leq (4P)^k$$

Similarly, all jobs released by $S$ have weight at least 1. The total weight of all jobs is at most

$$W \times k \times 2 = 2kW$$

Our goal is to show that ALG breaks down with respect to $S$ at some time $t \leq (4P)^k$. More precisely, we need to show that there is a time $t$ such that

$$w_{\text{ALG}}(t, I, 1) \geq \frac{2kW}{(4c+1)^{k-1c+1}} \quad \text{and} \quad w_{\text{OPT}}(t, I) \leq \frac{1}{c+1/4}w_{\text{ALG}}(t, I, 1)$$

Let $t_a = t_k + p(J)$. Let $p_a(J, t_a)$ be the remaining size of $J$ under ALG at time $t_a$. We consider two cases depending on whether $p_a(J, t_a) \leq 1$ or $p_a(J, t_a) > 1$.

Case 1 $p_a(J, t_a) \leq 1$. We show that ALG breaks down at $t_a$ in this case. Since $p_a(J, t_a) \leq 1$ and $t_a = t_k + p(J)$, during $[t_k, t_a]$, ALG processes at most 1 units of pending work for jobs in $I_1, \ldots, I_k$. Let $J'$ be a job in $I_k$ with remaining size at least $(4P)^{k-1}$ at time $t_a$. During $[t_k, t_a]$, $J'$ is processed for at most $(t_a - t_k)$ units of time. Thus the remaining size of $J'$ at time $t_a$ is at least $(4P)^{k-1} - (\sum_{i=k}^{k} (4P)^{k-1}P - 1)$, which is at least 1 for $i \leq k - 1$. Thus, for each $I_k$, with $i \leq k - 1$, we have that $w_{\text{ALG}}(t_a, I_k, 1) \geq w_{\text{ALG}}(t_a, I_k, (4P)^{k-1})$. Therefore

$$w_{\text{ALG}}(t_a, I, 1) \geq w_{\text{ALG}}(t_a, I_1, (4P)^{k-1}) + \ldots + w_{\text{ALG}}(t_a, I_k, (4P)^{k-1}) = \text{Weight} - w_{\text{ALG}}(t_k, I_k, 1) \geq \text{Weight} - \frac{(4c)^{k-1c}}{k}\text{Weight}$$

The last inequality follows from the fact that $w_{\text{ALG}}(t_k, I_k, 1) \leq W$ and that $\text{Weight} \geq kW/(4c)^{k-1c}$. Since $k = 4(c+1)(4c)^{k-1}$, we have that $w_{\text{ALG}}(t_a, I, 1)$ is at least $(1 - 4/(c+1))\text{Weight}$. We want to show that
it is at least \((2kW)/(4c + 1)^{4c+1}\). This is true as \(1 - \frac{1}{(4c + 1)}\) Weight is at least

\[
(2.3) \quad \frac{1}{c+1} \text{Weight} \geq \frac{kW}{2c(4c)^{4c}} \geq \frac{2kW}{(4c + 1)^{4c+1}}.
\]

The first inequality above follows from (2.1), and this proves the first part of (2.2).

To show the second part, we observe that an algorithm OPT can complete all jobs in \(I_t\) remaining at time \(t_k\) during \([t_k, t_0]\) and leave \(J\) untouched. Therefore, the total weight of jobs remaining in OPT at \(t_a\) is \(w_{OPT}(t_a, I) = w(J)\). Thus,

\[
w_{ALG}(t_a, I, 1) \geq \left(1 - \frac{(4c)^{4c}}{k}\right) \cdot \text{Weight} = \left(1 - \frac{(4c)^{4c}}{k}\right) \cdot w(J) = y \left(1 - \frac{(4c)^{4c}}{k}\right) w_{OPT}(t_a, I)
\]

By the choice of \(y = c + 1\) and \(k = 4(c + 1)(4c)^{4c}\), we have that

\[
\left(y - \frac{(4c)^{4c}}{k}\right) = c + \frac{3}{4} > c + \frac{1}{4}.
\]

This implies that (2.2) holds, and that ALG breaks down at \(t_a\).

**Case 2** \(p_a(J, t_a) > 1\). Let \(t_{b1}\) be the earliest time that \(p_a(J, t_{b1}) = 1\) and let \(t_{b2} = t_k + 2p(J)\). Observe that \(t_{b1} > t_k + p(J)\), otherwise we would be in case above. Let \(t_b = \min\{t_{b1}, t_{b2}\}\). We will show that ALG breaks down at \(t_b\) in this case, i.e. (2.2) holds at \(t = t_b\). Note that \(t_b \leq ((4P)^{k-1}P + (4P)^{k-2}P + \ldots + P) \times 3 \leq (4P)^{k}\).

We first consider the easy case of \(t_b = t_{b2}\). The remaining size of \(J\) under ALG at time \(t_b\) is at least \(1\) and thus \(w_{ALG}(t_b, I, 1) \geq w(J) = \frac{1}{2}\)Weight. By the choice that \(y = c + 1\) and (2.3), we have \(w_{ALG}(t_b, I, 1) \geq 2kW/(4c + 1)^{4c+1}\). An algorithm OPT can complete all jobs by time \(t_b\), so \(w_{OPT}(t_b, I) = 0\). These two facts imply that ALG breaks down at \(t_b\).

We henceforth assume that \(t_b = t_{b1}\). By definition of \(t_{b1}\) and (2.3) we have \(w_{ALG}(t_b, I, 1) \geq w(J) \geq 2kW/(4c + 1)^{4c+1}\). Hence the first part of (2.2) is satisfied. It remains to show the second part of (2.2).

Let \(x\) be the smallest integer such that during \([t_k, t_b]\), the instance \(I_x\) has been processed by ALG for at least \(2 \sum_{j=x+1}^{k} (4P)^{k-j}P\) units of time. Note that \(x\) exists as when \(x = k, 2 \sum_{j=x+1}^{k} (4P)^{k-j}P = 0\). Consider any \(i \leq x - 1\) and let \(J'\) be a job in \(I_i\) with remaining size at least \((4P)^{k-i}\) at time \(t_i\). During \([t_i, t_b]\), \(J'\) is processed for at most \((t_b - t_i) + 2 \sum_{j=i+1}^{k} (4P)^{k-j}P\) units of time. Since \((t_b - t_i) \leq \sum_{j=i+1}^{k} (4P)^{k-j}P\), the remaining size of \(J'\) at time \(t_b\) is at least

\[
(4P)^{k-i} - 3 \sum_{j=i+1}^{k} (4P)^{k-j}P \geq P^{k-i}(4^{k-i} - 3 \sum_{j=i+1}^{k} 4^{k-j}) \geq 1
\]

Thus, for each \(I_i\) with \(i \leq x - 1\), we have that

\[
w_{ALG}(t_b, I_i, 1) \geq w_{ALG}(t_i, I_i, (4P)^{k-i}).
\]

We can lower bound \(w_{ALG}(t_a, I, 1)\) as

\[
(2.4) \quad w_{ALG}(t_a, I, 1) \geq w_{ALG}(t_1, I_1, (4P)^{k-1}) + \ldots + w_{ALG}(t_x, I_{x-1}, (4P)^{k-x+1}) + w(J)
\]

\[
\geq \sum_{i=1}^{x} w_{ALG}(t_i, I_i, (4P)^{k-i}) - W + w(J)
\]

\[
\geq \sum_{i=1}^{x} w_{ALG}(t_i, I_i, (4P)^{k-i}) + \left(1 - \frac{(4c)^{4c}}{k}\right) \text{Weight}
\]

The second inequality follows as the total weight of jobs in \(I_i\) is at most \(W\), and the third inequality follows from (2.1).

We now claim that the OPT algorithm can instead finish \(J\) together with all the jobs in \(I_{x+1}, \ldots, I_k\) by time \(t_b\). Consider an algorithm that follows OPT for \(i = 1, \ldots, k\) during \([0, t_b]\). First, for \(x = k\), the claim follows as OPT can complete \(J\) during \([t_k, t_b]\) (recall that \(t_{b1} > p(J) + t_k\)). Second, suppose that \(x < k\). By considering the amount of work done by ALG during \([t_k, t_b]\), we have that

\[
t_b - t_k \geq p(J) - 1 + 2 \sum_{i=x+1}^{k} (4P)^{k-x}P \geq p(J) + \sum_{i=x+1}^{k} (4P)^{k-x}P
\]

Thus OPT can complete \(J\) and all jobs remaining in \(I_{x+1}, \ldots, I_k\). In both cases, the total weight of unfinished jobs in OPT at time \(t_b\) is

\[
w_{OPT}(t_b, I) \leq w_{OPT}(t_1, I_1) + \ldots + w_{OPT}(t_x, I_x) \leq \frac{1}{c} \sum_{i=1}^{x} w_{ALG}(t_i, I_i, (4P)^{k-i})
\]
Thus, together with (2.4) it follows that

\[ w_{ALG}(t_b, I, 1) \geq c \left( 1 + \frac{1}{y} - \frac{(4c)^k}{k} \right) w_{OPT}(t_b, I) \]

By the choice of \( y = c + 1 \) and \( k = 4(c + 1)(4c)^k \), we have that \( c + 1 \left( \frac{1}{y} - \frac{(4c)^k}{k} \right) = c + 1 - \frac{3c}{4(c+1)} > c + \frac{3}{8} > c + \frac{1}{4} \). Thus we have shown that ALG breaks down at \( t_b \).

### 2.3 The lower bounds on the competitive ratio

In this section, we show how the \( c \)-adversary together with local-competitiveness gives lower bounds for the competitive ratio of weighted flow time. We first compute the parameters required in Lemma 2.1 to construct a \( c \)-adversary for any \( c \geq 1 \).

**Lemma 2.2.** For any \( c \geq 1 \) that is a multiple of \( \frac{1}{4} \), there exists a \( c \)-adversary with parameters \((4f(c), f(c), 1)\), where \( f(c) = (4c)(4c)^2 \).

**Proof.** We first observe that if there exists a \( c \)-adversary \( S \) with parameters \((P, W, 1)\), then there exists a \( c \)-adversary with parameters \((P', W', 1)\) for any \( P' \geq P \) and \( W' \geq W \). The reason is because, \( S \) itself is a \( c \)-adversary with parameters \((P', W, 1)\). Next, we can scale up the weight of each job by \( W'/W \). The total weight of all jobs released is at most \( W' \) and when an algorithm ALG breaks down, the remaining weight for jobs with size at least 1 is \( W'/4c^k \).

We now prove the lemma by induction on the value of \( c \). When \( c = 1 \), Observation 1 states that there is a 1-adversary with parameters \((1, 1, 1)\), so it implies a 1-adversary with parameters \((4f(1), f(1), 1)\). Assume the lemma is true for some \( c \geq 1 \) that is a multiple of \( \frac{1}{4} \). Consider \( c + \frac{1}{4} \). By Lemma 2.1, there exist a \((c + \frac{1}{4})\)-adversary with parameters \((4f(c))\), \(2kf(c), 1\), where \( k = 4(c + 1)(4c)^k \). Note that

\[
2kf(c) = 2 \cdot 4(c + 1)(4c)^{4c} \cdot (4c)^{4c^2} \\
\leq 4c \cdot (4c)^{4c+2} \cdot (4c)^{4c^2} \\
\leq (4c + 1)^{4c(1 + \frac{1}{4})} = f(c + \frac{1}{4})
\]

and

\[
(4f(c)) < 4^{2kf(c)} \leq 4f(c + \frac{1}{4})
\]

This gives a \((c + \frac{1}{4})\)-adversary with parameters \((4f(c + \frac{1}{4}), f(c + \frac{1}{4}), 1)\), which completes the induction step.

We are now ready to show the lower bounds on weighted flow time.

**Theorem 2.1.** Any deterministic online algorithm is \( \Omega(\min\{\sqrt{\frac{\log W}{\log \log W}}, \sqrt{\log \log \log \Delta}\}) \)-competitive for weighted flow time, where \( W \) is the maximum to minimum ratio of job weight and \( \Delta \) is the maximum to minimum ratio of job size.

**Proof.** Let ALG be any algorithm. For any \( c \geq 1 \) that is a multiple of \( \frac{1}{4} \), we can invoke a \( c \)-adversary \( S \) with parameters \((4f(c), f(c), 1)\), where \( f(c) = (4c)(4c)^2 \). By definition, there exists a time \( t \) such that ALG breaks down. That is, let \( I \) be the set of jobs released by \( S \) during \([0, t]\) and let \( OPT \) be the optimal algorithm to process \( I \). We have \( w_{ALG}(t, I, 1) \geq c \cdot w_{OPT}(t, I) \). We stop the operation of \( S \) at time \( t \).

Let us use \( Y \) to denote \( w_{ALG}(t, I, 1) \). Starting at \( t \), we release a sequence of small jobs such that one small job is released at \( t + i/Y \) for each \( i = 0, 1, \ldots, N \). Each small job has size \( 1/Y \) and weight 1. Note that at any time during \([t, t + N/Y]\), the total weight of all unfinished jobs in \( ALG \) is at least \( Y \). On the other hand, OPT can process each small job as it is released and the total weight of all unfinished jobs in OPT is at most \( w_{OPT}(t, I) + 1 \). By making \( N \) arbitrarily large, the ratio of the total weighted flow time of \( ALG \) and \( OPT \) tends to

\[
\frac{w_{ALG}(t, I, 1)}{w_{OPT}(t, I) + 1} \geq c \cdot \frac{w_{OPT}(t, I)}{w_{OPT}(t, I) + 1} \geq \frac{c}{2}
\]

The ratio of the maximum to minimum job weight, i.e., \( W \), is at most \( f(c) = (4c)(4c)^2 \), and hence \( c = \Omega(\sqrt{\frac{\log W}{\log \log W}}) \). The ratio of the maximum to minimum job size, i.e., \( \Delta \), is at most \( 4f(c) \times Y \), because the smallest job size is \( 1/Y \). Note that \( Y \leq f(c) \), so \( \Delta \leq f(c) \cdot 4f(c) \leq 4f^2(c) \). Thus, \( 2f(c) \log 4 \geq \log \Delta \), and hence \( c = \Omega(\sqrt{\frac{\log \log \Delta}{\log \log \log \Delta}}) \), which is also a lower bound on the competitive ratio.

**Acknowledgments**

The authors would like to thank Kirk Pruhs and Stefano Leonardi for various illuminating discussions about weighted flow time over the last five years.

**References**