Lower bounds for on-line single-machine scheduling

Leah Epstein* Rob van Stee†

January 24, 2001

Abstract

The problem of scheduling jobs that arrive over time on a single machine is well-studied. We study the preemptive model and the model with restarts. We provide lower bounds for deterministic and randomized algorithms for several optimality criteria: weighted and unweighted total completion time, and weighted and unweighted total flow time.

1 Introduction

We consider on-line scheduling of n jobs on a single machine. The jobs arrive over time. Job J_j with processing time (or size) p_j is released (or arrives) at time r_j. This is also the time when it is revealed to the algorithm. The algorithm is required to assign each job to a machine. A job can be assigned at its arrival time or later. The algorithm may run at most one job on each machine at any time. In weighted problems, each job J_j is also given a positive weight w_j which represents its importance. We consider both deterministic and randomized algorithms.

In the standard scheduling model, a job which was assigned to a machine must be processed continuously to its completion. The preemptive scheduling model allows the algorithm to stop a running job and resume it later. A third model does not allow preemptions but allows restarts. In this case a running job may be stopped, but it has to be started from scratch when it is scheduled again. In this paper we focus on preemptive algorithms, and algorithms with restarts.

We consider two optimality criteria, each one is considered both in the weighted case and in the unweighted case. Let C_j be the completion time of J_j. The flow time F_j is the total time J_j exists in the system, i.e. $F_j = C_j - r_j$. This gives the following four criteria:

1. Minimizing the total completion time ($\sum C_j$).
2. Minimizing the total weighted completion time ($\sum w_j C_j$).
3. Minimizing the total flow time ($\sum F_j$).
4. Minimizing the total weighted flow time which is $\sum w_j F_j$.

*School of Computer and Media Sciences, The Interdisciplinary Center, P.O.B. 167, 46150 Herzliya, Israel. Epstein.Leah@idc.ac.il.

†Centre for Mathematics and Computer Science (CWI), Kruislaan 413, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands. Rob.van.Stee@cwi.nl.

Scheduling on a single machine simulates (e.g.) processing tasks on a serial computer. The flow time measure is used in applications where it is important to finish tasks fast, relative to their release time. On the other hand, the completion time measure is used when tasks need to be finished as fast as possible, relative to a starting time of the computer, with no connection to their arrival time. The weighted versions of the problems model cases where different jobs have different importance.

We study these problems in terms of competitive analysis. Thus we compare an (on-line) algorithm to an optimal off-line algorithm OPT that knows all jobs in advance, but cannot assign a job before its release time. If the on-line algorithm is allowed to preempt jobs, we assume that OPT can do this as well; however, in the other models we only consider non-preemptive off-line schedules. Let T_B be the cost of algorithm B. An algorithm A is $\mathcal{R}$-competitive if for every sequence $T_A \leq \mathcal{R} \cdot T_{OPT}$. The competitive ratio of an algorithm is the infimum value of $\mathcal{R}$ such that the algorithm is $\mathcal{R}$-competitive.

Known results There are several cases where the optimal schedule has a simple structure. The optimal schedule is the same both for (weighted) flow time and for (weighted) completion time, since the optimal costs differ by the constant $\sum r_i w_i$. For the weighted case, if all release times are zero (all jobs are released at the same time, hence the problem is always off-line), then an optimal off-line schedule is achieved by sorting the jobs by their ratios of size to weight (p_j/w_j), and processing them in non-decreasing order [13]. For the unweighted case, an optimal preemptive schedule can be built on-line by applying the SRPT algorithm. At all times, this algorithm processes the job with the smallest remaining processing time [9]. In an off-line environment, the simple structure makes the complexity of those problems polynomial. However, all weighted versions of the problem with general release times are strongly NP-hard [8], and so are the unweighted non-preemptive problems. (Naturally, nothing changes if restarts are allowed in an off-line environment, since all jobs are known in advance). Moreover, it is NP-hard to approximate the non-preemptive problem of minimizing total flow time to a factor of $O(n^{1/2-\varepsilon})$ [7]. This paper gives an off-line approximation of performance ratio $\Theta(\sqrt{n})$ for the same problem. Polynomial time approximation schemes for preemptive and non-preemptive weighted completion time, and for non-preemptive total completion time, were given recently by [1].

Total completion time: It is known that the best competitive ratio for the standard deterministic model is 2 [6, 14, 10], and $e/(e-1)$ for the standard randomized model [3, 15]. As mentioned earlier, it is also known that it is possible to get an optimal algorithm (i.e. achieve the competitive ratio 1) for the preemptive model. For the model with restarts, no better algorithms are known than the algorithms mentioned above, that do not use restarts. The best deterministic lower bound is 1.112 is due to Vestjens [16], no randomized lower bound is known.

Total weighted completion time: In this case the preemptive model is more interesting, since it is not clear if the problem can be solved optimally (competitive ratio 1) or almost optimally. The best deterministic preemptive algorithm has competitive ratio 2 [5, 11], and the best randomized preemptive algorithm has competitive ratio 4/3 [11]. Skutella [12] gave lower bounds of 31/30 for deterministic algorithms and 113/111 for randomized algorithms. For the standard model, the best algorithms are given in [5, 4] and are 2.415- *and* 1.686-*competitive* (deterministic and randomized, respectively). The best lower bounds are the same

	Restarts	
	deterministic	randomized
$\sum C_j$	1.2108	1.1068
$\sum F_j$	$\Omega(\sqrt{n})$	$\Omega(\sqrt{n})$

	Restarts		Preemptions	
	det.	rand.	det.	rand.
$\sum w_j C_j$	1.2232	1.1161	1.0730	1.0389
$\sum w_j F_j$	$\Omega(n)$	$\Omega(n)$	2	4/3

Table 1: The new lower bounds

as for the unweighted case. No results for algorithms with restarts (that do not follow from other results) are known.

Total flow time: The preemptive model is optimally solvable for this problem, hence the competitive ratio for preemptive scheduling is 1. However, the deterministic non-preemptive model is very hard to approximate, the best competitive ratio is $\Theta(n)$ and no better algorithms are known for any model. Stougie and Vestjens [15] gave a lower bound of $\Omega(\sqrt{n})$ for randomized non-preemptive algorithms, and [16] gave a lower bound of $\Omega(n^{1/4})$ for deterministic scheduling with restarts.

Total weighted flow time: It is easy to see that there can be no competitive (deterministic or randomized) algorithm for the standard model. No other results for weighted flow time are known.

Our results We give some new lower bounds, and improve some previously known lower bounds. Our results are presented in Table 1. Specifically, we improve the lower bounds of [16] for scheduling with restarts, both for total flow time and total completion time. We also improve the bounds of [12] for preemptive (deterministic and randomized) scheduling with the goal of minimizing the total weighted flow time. The existing lower bounds for these problems were very close to 1. The substantial improvements we show are due to new techniques we are using.

We begin by discussing several useful lower bounding methods, that are used in more than one proof, in Section 2. Section 3 contains our results on total (weighted) completion time, and Section 4 discusses the total (weighted) flow time measure.

2 Methods

To prove lower bounds for *randomized algorithms* we use the adaptation of Yao's theorem [17]. It states that a lower bound for the competitive ratio of deterministic algorithms on a fixed distribution on the input is also a lower bound for randomized algorithms and is given by $E(T_{ON}/T_{OPT})$, where T_{ON} is the cost of the on-line algorithm (see [2]).

A useful method for *weighted problems* is as follows. Assume that at time t, the on-line algorithm is left with one job of size $a \neq 0$ and weight b, and OPT has either completed all the jobs or it is left with a job of a smaller ratio of weight to size. We let k jobs of size ε arrive at times $t + (i-1)\varepsilon$ for $i = 1, \dots, k$. Each such job has weight $\frac{b}{a}\varepsilon$. Hence it does not matter for the total completion time or the total flow time in which order the on-line algorithm completes the jobs, and all the new jobs are interchangeable with the job of size a. Let $c = k\varepsilon$ and let ε tend to 0, keeping c constant. If OPT has no jobs left, and we are considering the total weighted completion time, then the extra cost of OPT is $tcb/a + c^2b/(2a)$ and the extra cost of the on-line algorithm is $tcb/a + cb + c^2b/(2a)$. The extra cost for other cases can be calculated similarly.

For algorithms that are allowed to *restart* jobs, it can be useful to let jobs of size 0 arrive at such a time that the on-line algorithm is forced to restart the job it is running, whereas OPT can run the jobs immediately due to its different schedule of the other jobs, or possibly delay them (in the case that more jobs arrive). This can be combined with a sequence of jobs with exponentially increasing sizes. By timing the arrival of the jobs, it is possible to force the on-line algorithm to restart every job in such a sequence (if it does not restart, we stop the sequence at that point).

3 Total completion time

3.1 Lower bounds for algorithms with restarts

We begin by showing bounds for the problem where all jobs have the same weight, first for deterministic algorithms and then for randomized algorithms.

Theorem 1 *Any deterministic algorithm for minimizing the total completion time on a single machine which is allowed to restart jobs, has a competitive ratio of at least* $\mathcal{R}_1 = 1.2102$.

Proof. Assume there is an algorithm A that has a competitive ratio of $\mathcal{R}_1 = 1.2102009$. A job of size 1 arrives at time 0. Since restarts are allowed, we may assume A starts it immediately. A sequence of jobs will now arrive in steps. In each step the online algorithm must restart. If it does not, the sequence stops at that point. Otherwise, the next item in the sequence arrives.

1. A job of size 0 arrives at time $x = 1/\mathcal{R}_1 - 1/2 \approx 0.326309$.
2. A job of size 0 arrives at time $y = 3/(2\mathcal{R}_1^2) - 1/(4\mathcal{R}_1) - 1/4 \approx 0.567603$.
3. Three jobs of size 0 arrive at time 1. If A does not restart, the implied competitive ratio is $(x + 5y + 4)/(x + y + 5) > \mathcal{R}_1$.

If A has restarted three times so far, we repeat the following for $i = 1, \ldots, 5$ or as long as A keeps restarting in step 5. OPT will complete the first six jobs by time 1 and pay 6 for them. Denote the first job that arrived (with size $x_0 = 1$) by J_0.

4. A job of size x_i arrives at the time OPT finishes J_{i-1}.
5. a_i jobs of size 0 arrive at the time OPT finishes J_i. (A is still executing J_i at this moment.)

If we fix $a_1, \ldots, a_5$ we can determine $x_1, \ldots, x_5$ so that if A does not restart on arrival of the a_i jobs of size 0, it pays exactly $\mathcal{R}_1$ times the optimal cost. Note that if A runs any J_i before J_{i-1}, it pays more than $\mathcal{R}_1$ times the optimal cost, and the sequence stops immediately without the arrival of a_i jobs of step 5 (when J_i arrives, the only job which is still not completed in the schedule of A is J_{i-1}).

i	1	2	3	4	5
a_i	3	2	2	2	1
x_i	2.13118	4.04404	8.33794	18.1366	36.2732

By fixing a_i ($i = 1, \ldots, 5$) as in this table, we can ensure that A pays more than $\mathcal{R}_1$ times the optimal cost for the entire sequence when the last job arrives. Since $x_5 = 2x_4$, A's costs are the same if it restarts the job of size x_5 for the last job and if it does not. □

Using a computer, we have been able to improve this bound slightly using $a_1 = 3$, $a_2 = \ldots = a_{45} = 2$, giving $\mathcal{R}_2 = 1.210883$. After J_{45} arrives, A has a cost of at least $\mathcal{R}_2$ times the optimal cost whether it restarts or not on arrival of the last 2 jobs of size 0.

Theorem 2 *Any randomized algorithm for minimizing the total completion time on a single machine which is allowed to restart jobs, has a competitive ratio of at least* $\mathcal{R}_3 = 114/103 \approx 1.1068$.

Proof. We use Yao's minimax principle [17] and consider a randomized adversary against a deterministic algorithm. Assume there exists an on-line algorithm A with a competitive ratio of $\mathcal{R}_3$. At time 0, a job of size 1 arrives. A will certainly start this job immediately since it is allowed to restart. At time 1/3, two jobs of size 0 arrive. With probability p, 10 more jobs of size 0 arrive at time 1, followed by 4 jobs of size 1 (either all these jobs arrive, or none of them).

If A restarts at time 1/3 and the jobs at time 1 do arrive, it has cost $30\frac{2}{3}$ independent of whether it restarts again. The optimal cost in case all jobs arrive is 27.

This implies that if A restarts at time 1/3, it has competitive ratio of at least $30\frac{2}{3}p/27 + (1-p)$; otherwise, it has competitive ratio $p+3(1-p)/2$. These ratios are equal for $p = 81/103$, and are then 114/103. This implies a competitive ratio of $\mathcal{R}_3$. □

The methods in these proofs can be adapted for the weighted problem to give somewhat higher bounds.

Theorem 3 *Any deterministic algorithm for minimizing the total weighted completion time on a single machine which is allowed to restart jobs, has a competitive ratio of at least* $\mathcal{R}_4 = 1.2232$.

Proof. Assume there is an algorithm A that has a competitive ratio of $\mathcal{R}_4 = 1.2232$. We use a somewhat similar structure as in Theorem 1. A job of size 1 and weight 1 arrives at time 0. Again we assume A starts it immediately. A sequence of jobs will now arrive in steps. In each step the online algorithm must restart. If it does not, the sequence stops at that point. Otherwise, the next item in the sequence arrives. We fix two weights $W = 0.79$ and $W' = 1.283$ to be used for the first part of the sequence.

1. A job of size 0 and weight W arrives at time $x = 1/\mathcal{R}_4 - 1/(W+1) \approx 0.258869$.
2. A job of size 0 and weight W' arrives at time $y = (\frac{xW+(1+x)(1+W')}{\mathcal{R}_4} - xW - 1)/(W'+1) \approx 0.574794$.
3. A job of size 0 and weight $Z = (xW\mathcal{R}_4 + yW'\mathcal{R}_4 + 2\mathcal{R}_4 - xW - yW' - y - 1)/(y + 1 - \mathcal{R}_4) \approx 3.07699$ arrive at time 1.

In all three cases, if A does not restart, the implied competitive ratio is $\mathcal{R}_4$. If A has restarted three times so far, we follow the procedure described below. OPT will complete the first four jobs by time 1 and will pay 6.14999 for them. The on-line cost for these jobs is $T_{ONL} = 6.01896$. Denote the first job that arrived (with size and weight 1) by J_0. Put $i = 1$. Let $x_0 = 1$ (denotes its size) $z_0 = 1$ (denotes its weight).

4. A job J_i of size $x_i = 2^i$ and weight z_i arrives at the time OPT finishes J_{i-1}, i.e. at time $2^i - 1$. If A completes J_{i-1} before J_i, go to step 5, otherwise go to step 6.

5. A job of size 0 and weight $w_i - z_i$ arrives at the time OPT finishes J_i (time $2^{i+1} - 1$). A is still executing J_i at this moment. If A does not restart, or $i = 5$, stop the sequence. Otherwise, increase i by 1 and go to Step 4.

6. k jobs of size ε and weight $\varepsilon z_{i-1}/x_{i-1}$ arrive at time $2^{i+1} - 1$, where $k\varepsilon = c_i$. (A can complete J_i no earlier than this.) The sequence stops.

In step 5, if it is possible to force a restart of J_i, then the cost of OPT will grow by $(2^{i+1} - 1)w_i$ whereas the on-line cost will grow by $(2^{i+1} - 1)(w_i - z_i) + (2^{i+1} + 2^i - 1)z_i$, hence the value z_i should be as large as possible. On the other hand, z_i should be small enough so that A has a competitive ratio of at least $\mathcal{R}_4$ if it runs J_i before J_{i-1} (as in Theorem 1, all smaller jobs are already completed by A when J_i arrives). We determine c_i in such a way that z_i is maximized, i.e. $c_i = (2^{i+1} + 2^{i-1} - 1 - \mathcal{R}_4(2^{i+1} - 1))/(\mathcal{R}_4 - 1)$. Now that we know c_i, we can calculate z_i and w_i to force a competitive ratio of $\mathcal{R}_4$ if A does not restart in step 5 or if it uses the wrong order for the jobs (step 6). We give the results in the following table.

i	1	2	3	4	5
z_i	1.10638	1.48772	2.24592	3.69664	6.91845
w_i	4.55118	5.34374	7.26472	10.2624	11.8410

In the last step, the competitive ratio of A is at least $\mathcal{R}_4$, independent of A's schedule. □

Using a computer, we have been able to improve this bound very slightly using 11 phases instead of 5. Fixing $W = 0.79$ and $W' = 1.285$ we can achieve a lower bound of 1.22324655.

Theorem 4 *Any randomized algorithm for minimizing the total weighted completion time on a single machine which is allowed to restart jobs, has a competitive ratio of at least* $\mathcal{R}_5 = 1.1161$.

Proof. We use Yao's minimax principle and consider a randomized adversary against a deterministic algorithm. We use the following job sequence.

time	size	weight	number
0	1	1	1
0.379739	0	1.88288	1
1	0	7.03995	1
1	ε	ε	k

where $k\varepsilon = c = 3.31003$ and the jobs at time 1 arrive with probability $p = 0.691404$ (either they all arrive, or none of them).

Suppose the jobs at time 1 do arrive, then if the online algorithm A restarts at time $t = 0.37978$, it can choose to restart again at time 1. If it does, it has costs $1.88288 \cdot t + 7.03995 + 2 + \varepsilon \sum_{i=1}^{k}(2 + i\varepsilon) = 9.75495 + 2c + \varepsilon^2 k(k+1)/2$. For $\varepsilon \to 0$, this tends to $16.3750 + c^2/2 = 21.8532$. If A does not restart again, it has costs $1.88288 \cdot t + (7.03995 + 1 + c)(t + 1) + \varepsilon^2 k(k + 1)/2$ which tends to the same limit.

The optimal costs in this case are $2.88288 + 7.03995 + c + \varepsilon^2 k(k + 1)/2 \to 18.7110$.

This implies that if A restarts at time t, it has a competitive ratio of at least $p \cdot 21.8532/18.7110 + 1 - p$, and otherwise, it has a competitive ratio of at least $p + (1 - p) \cdot 2.88288/(2.88288 \cdot t + 1)$. These ratios are equal for $p = 0.691404$, and are then 1.11610796. □

3.2 Lower bounds for preemptive algorithms

Since the unweighted problem can be solved to optimality, we only consider the weighted problem in this section. We show this problem cannot be solved optimally. In the unweighted problem, *SRPT* is optimal. However, in the case that jobs have weights, it is possible that when a new job arrives, the optimal schedule before that time is different compared to the situation where the new job does not arrive. This cannot occur in the unweighted version of the problem. We use this idea to show the following lower bounds.

Theorem 5 *Any deterministic preemptive on-line algorithm for minimizing the total weighted completion time, has a competitive ratio of at least* $\mathcal{R}_6 = 1.0730$.

Proof. The sequence starts with two jobs arriving at time zero. One job of size 1 and weight 1, and the other of size α and weight β, where $1 < \beta < \alpha$. Consider an on-line algorithm A at time α. If the smaller job is completed by then, k very small jobs of size ε and weight $\beta\varepsilon$, of total length c, arrive ($c = k\varepsilon$). Otherwise, no more jobs arrive. In the first case, OPT runs the larger job, then the small jobs and then the unit job. For $\varepsilon \to 0$, the cost is

$$T_{OPT} = \alpha\beta + c\alpha\beta + \beta k(k+1)\varepsilon^2/2 + \alpha + c + 1 = (c+1)(\alpha\beta+1) + \alpha + c^2\beta/2 \ .$$

A is left with a piece of size 1 of the larger job, hence it does not matter in which order it completes the remaining jobs. We can assume that it runs the unit job first, then the larger job, and then the small jobs. Its cost is at least $T_A \geq 1 + \beta(\alpha+1) + (c+1)\alpha\beta + \alpha + c^2\beta/2$. In the second case, OPT runs the unit job first, and hence $T_{OPT} = 1 + (\alpha+1)\beta$, whereas A can either finish the unit job first, but no earlier than time α (and pay $\alpha + \beta(\alpha+1)$), or finish the larger job first (and pay $\alpha\beta + \alpha + 1$). The second cost is always smaller since $\beta > 1$. Using a computer to search for good values for α, β and c, such that the competitive ratio in both cases is high, we get that for $\alpha = 3.4141$, $\beta = 2.5274$, and $c = 4.4580$, the competitive ratio is at least 1.073042. □

Theorem 6 *Any randomized preemptive on-line algorithm for minimizing the total weighted completion time, has a competitive ratio of at least* $\mathcal{R}_7 = 1.0388$.

Proof. We use Yao's minimax principle and consider a randomized adversary against a deterministic algorithm. We use the sequence from Theorem 5. The small jobs arrive at time α with probability p. Consider a deterministic algorithm A. Let $\mathcal{R}_8$ be the competitive ratio in the case A completes the smaller job by time α, and $\mathcal{R}_9$ be the competitive ratio if it does not. Then in the first case $E(T_A/T_{OPT}) \geq \mathcal{R}_8 p + (1-p)$, and in the second case $E(T_A/T_{OPT}) \geq \mathcal{R}_9(1-p) + p$. The best value of p for given $\mathcal{R}_8$ and $\mathcal{R}_9$ can be calculated by making the two expected competitive ratios equal. Using a computer to search for good values for α, β and c, such that the competitive ratio is high, we get that for $\alpha = 3.7299$, $\beta = 2.4036$, and $c = 5.4309$ (and $p = 0.36251$), the expected competitive ratio is at least 1.038872. □

4 Total flow time

For the standard problem without weights, it is known that the competitive ratio is $\Theta(n)$. It is easy to see that there cannot be a competitive algorithm for the standard weighted problem.

Lemma 1 *Any (deterministic or randomized) algorithm for minimizing the total weighted flow time on a single machine that is not allowed to restart or preempt jobs, has an unbounded competitive ratio.*

Proof. We use Yao's minimax principle and consider a randomized adversary against a deterministic algorithm. The adversary works as follows: at time 0, a job of size and weight 1 arrives. At some time t, uniformly distributed over $(0, N)$, where $N > 1$ is some constant, a second job arrives of size 0 and weight N^2. For all t, the optimal total flow time is bounded by 2. We will show the competitive ratio of any algorithm is bounded by $\Omega(N)$.

Suppose the on-line algorithm starts the first job at time S. If $S \geq N/2$, its expected cost is at least $N/2$ and we are done.

Otherwise, there is a probability of $1/(2N)$ that the second job arrives in the interval $(S, S+1/2)$, in which case the algorithm has a cost of at least $N^2/2$. This implies its expected cost is at least $\frac{1}{2N} \cdot \frac{N^2}{2} = \Omega(N)$.

Since we can choose $N > 1$ arbitrarily high, the lemma follows. □

We therefore turn to models where restarts or preemptions are allowed.

4.1 Lower bounds for algorithms with restarts

Theorem 7 *Any (deterministic or randomized) on-line algorithm for minimizing the total flow time, which is allowed to restart jobs, has a competitive ratio of $\Omega(\sqrt{n})$.*

Proof. Consider an on-line algorithm A. We use a job sequence consisting of $n-2$ jobs of size 0, one job of size 3 and one job of size 2. Let $q = \lfloor \sqrt{n-2} \rfloor$. The two large jobs become available at time 0. Also $n - 2 - q^2$ jobs of size 0 arrive at time 0. There are two cases to consider.

Case 1. If A completes the job of size 2 strictly before time 3, we continue as follows: at each time $3 + 2i$ (for $i = 0, 1 \ldots, q-1$) , q short jobs arrive. If A does not delay the process of any small job, then it can start the job of size 3 only at time $1 + 2q$, and $T_A \geq 2q$.

If A runs the job of size 3 earlier than that, then at least one set of small jobs is delayed by at least one unit of time and $T_A \geq q$. OPT assigns the longest job first, and the job of size 2 at time 3, hence no short jobs are delayed and $T_{OPT} = 8$.

Case 2. Otherwise, if at time 2, A is not in a mode where it can complete the job of size 2 strictly before time 3, then q jobs of size 0 arrive at time 2. If A is running some job at that point, and does not stop it then all small jobs will be delayed till time 3 (this is true for any non-zero job) and $T_A \geq q$. All other jobs arrive at time 5 (or any time later). OPT assigns the job of size 2 at time 0 and the other big job at time 2 and $T_{OPT} = 7$. Otherwise an additional $q-1$ sets of q small jobs each, arrive at times $5+i$, for $i = 0, \ldots, q-2$. A can only complete one big job till time 5. The other big job is either postponed till time $1+q$, or processed later, and then at least one set of short jobs is delayed by at least one unit of time, hence $T_A \geq q$. In both cases OPT completes both big jobs at time 5 and $T_{OPT} = 7$.

In all cases $T_{OPT} \leq 8$ and $T_A \geq q$, hence the competitive ratio $\mathcal{R}_{10}$ satisfies $\mathcal{R}_{10} = \Omega(\sqrt{n})$.

The proof can be extended for randomized algorithms with restarts. We use Yao's minimax principle and consider a randomized adversary against a deterministic algorithm. In this case, we use the following distribution on the input: choose with equal probability the first or the second sequence from the proof above. This gives the lower bound of $\Omega(\sqrt{n})$. □

If the jobs can have different weights, the competitive ratio increases to n.

Theorem 8 *Any (deterministic or randomized) on-line algorithm for minimizing the total weighted flow time, which is allowed to restart jobs, has a competitive ratio of $\Omega(n)$.*

Proof. The proof is very similar to that of the previous theorem. We make the following changes:

- let $q = n$,
- at time 0, one job of size 2 and weight 1 arrives and one job of size 3 and weight 1 (and no other jobs),
- in all places where q jobs used to arrive in that proof, we now let one job arrive of size 0 and weight $q = n$.

It is easy to see that still the competitive ratio is $\Omega(q)$, hence the theorem. □

4.2 Lower bounds for preemptive algorithms

In this section we again consider only weighted flow time, since SRPT clearly gives an optimal solution for total flow time. We can show the following lower bound for deterministic algorithms.

Theorem 9 *Any preemptive deterministic on-line algorithm for minimizing the total weighted flow time, has a competitive ratio of at least $\mathcal{R}_{11} = 2$.*

Proof. Consider the following sequence. At time 0 a job of size α and weight β arrives (we call this job the *large* job), such that $1 < \beta < \alpha$. For $i = 1 \ldots q$ ($q < \alpha$), a job of size and weight 1 arrives at time $i - 1$ (*medium* jobs). Consider the on-line algorithm A at time α. Let V be the total length of medium jobs that A processed till that time.

If $V < 1$, no more jobs arrive. In this case A is left with less than size 1 of the large job. Since running pieces of different medium jobs only increases the cost, we assume that A completed a size V of one of the medium jobs. Since no other jobs arrive, there are two cases to consider. It is either best to complete the large job and then all medium jobs, or to complete one medium job, then the large job and then the rest of the medium jobs. In both cases, $T_A \geq \alpha\beta + q(\alpha + 1)$. OPT will run all medium jobs before the large job and $T_{OPT} = q + \beta(\alpha + q)$.

If $V \geq 1$, A is left with at least 1 unit of the large job at time α. Let $\mu \geq \alpha$ be the time where A is left with exactly 1 unit of the large job. At each time $\mu + (i-1)\varepsilon$, for $i = 1, \ldots, k$, a job of size ε and weight $\beta\varepsilon$ is released (*small* jobs). Again we may assume that A does not start a medium job before completing the previous one. Then let $j = \lfloor \mu - \alpha + 1 \rfloor$ be the number of medium jobs completed by A before time μ. Let $V' = \mu - \alpha + 1 - j$, this is the part of a medium job that started its process by A but was not completed by time μ. Since no more jobs arrive, A decides whether it should complete this job before the small jobs. The rest of the medium jobs clearly run after the small jobs. Let j' be the number of medium jobs that A runs before the small jobs ($j' \in \{j, j+1\}$). Hence

$$T_A \geq j' + (\alpha + j')\beta + k(\varepsilon + 1)\beta\varepsilon + (q - j')(\alpha + k\varepsilon + 1) \ .$$

OPT runs only $j' - 1$ medium jobs before time μ and completes the large job at time μ. At time $j' + \alpha - 1$ OPT is left with all small jobs and $q - j' + 1$ medium jobs which have lower priority. Hence

$$T_{OPT} = j' - 1 + (\alpha + j' - 1)\beta + k\varepsilon\beta\varepsilon + (q - j' + 1)(\alpha + k\varepsilon + 1) \ .$$

Taking q to be large enough, $\beta = q$, $\alpha = q^2$ and $k\varepsilon = q^3$ where ε tends to zero, we get that the competitive ratio in both cases tends to 2. □

We use a similar method for randomized algorithms.

Theorem 10 *Any preemptive randomized on-line algorithm for minimizing the total weighted flow time, has a competitive ratio of at least* $\mathcal{R}_{12} = 4/3$.

Proof. We use Yao's minimax principle and consider a randomized adversary against a deterministic algorithm. Consider the sequence introduced in Theorem 9. We use the same sequence for $q = 1$ (this sequence is similar to the one given in the proofs for completion time, except that now the small jobs arrive with intervals of ε apart). Since $V \leq 1$, the second case must satisfy $\mu = \alpha$. With probability p, the small jobs arrive starting at time α as in Theorem 9 (they all arrive, or none of them arrives).

We fix $\alpha >> \beta$, and $k\varepsilon = \alpha^2$. Then the competitive ratio in the first case is $(\beta + 1)/\beta$ if the small jobs do not arrive (and 1 otherwise). In the second case, $j' = 1$ and the competitive ratio is β. The best choice for p is $(\beta^2 - \beta)/(\beta^2 - \beta + 1)$. Then the expected competitive ratio is $\beta^2/(\beta^2 - \beta + 1)$. Maximizing this expression we get $\beta = 2$ and expected competitive ratio of at least $4/3$. □

5 Conclusions and open questions

An interesting general question is what the difference is between minimizing the weighted and unweighted total completion time, in terms of the competitive ratios that can be achieved.

Based on the results in this paper, we know that the preemptive versions of the weighted completion and flow time problems are different from the unweighted versions, since if the jobs all have the same weight it is possible to schedule the jobs optimally and have a 1-competitive algorithm, both for completion times and for flow times.

It is possible however, that minimizing the total completion time in the standard model and in the model with restarts is as hard in the weighted problem as it is in the unweighted problem. These problems are still open.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–43, October 1999.

[2] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[3] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 609–618. SIAM, Philadelphia, PA, 1997.

[4] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. manuscript, 1999.

[5] Michel X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 591–598, New York / Philadelphia, 1997. ACM / SIAM.

[6] J.A. Hoogeveen and A.P.A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proc. 5th Int. Conf. Integer Programming and COmbinatorial Optimization*, LNCS, pages 404–414. Springer, 1996.

[7] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single maching. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 418–426, Philadelphia, Pennsylvania, 1996.

[8] J. Labetoulle, E..L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. *Progress in Combinatorial Optimization*, pages 245–261, 1984.

[9] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In *Handbooks in operations research and management science*, volume 4, pages 445–522. North Holland, 1993.

[10] C.A. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95)*, volume 955 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 1995.

[11] A.S. Schulz and M. Skutella. The power of alpha-points in preemptive single machine scheduling. manuscript, 1999.

[12] M. Skutella. personal communication, 2000.

[13] W. E. Smith. Various optimizers for single-stage production. *Naval Research and Logistics Quarterly*, 3:59–66, 1956.

[14] L. Stougie. unpublished manuscript, 1995.

[15] L. Stougie and A.P.A. Vestjens. Randomized on-line scheduling: How low can't you go? unpublished manuscript, 1997.

[16] A. P. A. Vestjens. On-line machine scheduling. Technical report, Ph.D. thesis, Eindhoven University of Technology, The Netherlands, 1997.

[17] A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 18th Annual Symposium on Foundations of Computer Science*, pages 222–227. IEEE, 1977.