Available online at www.sciencedirect.com

SCIENCE DIRECT®

Discrete Applied Mathematics 154 (2006) 611–621

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Optimal on-line flow time with resource augmentation☆

Leah Epstein[a,1], Rob van Stee[b,*,2]

[a]*Department of Mathematics, University of Haifa, 31905 Haifa, Israel*
[b]*Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany*

Received 14 November 2001; received in revised form 14 June 2004; accepted 6 May 2005
Available online 26 October 2005

Abstract

We study the problem of scheduling n jobs that arrive over time. We consider a non-preemptive setting on a single machine. The goal is to minimize the total flow time. We use extra resource competitive analysis: an optimal off-line algorithm which schedules jobs on a single machine is compared to a more powerful on-line algorithm that has ℓ machines. We design an algorithm of competitive ratio $1 + 2\min(\Delta^{1/\ell}, n^{1/\ell})$, where Δ is the maximum ratio between two job sizes, and provide a lower bound which shows that the algorithm is optimal up to a constant factor for any constant ℓ. The algorithm works for a hard version of the problem where the sizes of the smallest and the largest jobs are not known in advance, only Δ and n are known. This gives a trade-off between the resource augmentation and the competitive ratio.

We also consider scheduling on parallel identical machines. In this case the optimal off-line algorithm has m machines and the on-line algorithm has ℓm machines. We give a lower bound for this case. Next, we give lower bounds for algorithms using resource augmentation on the speed. Finally, we consider scheduling with hard deadlines, and scheduling so as to minimize the total completion time.

Keywords: Flow time; On-line algorithms; Scheduling; Resource augmentation

1. Introduction

Minimizing the total flow time is a well-known and hard problem, which has been studied widely both in on-line and in off-line environments [1,7,8]. The flow time $f(J)$ of a job J is defined as its completion time, $C(J)$, minus the time at which it arrived, $r(J)$ (the release time of J). This measure is applicable to systems where the load is proportional to the total number of bits that exist in the system over time (both of running jobs and of waiting jobs). In this paper, we consider on-line algorithms using resource augmentation, and we examine the effects on the performance of an algorithm if it has more or faster machines than the off-line algorithm (see [6,9]).

We consider the following on-line scheduling problem. The algorithm has parallel identical machines, on which it must schedule jobs that arrive over time. A job J (which arrives at time $r(J)$) with processing requirement $P(J)$ (also

☆ A preliminary version of this paper appeared in Proceedings of 13th Fundamentals of Computation Theory, Lecture Notes in Computer Science, vol. 2138, 2001, pp. 472–482. It was presented at the WEA workshop.

* Corresponding author.
E-mail addresses: lea@math.haifa.ac.il (L. Epstein), vanstee@ira.uka.de (R. van Stee).

[1] Work carried out while the author was at Tel-Aviv University.

[2] Work supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-30-002, and partially by project number 612-061-000. Work carried out while the author was at CWI, The Netherlands.

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.05.016

called running time or size) that becomes known upon arrival has to be assigned to one of the machines and run there continuously for $P(J)$ units of time. The objective is to minimize the sum of flow times of all jobs. The total number of jobs is n.

We compare on-line algorithms that are running on ℓm machines ($\ell \geqslant 1$) to an optimal off-line algorithm, denoted by OPT, that is running on m machines but knows all the jobs in advance. Such on-line algorithms are also called ℓ-machine algorithms, since they use ℓ times as much machines as the optimal off-line algorithm. An algorithm that uses the same number of machines as the off-line algorithm, but uses machines which are $s > 1$ times faster, is called an s-speed algorithm.

For a job sequence σ and an on-line algorithm A, we denote the total flow time of σ in the schedule of A on ℓm machines by $A_{\ell m}(\sigma)$. We denote the optimal total flow time for σ on m machines by $\mathrm{OPT}_m(\sigma)$. The competitive ratio using resource augmentation is defined by

$$r_{m,\ell m}(A) = \sup_{\sigma} \frac{A_{\ell m}(\sigma)}{\mathrm{OPT}_m(\sigma)},$$

where the supremum is taken over all possible job sequences σ. The goal of an on-line algorithm is to minimize this ratio.

Approximating the flow time is hard even in an off-line environment. For a single machine, Kellerer et al. [7] presented an $O(\sqrt{n})$-approximation algorithm and gave a lower bound of $\Omega(n^{1/2-\varepsilon})$ on the approximation ratio of any polynomial-time algorithm, provided $P \neq NP$. For parallel machines, Leonardi and Raz [8] showed that when preemption is allowed, the algorithm Shortest Remaining Processing Time (SRPT) is an $O(\log(\min\{n/m, P\}))$-approximation and this is optimal. The solution from SRPT can be transformed into a non-preemptive solution at a cost of an $O(\sqrt{n/m})$ factor in the approximation ratio. Finally, they showed an $\Omega(n^{1/3-\varepsilon})$ lower bound on the approximability of the non-preemptive problem, provided $P \neq NP$.

In an on-line environment it is well-known that the best competitive ratio of any algorithm that uses a single machine is n (easily achieved by a greedy algorithm). The problem has been studied introducing resource augmentation by Phillips et al. [9]. They give algorithms with augmentation on the number of machines. These are an $O(\log n)$-machine algorithm (which has a competitive ratio $1+o(1)$) and an $O(\log \Delta)$-machine algorithm (which achieves the competitive ratio 1), where Δ is the maximum ratio between sizes of jobs. Both algorithms are valid for every m.

We give an algorithm LEVELS and show that $r_{1,\ell}(\text{LEVELS}) = 1 + 2\min(n^{1/\ell}, \Delta^{1/\ell}) = O(\min(n^{1/\ell}, \Delta^{1/\ell}))$, where n is the number of jobs that arrive. This algorithm works for a hard version of this problem where the sizes of the smallest and the largest jobs are not known in advance; only Δ and n are known in advance. Thus the performance of our algorithm is invariant under job scaling. The algorithms above from [9] work only if the job size limits are known in advance; particularly the size of the largest job must be known. We do not see how these algorithms can be adapted for the harder version of the problem.

Furthermore, we show that for all on-line algorithms A and number $1 \leqslant m_1 \leqslant \ell$ of off-line machines we have $r_{m_1,\ell}(A) = \Omega(\min(n^{1/\ell}, \Delta^{1/\ell})/(12\ell)^{\ell})$. This shows that LEVELS is optimal up to a constant factor for any constant ℓ against an adversary on one machine.

In [4], a related problem on a network of links is considered. It immediately follows from our lower bounds that any constant competitive algorithm has a polylogarithmic number of machines. More precisely, if A has a constant competitive ratio and ℓm machines, $\ell \geqslant \Omega(\sqrt{\log(\min(n, \Delta))}/(m\sqrt{\log\log(\min(n, \Delta))}))$. This result can also be deduced from Theorem 10 in [4]. However, using their proof for the general lower bound would give only an exponent of $1/2\ell$. Improving the exponent to be the tight exponent $1/\ell$ is non-trivial. Our results imply that by choosing a given amount of resource augmentation, the competitive ratio is fixed. We adapt the lower bound for the case where the on-line algorithm has faster machines than the off-line algorithm. This results in a lower bound of $\Omega(n^{1/(2m^2)})$ on the speed of on-line machines, if $\ell = 1$.

We also consider the following scheduling problem studied in [3,9]. Each job J has a deadline $d(J)$. Instead of minimizing the flow time, we require that each job is finished by its deadline, effectively limiting the flow time of job J to $d(J) - r(J)$. The goal is to complete all jobs on time. We show that a non-preemptive on-line algorithm can only succeed on any sequence if $\ell = \Omega(\log \Delta)$ for constant speed s, or $s = \Omega(\Delta^{1/\ell})$ for a constant number ℓ of machines.

Finally, we discuss the problem of minimizing the total completion time using resource augmentation. We present an algorithm WAIT which has an optimal competitive ratio of $1 + 1/s$ on one machine of speed s, and a competitive ratio of $1 + 1/\sqrt{\ell}$ on ℓ machines of speed 1.

Throughout the paper, for a specific schedule ζ for the jobs, we denote the starting time of job J by $S_\zeta(J)$, and its flow time by $f_\zeta(J) = C_\zeta(J) - r(J)$. We omit the subscripts if the schedule is clear from the context.

2. Algorithms with resource augmentation

As stated, in this paper we consider the case where n is known and the on-line algorithm has more machines than the off-line algorithm that it is compared to. In the following two lemmas, we show what happens if one of these conditions does not hold.

Lemma 1. *Any on-line algorithm for minimizing the total flow time on ℓ parallel machines has a competitive ratio of $\Omega(n^{1-\varepsilon})$ for any $\varepsilon > 0$ if it does not known in advance, even if it is compared to an off-line algorithm on one machine.*

Proof. Consider an on-line algorithm $\mathscr{A}$ and a constant $0 < \varepsilon < 1$. Take $N = \max(4, 2/\varepsilon)$. Define $N_0 = 1$, $N_1 = N$ and $N_i = N_{i-1}^N$ for $i > 1$.

One job of size 1 arrives at time 0. This is phase 0. For any phase $i > 0$, denote the time that $\mathscr{A}$ starts the first job of phase $i - 1$ by t_i.

In phase $i = 1, \ldots, \ell$, jobs of size $1/N_i$ arrive starting at time t_i and with intervals of $1/N_i$ during the next time interval of length $1/(2N_{i-1})$ or until $\mathscr{A}$ starts one of them. If $t_{i+1} < t_i + 1/(2N_{i-1})$ and $i < m$, the next phase starts; otherwise, the sequence stops.

If the sequence continues until phase j, then $\mathscr{A}$ is using j machines at the same time. Thus, the sequence continues until at most phase ℓ.

Suppose the sequence continues until phase j. Then at least $N_j/(2N_{j-1})$ jobs of size $1/N_j$ have an average flow time of at least $1/(4N_{j-1})$. This gives a total flow time of at least $N_j/(8N_{j-1}^2) = N_{j-1}^{N-2}/8$. Moreover, the total number of jobs in this case is

$$n \leqslant \sum_{i=0}^{j} N_i = 1 + \sum_{i=0}^{j-1} N^{N^i} < \sum_{i=0}^{N^{j-1}} N^i$$
$$= \frac{N^{N^{j-1}+1} - 1}{N-1} < \frac{N}{N-1} N^{N^{j-1}} = \frac{N}{N-1} N_j < 2N_j$$

and therefore

$$\frac{N_{j-1}^{N-2}}{8} = \frac{N_j^{1-2/N}}{8} > \left(\frac{n}{2}\right)^{1-2/N}/8 = \Omega(n^{1-\varepsilon}).$$

We now turn to the off-line assignment of the jobs. The idea is to schedule each job at a time that $\mathscr{A}$ does not schedule it. The very first job starts at time 0 if $t_1 \geqslant 1$ and at time $t_1 + 1 < 2$ otherwise, and has a flow time of at most 3. In each phase $i > 0$, each job is started at the time it arrives unless this would overlap with time t_{i+1}. At time t_{i+1}, no more jobs from phase i will arrive and OPT has completed all but one of them.

The last job in phase i is started by OPT at time

$$t_{i+1} + \frac{1}{N_i} < t_i + \frac{1}{2N_{i-1}} + \frac{1}{N_i} < t_i + \frac{1}{N_{i-1}} - \frac{1}{N_i}$$

and thus completes before time $t_{i-1} + 1/N_{i-1}$ (which is the time at which OPT starts the last job of phase $i - 1$). Thus, the total flow time of jobs in this phase is at most $1/N_{i-1}$, which is the length of the interval in which all these jobs arrive and are completed.

In this way, it is possible to assign all the jobs without any overlap, thus creating a valid off-line schedule on a single machine. The total flow time of this schedule is at most $3 + 1 + 1/N_1 + \cdots + 1/N_{\ell-1} < 4 + \sum_{i=1}^{\infty} 1/2^i = 5$.

Since the optimal total flow time is bounded by a constant, we find that the competitive ratio is $\Omega(n^{1-\varepsilon})$. □

Lemma 2. *$r_{m,m}(\mathscr{A}) = \Omega(n/m^2)$ for all on-line algorithms A for minimizing the total flow time on m parallel machines.*

Proof. A single unit job arrives at time 0. Let t be the time at which $\mathscr{A}$ starts this job. Let $\mu = m/(2n-2)$. For $j = 0, \ldots, (n-1)/m$, m jobs of length μ are released at time $t + j\mu$. The optimal schedule runs all these jobs immediately when they arrive, and the first job either before or after them. Thus it has a total flow time equal to the total size of the small jobs plus 3. This is $(n-1)\mu + 3 = \Theta(m)$.

On the other hand, at each time $t + j\mu$ for $j = 0, \ldots, (n-1)/m$, one small job arrives that $\mathscr{A}$ cannot start immediately. In total, it must delay $(n-1)/m$ such jobs until time $t + 1/2$. For the total flow time, it does not matter which of the jobs $\mathscr{A}$ delays, since all the small jobs have the same size. For the calculations, assume it delays one job from each step until time $t + 1/2$. This means it delays $(n-1)/m$ small jobs for at least $\frac{1}{4}$ time on average. Thus its flow time is $\Omega(n/m)$. Consequently, $r_{m,m}(A) = \Omega(n/m^2)$. □

Given these results, from now on we assume that n is known in advance. The number of off-line machines is $m = 1$, whereas the online algorithm has $\ell m = \ell$ machines. We define an algorithm LEVELS. LEVELS uses ℓ priority queues $Q_1, \ldots, Q_\ell$ (one for each machine) and ℓ variables $D_1 \geqslant \cdots \geqslant D_\ell$. We initialize $Q_i = \emptyset$ and $D_i = 0$. An *event* is either an arrival of a new job or a completion of a job by a machine. Let $\gamma = n^{1/\ell}$, where n is the number of jobs.

Algorithm (LEVELS)**.**

- If a few events occur at the same time, the algorithm first deals with all arrivals before it deals with job completions.
- On completion of a job on machine i, if $Q_i \neq \emptyset$, a job of minimum release time among jobs with minimum size in Q_i is scheduled immediately on machine i. (The job is dequeued from Q_i.)
- On arrival of a job J, let i be a minimum index of a machine for which $D_i \leqslant \gamma P(J)$. If there is no such index, take $i = m$. If machine i is idle, J is immediately scheduled on machine i, and otherwise, J is enqueued into Q_i. If $P(J) > D_i$, D_i is modified by $D_i \leftarrow P(J)$.

We analyze the performance of LEVELS compared to a preemptive OPT on a single machine. Denote the schedule of LEVELS by π. Partition the schedule of each machine into blocks. A block is a maximal sub-sequence of jobs of non-decreasing sizes, that run on one machine consecutively, without any idle time.

- Let N_i be the number of blocks in the schedule of LEVELS on machine i.
- Let $B_{i,k}$ be the kth block on machine i.
- Let $b_{i,k,j}$ be the jth job in block $B_{i,k}$.
- Let $N_{i,k}$ be the number of jobs in $B_{i,k}$.
- Let $P_{i,k}$ be the size of the largest job in blocks $B_{i,1}, \ldots, B_{i,k}$ i.e.

$$P_{i,k} = \max_{1 \leqslant r \leqslant k} \max_{1 \leqslant j \leqslant N_{i,r}} P(b_{i,r,j}),$$
$$P_{i,0} = 0 \quad \text{for all } 1 \leqslant i \leqslant \ell.$$

- Let $I = \bigcup_{1 \leqslant i \leqslant \ell, 1 \leqslant k \leqslant N_i} B_{i,k}$, i.e. I is the set of all jobs.

Similar to the proof in [5], we define a pseudo-schedule ψ on ℓ machines, in which job $b_{i,k,j}$ is scheduled on machine i at time $S_\pi(b_{i,k,j}) - P_{i,k-1}$. Note that ψ is not necessarily a valid schedule, since some jobs might be assigned in parallel, and some jobs may start before their arrival times.

The amount that jobs are shifted backwards increases with time. Therefore, if there is no idle time between jobs in π, there is no idle time between them in ψ either. Note that in ψ, the flow time of a job J can be smaller than $P(J)$, and even negative.

We introduce an extended flow problem. Each job J has two parameters $r(J)$ and $r'(J)$, where $r'(J) \leqslant r(J)$. $r'(J)$ is the pre-release time of job J. Job J may be assigned starting from time $r'(J)$. The flow time is still defined by the completion time minus the release time, i.e., $f(J) = C(J) - r(J)$. Going from an input σ for the original problem to an input σ' of the extended problem requires definition of the values of r' for all jobs. Clearly, the optimal total flow time for an input σ' of the extended problem is no larger than the flow time of σ in the original problem.

Let I_i be the set of jobs that run on machine i in π. We define an instance I_i' for the extended problem. I_i' contains the same jobs as I_i. For each $J \in I_i$, $r(J)$ remains the same. Define $r'(J) = \min\{r(J), S_\psi(J)\}$. Clearly $\text{OPT}(I) \geqslant \sum_{i=1}^{\ell} \text{OPT}(I_i) \geqslant \sum_{i=1}^{\ell} \text{OPT}(I_i')$, where $\text{OPT}(I_i)$ is the preemptive optimal off-line cost for the jobs that LEVELS scheduled on machine i. We consider a preemptive optimal off-line schedule ϕ_i for I_i' on a single machine. In ϕ_i, jobs of equal size are completed in the order of arrival. Ties are broken as in π. The following lemma is similar to [5].

Lemma 3. *For each job $J \in I_i'$, $f_{\phi_i}(J) \geqslant f_\psi(J)$.*

Proof. Since $r_{\phi_i}(J) = r_\psi(J)$ for each job J, we only have to show that in ϕ_i, J does not start earlier than it does in ψ. Assume to the contrary this is not always the case. Let J_1 be the first job in ϕ_i for which $S_{\phi_i}(J_1) < S_\psi(J_1)$. Note that in this case $r'(J_1) < S_\psi(J_1)$ and hence $r(J_1) < S_\psi(J_1)$. Let t be the end of the last idle time before $S_\psi(J_1)$, and let $B_{i,k}$ be the block that contains J_1.

Suppose $P_{i,k-1} \leqslant P(J_1)$. Then all jobs that run on machine i from time t until time $S_\psi(J_1)$ in ψ are either smaller than $P(J_1)$ or have the same size, but are released earlier. Moreover, these jobs do not arrive earlier than time t, hence in ϕ_i they do not run before time t. They do run before $S_{\phi_i}(J_1)$ because they have higher priority, hence $S_{\phi_i}(J_1) \geqslant S_\psi(J_1)$, a contradiction.

Suppose $P_{i,k-1} > P(J_1)$. J_1 was available to be run in ψ during the interval $[r(J_1), S_\psi(J_1)]$ since $r(J_1) < S_\psi(J_1)$. In π, all jobs running in the interval $[r(J_1), S_\pi(J_1)]$ are smaller than J_1 (or arrived before, and have the same size), except for the first one, say J_2. Since in ψ, all these jobs are shifted backwards by at least the size of J_2, during $[r(J_1), S_\psi(J_1)]$ only jobs with higher priority than J_1 are run in ψ. J_1 is the first job which starts later in ψ than it does in ϕ_i, so these jobs occupy the machine until time $S_\psi(J_1)$, hence $S_\psi(J_1) \leqslant S_{\phi_i}(J_1)$. □

Theorem 1. $r_{1,\ell}(\text{LEVELS}) = 1 + 2n^{1/\ell}$.

Proof. Using Lemma 3 we can bound the difference between ψ and π. Since $\text{LEVELS}(b_{i,k,j}) = C_\psi(b_{i,k,j}) + P_{i,k-1} - r(b_{i,k,j})$, we have

$$\begin{aligned}
\text{LEVELS}(I) &= \sum_{1 \leqslant i \leqslant \ell} \sum_{1 \leqslant k \leqslant N_i} \sum_{1 \leqslant j \leqslant N_{i,k}} (C_\psi(b_{i,k,j}) + P_{i,k-1} - r(b_{i,k,j})) \\
&\leqslant \sum_{b_{i,k,j} \in I} (C_\psi(b_{i,k,j}) - r(b_{i,k,j})) + \sum_{b_{i,k,j} \in I} P_{i,k-1} \\
&\leqslant \text{OPT}(I) + \sum_{b_{i,k,j} \in I} P_{i,k-1}.
\end{aligned}$$

Let P be the maximum job size. We show the following properties:

$$P_{i,k-1} \leqslant \gamma P(b_{i,k,j}) \quad \text{for each job } b_{i,k,j},\ 1 \leqslant i \leqslant \ell - 1, \tag{1}$$

$$P_{\ell,k-1} \leqslant \frac{P}{\gamma^{\ell-1}} \quad \text{for each job } b_{\ell,k,j}. \tag{2}$$

Adding both properties together we get

$$\begin{aligned}
\text{LEVELS}(I) &\leqslant \text{OPT}(I) + \sum_{\substack{b_{i,k,j} \in I \\ i \neq \ell}} P_{i,k-1} + \sum_{b_{\ell,k,j} \in I} P_{\ell,k-1} \\
&\leqslant \text{OPT}(I) + \gamma \sum_{b_{i,k,j} \in I} P(b_{i,k,j}) + \sum_{b_{\ell,k,j} \in I} \frac{P}{\gamma^{\ell-1}} \\
&\leqslant \text{OPT}(I) + \gamma \text{OPT}(I) + n \cdot \frac{\text{OPT}(I)}{n^{(\ell-1)/\ell}} = (2\gamma + 1) \cdot \text{OPT}(I).
\end{aligned}$$

This holds since $\text{OPT}(I) \geqslant P$ and $\text{OPT}(I)$ is at least the sum of all job sizes, and since $|I| = n$.

To prove (1) we recall that $b_{i,k,j}$ was assigned to machine i because it satisfied $D_i \leqslant \gamma P(b_{i,k,j})$. If $P_{i,k-1} \leqslant P(b_{i,k,j})$ we are done. Otherwise the job of size $P_{i,k-1}$ arrived before $b_{i,k,j}$ and hence when $b_{i,k,j}$ arrived, D_i satisfied $D_i \geqslant P_{i,k-1}$, hence, $P_{i,k-1} \leqslant D_i \leqslant \gamma P(b_{i,k,j})$.

To prove (2) we show by induction that every job J on machine i in LEVELS satisfies $P(J) \leqslant P/\gamma^{i-1}$. This is trivial for $i = 1$. Assume it is true for some machine $i \geqslant 1$, then at all times $D_i \leqslant P/\gamma^{i-1}$ holds. Hence, a job J' that was too small for machine i satisfied $P(J') \leqslant D_i/\gamma \leqslant P/\gamma^i$. This completes the proof. □

We give a variant of LEVELS with a competitive ratio which depends on Δ, the ratio between the size of the largest job and the size of the smallest job.

Algorithm. REVISED LEVELS: Run LEVELS with $\gamma = \Delta^{1/\ell}$.

Theorem 2. $r_{1,\ell}(\text{REVISED LEVELS}) = 1 + \Delta^{1/\ell}$.

Proof. The proof is very similar to the proof of Theorem 1. The only difference in the proof is that property (1) also holds for machine ℓ (this follows from property (2) and the definition of Δ), hence the competitive ratio is now $\gamma + 1$. □

Hence if $\Delta < n$, we can get a competitive ratio of $1 + \Delta^{1/\ell}$, and otherwise, we find a competitive ratio of $1 + 2n^{1/\ell}$. Taking $\gamma = \min(n^{1/\ell}, \Delta^{1/\ell})$ we can get a competitive ratio of at most $1 + 2\gamma = \mathrm{O}(\min(n^{1/\ell}, \Delta^{1/\ell}))$.

3. Lower bounds for resource augmentation

Theorem 3. *Let A be an on-line scheduling algorithm to minimize the total flow time on ℓ machines. Then for any $1 \leqslant m_1 \leqslant \ell$ and sequences consisting of $\mathrm{O}(n)$ jobs, $r_{m_1,\ell}(A) = \Omega(n^{1/\ell}/(12\ell)^{\ell-1})$.*

We first describe a job sequence σ and then show that it implies the theorem. Let n be an integer. There will be at most $\sum_{i=0}^{\ell} n^{i/\ell}$ jobs in σ (note $\sum_{i=0}^{\ell} n^{i/\ell} = \Theta(n)$). We build σ recursively, defining the jobs according to the behavior of the on-line algorithm A.

Definition. A job j of size α is considered *active*, if the previous active job of size α is completed by A at least α units of time before j is assigned, and j finishes before or when the job that caused its arrival finishes.

The first job in σ has size n and arrives at time 0. We consider it to be an active job. On an assignment of a job j of size α by A, do the following:

- If j is active, and all other machines are running larger jobs (all machines are consequently busy for at least (α units of time), n jobs of size 0 arrive immediately). No more jobs will arrive.
- Otherwise, if j is active, then j causes the arrival of $n^{1/\ell}$ jobs of size $\frac{1}{3}\alpha/n^{1/\ell}$. These jobs arrive starting the time that j is assigned, every $\alpha/n^{1/\ell}$ units of time, until they all have arrived.
- In all other cases (j is not active), no jobs arrive till the next job that A starts.

Lemma 4. $\mathrm{OPT}_1(\sigma) \leqslant 6n$.

Proof. We show that all jobs can be assigned on a single machine, during an interval of length $2n$, so that a job of length α has a flow time of at most 3α. The total flow time then follows.

We show how to assign all jobs of a certain size α so that no active jobs of size α are running at the same time on on-line machines, i.e. the intervals used by A to run active jobs of size α and the intervals that are used by OPT to run jobs of size α are disjoint. Smaller jobs are assigned by OPT during the intervals in which A assigned active jobs of size α. Hence, the time slots given by the optimal off-line for different jobs are disjoint.

Finally, we show how to define those time slots. A job j of size α that arrives at time t is not followed by other jobs of size α until time $t + 3\alpha$. Since an active on-line job starts at least α units of time after the previous active job of this

size (α) is completed, there is a time slot of size at least α during the interval $[t, t+3\alpha]$ where no active job of size α is running on any of the on-line machines. The optimal off-line algorithm can assign j during that time. This is true also for the first job. Finally, the optimal algorithm can also manage the jobs of size 0 easily by running them immediately when they arrive. Hence, the total time that the optimal off-line machine is not idle is at most $2n$. □

We partition jobs into three types, according to the on-line assignment. A job that arrived during the processing of a job of size α, and has size $\frac{1}{3}(\alpha/n^{1/\ell})$ is either active or *passive* (if it is not active, but completed before the job of size α is completed). Otherwise, the job is called *late*. Let $P(\alpha)$, $T(\alpha)$ and $L(\alpha)$ denote the number of passive, active and late jobs of size α (respectively). Let $N(\alpha) = P(\alpha) + T(\alpha) + L(\alpha)$.

Claim 1. $T(\alpha) \geqslant \lceil (P(\alpha) + T(\alpha))/(2\ell) \rceil$.

Proof. The number of jobs of size α that the on-line algorithm can complete during 2α units of time (until a job can be active again) is at most 2ℓ. □

Now we are ready to prove Theorem 3.

Proof of Theorem 3. According to the definition of the sequence, $N(\alpha) = n^{1/\ell} \cdot T(3\alpha n^{1/\ell})$. We distinguish two cases.

Case 1: In all phases $L(\alpha) \leqslant \frac{1}{2} N(\alpha)$. Hence $T(\alpha) \geqslant (1/4\ell)N(\alpha)$ for all α. This is true for $\alpha = (\frac{1}{3})^{\ell-1} n^{1/\ell}$ (the smallest non-zero jobs) and hence there are at least $n^{\ell-1/\ell} \cdot (1/4\ell)^{\ell-1} > 0$ such jobs. Therefore, the zero jobs arrive and are delayed by at least $(\frac{1}{3})^{\ell-1} \cdot n^{1/\ell}$ units of time. Since their flow time is at least $n \cdot n^{1/\ell} \cdot (\frac{1}{3})^{\ell-1}$, and the optimal flow time is at most $6n$, the competitive ratio follows.

Case 2: There is a phase where $L(\alpha) > \frac{1}{2} N(\alpha)$. Consider the phase with largest α in which this happens. Since for larger sizes α' we have $L(\alpha') \leqslant \frac{1}{2} N(\alpha')$, we can bound the number of jobs of size α (for $\alpha = (\frac{1}{3})^i n^{(1-i)/\ell}$) by $N(\alpha) \geqslant n^{i/\ell}/(4\ell)^i$. The late jobs are delayed by at least $\frac{3}{4}\alpha n^{1/\ell}$ on average. (This is the delay if for each job of size $3\alpha n^{1/\ell}$ the *last* $\frac{1}{2} n^{1/\ell}$ jobs of size α that arrive are the ones that are late; in all other cases, the delay is bigger.)

The total flow time is at least

$$\begin{aligned} A_l(\sigma) &\geqslant L(\alpha) \frac{1}{4} 3\alpha n^{1/\ell} \geqslant \frac{1}{2} n^{i/\ell} \left(\frac{1}{4\ell}\right)^i \frac{1}{4} \left(\frac{1}{3}\right)^{i-1} n^{1-i/\ell+1/\ell} \\ &= \frac{1}{(4\ell)^i} \frac{1}{8} \left(\frac{1}{3}\right)^{i-1} n^{1+1/\ell} = \left(\frac{1}{12\ell}\right)^i \frac{3}{8} n^{1+1/\ell} \\ &\geqslant \frac{1}{(12\ell)^{\ell-1}} \frac{3}{8} n^{1+1/\ell} = \Omega\left(\frac{n^{1/\ell}}{(12\ell)^{\ell-1}}\right) \Theta(n). \end{aligned}$$

Since the optimal flow is $\Theta(n)$, the competitive ratio follows. □

Theorem 4. *Let A be an on-line scheduling algorithm to minimize the total flow time on ℓ machines. Then $r_{m_1,\ell}(A) = \Omega(\Delta^{1/\ell}/(12\ell)^\ell)$ for any $1 \leqslant m_1 \leqslant \ell$ if the maximum ratio between jobs is Δ.*

Proof. We adjust σ by starting with a job of size Δ and fixing $n = \Delta/3^\ell$. We assume $\Delta \geqslant 6^\ell$ so that $n \geqslant 2^\ell$ and $n^{1/\ell} \geqslant 2$, which is needed for the construction of the sequence.

Starting from here, we build a sequence σ' in exactly the same way as σ, except that we do not let jobs of size 0 arrive. Clearly, $\text{OPT}_1(\sigma') \leqslant 6\Delta$. We can follow the proof of Theorem 3. However, we now know that all the smallest jobs will be late. If they arrive, we are in the second case of the proof; but if they do not, then for an earlier α we must have $L(\alpha) > \frac{1}{2} N(\alpha)$. So only Case 2 remains of that proof.

The total flow is at least $\frac{3}{8} n^{1/\ell}/(12\ell)^\ell \Delta = \frac{1}{8} \Delta^{1/\ell}/(12\ell)^\ell \Delta$ (because now $i \leqslant \ell$ instead of $i \leqslant \ell - 1$), giving the desired competitive ratio. □

A direct consequence of Theorems 3 and 4 is the following bound on the number of machines needed to maintain a constant competitive ratio. This corollary can also be proved using a simple adaptation of Theorem 10 in [4].

Corollary 1. *Any on-line algorithm for minimizing total flow time on m machines that uses resource augmentation and has a constant competitive ratio, is an* $\Omega(\sqrt{\log(\min(n, \Delta))}/(m\sqrt{\log\log(\min(n, \Delta))}))$*-machine algorithm* (*on sequences of* $\Theta(n)$ *jobs*).

Next we consider resource augmentation on the speed as well as on the number of machines. We consider an on-line algorithm which uses machines of speed $s > 1$. The optimal off-line algorithm uses machines of speed 1.

Theorem 5. *Let A be an on-line scheduling algorithm to minimize the total flow time on* ℓ *machines. Let* $s > 1$ *be the speed of the on-line machines. Then* $r_{m_1,\ell}(A) = \Omega(n^{1/\ell}/(s(12\ell s^2)^{\ell-1}))$ *for any* $1 \leqslant m_1 \leqslant \ell$ *and sequences consisting of* $O(n)$ *jobs. Furthermore,* $r_{m_1,\ell}(A) = \Omega(\Delta^{1/\ell}/(s(12\ell s^2)^{\ell}))$ *for any* $1 \leqslant m_1 \leqslant \ell$.

Proof. Again, we use a job sequence similar to σ. The jobs of phase i now have size $1/(3s^2n^{1/\ell})^i$. For the Δ-part of the proof, we fix $n = \Delta/(3s^2)^{\ell}$. Similar calculations as in the previous proofs result in the stated lower bounds. □

Corollary 2. *Any on-line algorithm for minimizing total flow time on m machines that uses resource augmentation on the speed and has a constant competitive ratio is an* $\Omega(n^{1/(2m^2)})$*- speed algorithm* (*on sequences of* $\Theta(n)$ *jobs*) *and an* $\Omega(\Delta^{1/(2m^2)})$*-speed algorithm.*

4. Other results

4.1. Hard deadlines

We consider the problem of non-preemptive scheduling of jobs with hard deadlines. Each arriving job J has a deadline $d(J)$ by which it must be completed. The goal is to produce a schedule in which all jobs are scheduled such that all of them are completed on time (i.e. by their deadlines). We give a lower bound on the resource augmentation required so that all jobs finish on time. We use a similar lower bounding method to the method we used in Section 3. We allow the on-line algorithm resource augmentation in both the number of machines and their speed. We compare an on-line algorithm that schedules on ℓ machines of speed s to an optimal off-line algorithm that uses a single machine of speed 1.

Let Δ denote the ratio between the largest job in the sequence and the smallest job. The lower bound sequence consists of $\ell + 1$ jobs $J_0, \ldots, J_\ell$ where $P(J_i) = 1/(2s + 1)^i$. We define release times and deadlines recursively; $r(J_0) = 0$ and $d(J_0) = 2 + 1/s$. Let π be the on-line schedule, then $r(J_{i+1}) = S_\pi(J_i)$ and $d(J_{i+1}) = C_\pi(J_i)$. Hence J_{i+1} runs in parallel to all jobs $J_0, \ldots, J_i$ in any feasible schedule π.

Lemma 5. *An optimal off-line algorithm on a single machine of speed* 1 *can complete all jobs on time.*

Proof. For each $i > 0$, $P(J_i) = 1/(2s+1)^i$, hence $d(J_i) - r(J_i) = P(J_{i-1})/s = ((2s+1)/s)P(J_i)$. This holds also for J_0, since $P(J_0) = 1$ and $d(J_0) - r(J_0) = (2s+1)/s$. All jobs arriving after J_i have release times and deadlines in the interval $[S_\pi(J_i), C_\pi(J_i)]$. The optimal off-line algorithm can schedule J_i outside this time interval, and avoid conflict with future jobs. By induction, previous jobs are scheduled before $r(J_i)$ or after $d(J_i)$, so there is no conflict with them either. If $S_\pi(J_i) - r(J_i) \geqslant P(J_i)$, schedule J_i at time $r(J_i)$. Otherwise, $C_\pi(J_i) = S_\pi(J_i) + P(J_i)/s < r(J_i) + P(J_i)(1 + 1/s)$, hence J_i is scheduled at time $C_\pi(J_i)$ and completed at $C_\pi(J_i) + P(J_i) < r(J_i) + P(J_i)(2 + 1/s) = d(J_i)$. □

It is easy to see that the on-line algorithm cannot finish all jobs on time. If the first ℓ jobs finish on time, then all ℓ machines are busy during the time interval $[r(J_\ell), d(J_\ell)]$ and it is impossible to start J_ℓ before time $d(J_\ell)$. We omit the proof of the following theorem

Theorem 6. *The on-line algorithm fails if* $\Delta \geqslant (2s + 1)^\ell$.

We show some corollaries from the lower bound on Δ. These are necessary conditions for an on-line algorithm to succeed on any sequence. Given machines of constant speed s, the number of machines ℓ must satisfy $\ell \geqslant \log \Delta / \log(2s+1)$

i.e. $\ell = \Omega(\log \Delta)$. On the other hand, for a constant number ℓ of machines, s has to satisfy $2s + 1 \geqslant \Delta^{1/\ell}$, i.e. $s = \Omega(\Delta^{1/\ell})$.

The lower bound on Δ clearly holds also for the case where the optimal off-line algorithm is allowed to use $m_1 > 1$ machines. Consider a k-machine algorithm that always succeeds in building a feasible schedule ($k = \ell/m_1$), then k satisfies $k = \Omega(\log \Delta/m)$ for constant s. Finally, s satisfies $s = \Omega(\Delta^{1/mk})$ for constant k.

4.2. Total completion time

Finally, we mention some results on minimizing the total completion time using resource augmentation. We begin by defining an algorithm which can use both resource augmentation on the speed and on the number of machines.

Algorithm (WAIT). We give an algorithm that is based on alpha points. The algorithm works as follows. In the background we run an optimal preemptive algorithm OPT on one machine (for the record). A job becomes eligible when α of it is completed by OPT. Eligible jobs are run in the order they become allowed, as soon as the machine (or a machine, in case there are more machines) becomes available.

Theorem 7. *The competitive ratio for the problem of minimizing the total completion time on a single machine, using resource augmentation on the speed, is* $1 + 1/s$.

Proof. We analyze WAIT in this environment. Consider a job sequence σ. Denote the preemptive optimal schedule for σ by ϕ and consider a job J. Define $W(J) = C_\phi(J) - P(J)$. Then J becomes eligible no later than at time $W(J) + \alpha P(J)$. At this time, the unprocessed parts of all the jobs that became eligible earlier have total size at most $W(J)/\alpha$. It takes $W(J)/(\alpha s)$ time to complete them. Hence J completes at the latest at time $W(J) + \alpha P(J) + W(J)/(\alpha s) + P(J)/s \leqslant (1 + 1/(\alpha s))W(J) + (\alpha + 1/s)P(J)$, which is at most $(1 + 1/s)(W(J) + P(J)) = (1 + 1/s)C_\phi(J)$ if we choose $\alpha = 1$. Since this holds for every job J, we are done.

To show that no algorithm can do better, consider the following job sequence. A job of size 1 arrives at time 0. If an algorithm A starts to run it on or after time 1, its completion time is at least $1 + 1/s$ and we are done. Otherwise, as soon as A starts it, N jobs of size 0 arrive. OPT runs these jobs before the job of size 1. Letting N increase without bound, we get the desired competitive ratio. □

Theorem 8. WAIT *has a competitive ratio of* $1 + 1/\sqrt{\ell}$ *for the problem of minimizing the total completion time using resource augmentation on the number of machines.*

Proof. We use the same definitions as in the previous proof. We have the same bound on when J becomes eligible. In the current case, it takes $W/(\alpha\ell)$ time to complete the other jobs if they are balanced over the machines; if they are not balanced, some machine is available earlier. Thus J completes at the latest at time $W + \alpha P(J) + W/(\alpha\ell) + P(J) \leqslant (1 + 1/(\alpha\ell))W + (1 + \alpha)P(J)$ which is at most $(1 + 1/\sqrt{\ell})C_\phi(J)$ for $\alpha = 1/\sqrt{\ell}$. □

The above shows that augmentation in the number of machines lets the competitive ratio tend to 1 as the number of machines grows. It is not clear whether the competitive ratio reaches the value 1 for some fixed number of machines. We show that for 2 and 3 machines, the best competitive ratio is strictly above 1.

Theorem 9. *No algorithm for minimizing the total completion time using resource augmentation on the number of machines can have a better competitive ratio than* $(11 + \sqrt{89})/16 \approx 1.27712$ *for* $\ell = 2$, *and* 1.033526 *for* $\ell = 3$.

Proof. Suppose we have an online algorithm A with a better competitive ratio than stated in the theorem.

$\ell = 2$: Take $R = (11 + \sqrt{89})/16$. Consider the following job sequence. One job of size 1 arrives at time 0. Denote the time that A starts it by t. If $t \geqslant R - 1$, no more jobs arrive and we are done.

Otherwise, two jobs of size $R - 1$ arrive. OPT runs these before the job of size 1 and has a total cost of $3t + 5R - 4$. Denote the starting time of the first of these jobs in the schedule of A by t'. Suppose $t' > t + 2 - R$, then the second smaller job may start no earlier than time $t + 1$. A has total cost at least $2t + t' + 2R$, which is at least $R(3t + 5R - 4)$ since $t \leqslant R - 1$. We are done.

Suppose $t' \leqslant t + 2 - R$. In this case, N jobs of size 0 arrive at time t'. A completes them no earlier than at time $t' + R - 1 \leqslant t + 1$. By letting N grow without bound, this implies a competitive ratio of $(t' + R - 1)/t' \geqslant 1 + (R - 1)/(t + 2 - R) \geqslant 1 + (R - 1)/1 = R$.

$\ell = 3$: Take R equal to the smallest root of the equation $10R^3 - 55R^2 + 51R - 5 = 0$ that is larger than 1. Then $R \approx 1.033526$.

One job of size 1 arrives at time 0. A must start it at some time $t_1 \leqslant R - 1$. (If $t_1 > R - 1$, we are done immediately.) We set $x = \frac{3}{2} - R \approx 0.466474$.

At time t_1, a second job arrives of size x. By our choice of x, A must start x at some time t_2 so that $t_2 + x \leqslant t_1 + 1$. Otherwise its total cost is at least $2(t_1 + 1)$, whereas the optimal cost is only $2t_1 + 2x + 1$, and

$$\frac{2t_1 + 2}{2t_1 + 2x + 1} \geqslant \frac{2R}{2R - 1 + 2(\frac{3}{2} - R)} = R.$$

We define $y = ((R - 1)/R)(t_2 + x)$. At time t_2, two jobs of size y arrive. We have $t_2 = Ry/(R - 1) - x$.

Suppose A starts them both at or after time $t_2 + x - y$. Suppose first $t_2 > t_1 + x$. Then the optimal cost is $t_1 + x + 3t_2 + 5y + 1$, and A pays at least $t_1 + 1 + 3(t_2 + x) + y$. Using $t_2 + x \leqslant t_1 + 1$, and $t_1 + 1 \leqslant R$ we get that $y \leqslant R - 1$. The ratio is at least

$$\frac{t_1 + 1 + 3(t_2 + x) + y}{t_1 + 1 + x + 3t_2 + 5y} = \frac{t_1 + 1 + y(3(R/(R - 1)) + 1)}{t_1 + 1 - 2x + y(5 + 3(R/(R - 1)))}.$$

Since this expression is at least 1, it reaches its minimum for the maximum values of t_1 and of y. Substituting $y = R - 1$ and $t_1 = R - 1$, we get that the ratio is at least $(5R - 1)/(11R - 8) > 1.2 > R$.

If $t_2 \leqslant t_1 + x$, then depending on t_2, OPT runs the jobs of size y before or after the job of size x. The optimal cost is $\min(4t_1 + 4x + 5y + 1, 4t_2 + 7y + 2x + 1)$. Suppose $t_2 < t_1 + (x - y)/2$, then $y \leqslant ((R - 1)/(3R - 1))(2t_1 + 3x)$ and we have

$$\begin{aligned}\frac{t_1 + 1 + 3(t_2 + x) + y}{4t_2 + 2x + 7y + 1} &\geqslant \frac{4t_1 + 1 + \frac{9}{2}x - \frac{1}{2}y}{4t_1 + 1 + 4x + 5y} \geqslant \frac{\frac{15}{4} - \frac{1}{2}R - ((R - 1)/(3R - 1))(t_1 + \frac{3}{2}x)}{3 + 5((R - 1)/(3R - 1))(2t_1 + 3x)} \\ &\geqslant \frac{15 - 2R - ((R - 1)/(3R - 1))(5 - 2R)}{12 + 5((R - 1)/(3R - 1))(10 - 4R)} = R.\end{aligned}$$

(The inequalities are true using similar considerations to the previous case).

If on the other hand $t_2 \geqslant t_1 + (x - y)/2$, then we have $y \geqslant ((R - 1)/(3R - 1))(2t_1 + 3x)$ and the ratio becomes

$$\begin{aligned}\frac{t_1 + 1 + 3(t_2 + x) + y}{4t_1 + 4x + 5y + 1} &= \frac{t_1 + 1 + (3R/(R - 1) + 1)y}{4t_1 + 1 + 4x + 5y} \\ &\geqslant \frac{t_1 + 1 + ((4R - 1)/(R - 1))((R - 1)/(3R - 1))(2t_1 + 3x)}{4t_1 + 1 + 4x + 5((R - 1)/(3R - 1))(2t_1 + 3x)}) \\ &\geqslant \frac{R + (4R - 1)/(3R - 1)(\frac{5}{2} - R)}{3 + 5(R - 1)/(3R - 1)(\frac{5}{2} - R)} = R.\end{aligned}$$

This shows that A must start one of the jobs of size y at some time $t_3 \leqslant t_2 + x - y$. At time t_3, N jobs of size 0 arrive. A can only start these jobs after it completes the job of size y, implying a competitive ratio of

$$\frac{t_3 + y}{t_3} \geqslant \frac{t_2 + x}{t_2 + x - y} = \frac{t_2 + x}{t_2 + x - ((R - 1)/R)(t_2 + x)} = R$$

if we let N tend to infinity. □

We conjecture that the competitive ratio is greater than 1 for all values of m, and that it approaches 1 in a rate depending super-exponentially on $1/m$. However, this problem remains open.

5. Conclusions and open problems

We have presented an algorithm for minimizing the flow time on ℓ identical machines with competitive ratio $O(\min(\Delta^{1/\ell}, n^{1/\ell}))$ against an optimal off-line algorithm on a single machine, and we have shown a lower bound of $\Omega(\min(n^{1/\ell}, \Delta^{1/\ell})/(12\ell)^{\ell})$ on the competitive ratio of any algorithm, even against an adversary on one machine. For every constant ℓ, this gives an exact trade-off between the amount of resource augmentation and the number of on-line machines.

An interesting remaining open problem is to find an algorithm which is optimally competitive against an off-line algorithm on a single machine for any ℓ.

For the problem of total completion time we showed an algorithm whose competitive ratio decreases and tends to 1 as ℓ grows. It is interesting to find out whether a fixed value of ℓ can already give competitive ratio of 1, or otherwise, to find out how fast the best competitive ratio increases as a function of ℓ.

Acknowledgements

We thank Kirk Pruhs for helpful discussions.

References

[1] B. Awerbuch, Y. Azar, S. Leonardi, O. Regev, Minimizing the flow time without migration, in: Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 198–205.
[3] M.L. Dertouzos, A.K.-L. Mok, Multiprocessor on-line scheduling of hard-real-time tasks, IEEE Trans. Software Engrg. 15 (1989) 1497–1506.
[4] A. Goel, M.R. Henzinger, S. Plotkin, E. Tardos, Scheduling data transfers in a network and the set scheduling problem, in: Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999.
[5] J.A. Hoogeveen, A.P.A. Vestjens, Optimal on-line algorithms for single-machine scheduling, in: Proceedings of the Fifth International Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, Springer, Berlin, 1996, pp. 404–414.
[6] B. Kalyanasundaram, K. Pruhs, Speed is as powerful as clairvoyance, J. Assoc. Comput. Mach. 47 (4) (2000) 214–221.
[7] H. Kellerer, T. Tautenhahn, G.J. Woeginger, Approximability and nonapproximability results for minimizing total flow time on a single machine, in: Proceedings of the 28th ACM Symposium on the Theory of Computing, 1996, pp. 418–426.
[8] S. Leonardi, D. Raz, Approximating total flow time on parallel machines, in: Proceedings of the 29th ACM Symposium on the Theory of Computing, 1997, pp. 110–119, (to appear in Journal of Computer and System Sciences).
[9] C.A. Philips, C. Stein, E. Torng, J. Wein, Optimal time-critical scheduling via resource augmentation, Algorithmica 32 (2) (2002) 163–200.