THEORY OF COMPUTING, Volume 12 (14), 2016, pp. 1–14
www.theoryofcomputing.org

SPECIAL ISSUE: APPROX-RANDOM 2015

Minimizing Maximum Flow-Time on Related Machines

Nikhil Bansal* Bouke Cloostermans

Received November 27, 2015; Revised June 28, 2016; Published September 14, 2016

Abstract: We consider the online problem of minimizing the maximum flow-time on related machines. This is a natural generalization of the extensively studied makespan minimization problem to the setting where jobs arrive over time. Interestingly, natural algorithms such as Greedy or Slow-fit that work for the simpler identical machines case or for makespan minimization on related machines, are not $O(1)$-competitive. Our main result is a new $O(1)$-competitive algorithm for the problem. Previously, $O(1)$-competitive algorithms were known only with resource augmentation, and in fact no $O(1)$-approximation was known even in the offline model.

ACM Classification: F.2.2

AMS Classification: 68W25

Key words and phrases: related machines scheduling, maximum flow-time minimization, online algorithm, approximation algorithm

1 Introduction

Scheduling a set of jobs on machines to optimize some quality of service measure is one of the most well studied problems in computer science. A very natural measure of service received by a job is the flow-time, defined as the amount of time the job spends in the system. In particular, if a job j arriving at

A preliminary version of this paper appeared in the Proceedings of the 18th Internat. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'15), 2015, pp. 85–95 [7].

*Supported by NWO grant 639.022.211 and ERC consolidator grant 617951.

 DOI: 10.4086/toc.2016.v012a014

time r_j completes its processing at time C_j, then its flow-time F_j is defined as $C_j - r_j$, i. e., its completion time minus its arrival time. Over the last few years, several variants of flow-time related problems have received a lot of attention: on single and multiple machines, in online or offline setting, with or without resource augmentation, in weighted or unweighted setting and so on. Also different objectives have been studied, such as total flow-time, ℓ_p norms of flow-time and stretch. We refer the reader to [16, 13, 12, 4] for a survey of some of these results.

In this paper we focus on the objective of minimizing the maximum flow-time. This is desirable when we want to guarantee that *each* job has a small delay. Maximum flow-time is also a very natural generalization of the minimum makespan or the load-balancing problem, that has been studied extensively (see, e. g., [5, 9, 1] for a survey). In particular, if all jobs have identical release times, then the maximum flow-time value is precisely equal to the makespan. Minimizing the maximum flow-time is also related to deadline scheduling problems. In particular, the maximum flow-time is at most D if and only if each job j completes by $r_j + D$. Moreover, note that arbitrary deadlines d_j can be modeled by considering the weighted version of maximum flow-time.[1]

Known results for maximum flow-time. For a single machine, it is easy to see that First In First Out (FIFO) is an optimal (online) algorithm for minimizing the maximum flow-time. For identical multiple machines, Mastrolilli [15] showed that the GREEDY algorithm, which dispatches the incoming job to the least loaded machine is $3 - 2/m$-competitive, where m is the number of machines. They also showed that this bound is tight for GREEDY. If jobs can be preempted and migrated (moved from one machine to another), [2] gave a 2-competitive algorithm.

A systematic investigation of the problem for various machine models was initiated recently by Anand et al. [3]. In the related machines model each machine i has speed s_i, and processing job j on machine i takes $p_{ij} = p_j/s_i$ units of time, where p_j is the size of job j. In the more general unrelated machines model, p_{ij} can be completely arbitrary.

Among other results, Anand et al. [3] gave a $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive algorithm for the unrelated machine case, for any given $\varepsilon > 0$. Here the online algorithm can process $1+\varepsilon$ units of volume per time step, but is compared to an offline optimum that does not have this extra *resource augmentation* [14, 16]. They also showed that in the unrelated setting any online algorithm without resource augmentation must be $\Omega(m)$ competitive. For the weighted maximum flow-time objective, they gave a $(1+\varepsilon)$-speed, $O(1/\varepsilon^3)$-competitive algorithm for the related machines setting, and showed that no $O(1)$-speed, $O(1)$-competitive algorithm exists in the unrelated setting.

A natural question that remains is the complexity of the problem for related machines. Is there an $O(1)$-competitive algorithm for the related machines setting, without using resource augmentation or migration?

This question is particularly intriguing as it is not at all clear what the right algorithm should be [2]. In fact, no $O(1)$-approximation is known even in the offline model. One issue is that the natural SLOW-FIT algorithm (described in the following section), that is $O(1)$-competitive for makespan minimization (even when the jobs are temporary and have unknown durations [6]), is not $O(1)$-competitive for maximum flow-time (Lemma 2.1 below). The algorithm of [3] for weighted maximum flow-time with resource

[1] In deadline scheduling however, the deadlines are typically considered fixed and the focus is on maximizing the throughput, that is, maximizing the number of deadlines met.

augmentation is also a variant of SLOW-FIT. Recently, [8] obtained an $O(\log n)$ approximation for minimizing maximum flow-time on unrelated machines, where n is the number of machines. However their techniques do not seem to give anything better for the related machines setting either.

Our main result is the following.

Theorem 1.1. *There is a* 13.5*-competitive algorithm,* DOUBLE-FIT*, for minimizing maximum flow-time on related machines.*

This also gives the first $O(1)$-approximation for the offline problem. We also show that no such result is possible in the weighted case (without resource augmentation), and give an $\Omega(W)$ lower bound on the competitive ratio where W is the maximum to minimum weight ratio.

High-level approach. There are two competing trade-offs while scheduling on related machines. On one hand the algorithm should keep as many machines busy as possible, otherwise load might accumulate and delay future jobs. We could end up with a large backlog that is impossible to get rid of without resource augmentation. On the other hand, the algorithm should keep fast machines empty for processing large jobs that might arrive later. In particular, fast machines are a scarce resource that should not be wasted on processing small jobs unnecessarily. It is instructive to consider the lower bounds in Section 2, where both SLOW-FIT and GREEDY are shown to perform badly due to these opposite reasons.

To get around this, we design an algorithm that combines the good properties of both SLOW-FIT and GREEDY. In particular, the algorithm uses a two-phase strategy while assigning jobs to machines at each step. First, the jobs are spread out to ensure that machines are busy as much as possible. Once machines are *saturated*, the algorithm shifts into a *Slow-fit* mode, which ensures that small jobs do not unnecessarily go on fast machines.

The key difficulty in the analysis is to control how the two phases interact with each other. To do this, we maintain two invariants that capture the dynamics of the algorithm, and control how much the online algorithm's load on a subset of machines deviates from the offline algorithm's load on those machines. The main part of the argument is to show inductively that these invariants are maintained over time.

Notation and formal problem description. There are m machines indexed by non-decreasing order of speeds $s_1 \leq s_2 \leq \cdots \leq s_m$. The processing requirement of job j is p_j, and it requires time p_j/s_i on machine i. We will call p_j the size of j, and p_j/s_i its load on machine i. Jobs arrive online over time and p_j is known immediately upon its release time r_j. The goal is to find a schedule that minimizes the maximum flow-time, and we assume that a job cannot be migrated from one machine to another. We use Opt to denote some fixed optimum offline schedule, and also to denote the value of this solution.

2 Lower bounds on SLOW-FIT and GREEDY

SLOW-FIT. Algorithm SLOW-FIT takes as input a threshold F_{opt} (the current guess on optimum), and dispatches every incoming job to the slowest possible machine while keeping the load below F_{opt}. If the jobs cannot be feasibly scheduled on any machine, the algorithm fails and the threshold is doubled.

Lemma 2.1. SLOW-FIT *has a competitive ratio of* $\Omega(m)$.

Proof. We describe an instance where the threshold F_{opt} keeps doubling until $F_{\text{opt}} > m$ even though $\text{Opt} = 2$.

There are m identical machines (but we arbitrarily order them from slow to fast). Next, we assume that $F_{\text{opt}} \geq 2$, which can be achieved by giving $2m$ unit-size jobs initially at $t = 0$.

The instance consists of two alternating phases: a buildup phase, and an overflow phase. In the buildup phase we build up load on slow machines to $F_{\text{opt}} - 1$. In the overflow phase we release a burst of jobs, causing F_{opt} to double. This happens until F_{opt} exceeds m.

We start with the buildup phase. At each time step $t \geq 2$, m unit-size jobs arrive. SLOW-FIT will keep machine m empty until all machines $1, \dots, m-1$ have load F_{opt}. This means at most $m-1$ jobs get processed until some time t_0 at which all machines $1, \dots, m-1$ have load F_{opt}. Once these machines have high load we move to the overflow phase. At time $t_0 + 1$, $2m$ unit-size jobs arrive. As there is at most $m - 1 + F_{\text{opt}}$ total space available below F_{opt}, these jobs cannot be scheduled feasibly, causing F_{opt} to double. No jobs arrive at time $t_0 + 2$ (which will allow Opt to clear its machines). At time $t_0 + 3$ we continue with the next buildup phase. This process continues until $F_{\text{opt}} \geq m + 1$, so that there is $m - 1 + F_{\text{opt}} \geq 2m$ space available for the overflow phase.

We claim that Opt has a maximum flow-time of 2. This value can be achieved by distributing the incoming jobs equally over all machines (i. e., it does GREEDY). During the buildup phase every machine receives exactly one job per time unit so that all jobs finish after unit time. During the overflow phase every machine receives exactly 2 jobs so that when the next buildup phase starts all machines are empty. □

Intuitively, SLOW-FIT unnecessarily builds up load on slow machines while keeping the fast machines empty, and cannot recover if there is small burst of jobs.

GREEDY. When a job j arrives, GREEDY dispatches j to the machine that minimizes the flow-time of j (assuming FIFO order). Ties are broken arbitrarily. The following bound is well-known [11], but we sketch it here for completeness. The idea is that GREEDY puts too many slow jobs on fast machines, which causes problems when large jobs arrive.

Lemma 2.2. GREEDY *has a competitive ratio of* $\Omega(\log m)$.

Proof. Consider an instance with k groups of machines where group G_i contains 2^{2k-2i} machines of speed 2^i. Thus, the total processing power of group G_i is equal to $S_i = 2^{2k-i}$. The processing power of groups $G_1, \dots, G_k$ combined is equal to

$$P_i = \sum_{i'=i}^{k} 2^{2k-i'} = 2^{2k-i+1} - 2^k < 2S_i\,.$$

k sets of jobs arrive, all at time 0, but in increasing order of size. For all $i = 1, \dots, k$, the set J_i contains $P_i/2^i$ jobs of size 2^i. We claim that GREEDY will distribute exactly $2^{i-i'}$ jobs from J_i to each machine in $G_{i'}$ for $i \leq i' \leq m$. This will cause the load on all these machines to increase by exactly 1.

We use an inductive argument. Just before the jobs from set J_i are distributed, for all $i \leq i' \leq m$, each machine in group G'_i has a load of exactly $i - 1$. For all $1 \leq i' \leq i - 1$, each machine in group G'_i has a load of exactly i'. GREEDY does not use groups $G_1, \dots, G_{i-1}$ to process jobs from J_i, since distributing a job from J_i to a machine in group $G_{i'}$ ($i' < i$) would result in a flow time of $i' + 2^{i-i'} > i$. As J_i contains jobs with a total size of P_i, GREEDY will assign a load of 1 to each machine in $G_i, \dots, G_m$.

After scheduling k sets of jobs, the single machine in G_k has received a load of k. However, optimum can schedule the i-th batch of jobs on group i machines, incurring a maximum load of 2 (i. e., it does SLOW-FIT with threshold 2). □

3 The algorithm DOUBLE-FIT

We describe our algorithm, denoted by DOUBLE-FIT hereafter. DOUBLE-FIT takes as input a parameter F_{opt}, which is supposed to be our estimate of Opt. By a slight variation on the doubling trick that loses an additional factor of 1.5 (see Section 3.4), we will assume henceforth that $F_{\text{opt}} \in [\text{Opt}, 1.5\,\text{Opt})$.

We divide time into intervals I_k of size $3F_{\text{opt}}$ as $I_k = [3(k-1)F_{\text{opt}}, 3kF_{\text{opt}})$. We refer to time $3kF_{\text{opt}}$ as the k-th epoch. For each $k = 1, 2, \dots$, DOUBLE-FIT batches the jobs that arrive during I_k and schedules them at epoch k using the algorithm in Figure 1. We use $[i:m]$ to denote the machines $i, \dots, m$. If the total remaining size of jobs on machine i is $w(i)$ at time t, we say that it has load $w(i)/s_i$.

1. Let J denote the set of jobs arriving during I_k.
2. Partition jobs in J into classes $J_1, \dots, J_m$, where each job j is in class J_i with the smallest index i such that $p_j \le s_i \cdot F_{\text{opt}}$.
3. **For** $i = m, m-1, \dots, 1$
4. Consider the jobs j in J_i in arbitrary order and assign them as follows.
5. **(Saturation Phase) If** some machine in $[i:m]$ is loaded below $3F_{\text{opt}}$
6. dispatch j to the slowest such machine.
7. **(Slow-fit Phase) Else** dispatch j to the slowest machine in $[i:m]$
8. such that its load stays below $6F_{\text{opt}}$.
9. **If** no such machine exists return FAIL.

Figure 1: Algorithm DOUBLE-FIT for the epoch k.

Description. First, DOUBLE-FIT classifies the jobs arriving during I_k depending on the slowest machine on which they have load no larger than F_{opt}. Note that as $F_{\text{opt}} \ge \text{Opt}$, if job j is put in class J_i, then Opt cannot schedule job j onto a machine smaller than i either.

DOUBLE-FIT considers jobs from classes J_m down to J_1 (this ordering will be used crucially). Each class is scheduled in two phases. In the saturation phase, when scheduling a job j from class i, it checks if there is some machine in $[i:m]$ with load less than $3F_{\text{opt}}$. If so, j is dispatched to the slowest such machine. If no such machine exists, the algorithm enters the Slow-fit phase (for class J_i), and performs SLOW-FIT for class J_i on machines $[i:m]$ with threshold $6F_{\text{opt}}$.

3.1 Analysis

Our goal in this section is to show the following result.

Theorem 3.1. *If* $F_{\text{opt}} \geq \text{Opt}$*, then* DOUBLE-FIT *never fails.*

This directly implies Theorem 1.1 as follows. Each job spends at most $3F_{\text{opt}}$ time waiting to be assigned, and at most $6F_{\text{opt}}$ on its designated machine, thus the flow-time of any job is at most $9F_{\text{opt}}$. As $F_{\text{opt}} \leq 1.5\,\text{Opt}$ by the doubling trick, this implies a competitive ratio of 13.5

For the purpose of analysis, it will be convenient to consider a *restricted* Opt that also batches jobs and dispatches the jobs arriving during I_k at epoch k. Note that such a restricted algorithm has objective at most $3F_{\text{opt}} + \text{Opt} \leq 4F_{\text{opt}}$ (as we can take the original schedule and delay every job by $3F_{\text{opt}}$). To prove Theorem 3.1, we will in fact prove the following stronger result: DOUBLE-FIT never fails for any instance where the restricted Opt has value at most $4F_{\text{opt}}$.

The invariants. Fix an epoch k. Let $A_i(k)$ and $B_i(k)$ denote the total remaining size of jobs on machines $[i:m]$ in DOUBLE-FIT's schedule just before and just after all the jobs from interval I_k are dispatched respectively. Similarly, let $A_i^{\text{opt}}(k)$ be the total size of jobs remaining on machines $[i:m]$ in Opt's schedule before dispatching the jobs which arrived during interval I_k. We define $B_i^{\text{opt}}(k)$ slightly more carefully. Let $L_i(k)$ be the load on machine i in Opt's schedule after scheduling jobs at epoch k. Then we define $B_i^{\text{opt}}(k)$ as

$$B_i^{\text{opt}}(k) = \sum_{i'=i}^{m} \max\{L_{i'}, 3F_{\text{opt}} \cdot s_{i'}\}\,.$$

We will show that the following two invariants hold at each epoch k.

$$A_i(k) \leq A_i^{\text{opt}}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}\,, \tag{3.1}$$

$$B_i(k) \leq B_i^{\text{opt}}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}\,. \tag{3.2}$$

Roughly speaking, invariants (3.1) and (3.2) show that the load on any suffix of DOUBLE-FIT's machines stays close to Opt's load on those machines, both before and after the jobs are dispatched at epoch k. We will prove that (3.1) and (3.2) hold by a careful induction over i and k.

Before we prove these invariants, let us first see why they imply Theorem 3.1.

Proof of Theorem 3.1. Consider a fixed epoch k. As (restricted) Opt has maximum flow-time at most $4F_{\text{opt}}$, for each i it must hold that $B_i^{\text{opt}}(k) \leq 4F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}$. Thus by (3.2) it follows that $B_i(k) \leq 5F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}$ for each i. Choosing $i = m$, this implies that DOUBLE-FIT never loads machine m above $5F_{\text{opt}}$ and thus never fails (as machine m always has room for an additional job). □

Proving the invariants. The strategy for proving that (3.1) and (3.2) hold at all epochs k will be to show the following two lemmas.

Lemma 3.2. *If at epoch k,* (3.1) *holds for all machines, then* (3.2) *also holds for all machines.*

The next step will be to relate the conditions at epochs k and $k+1$.

Lemma 3.3. *If at epoch k, (3.2) holds for all machines, then (3.1) also holds for all machines at epoch $k+1$.*

As (3.1) trivially holds for $k=0$ (as $A_i(0)=A_i^{\text{opt}}(0)=0$ for all i), applying Lemma 3.2 and Lemma 3.3 alternately implies that (3.1) and (3.2) hold for all k.

3.2 Proof of Lemma 3.2

We first show that DOUBLE-FIT is conservative in scheduling small jobs on fast machines.

Lemma 3.4. *Let $i_1 < i_2$. If some job j of class i_1 is dispatched by* DOUBLE-FIT *to machine i_2 during the saturation phase (i. e., using threshold $3F_{\text{opt}}$), then* all *jobs of class i for $i_1 < i \leq i_2$ are also dispatched during the saturation phase.*

Proof. Consider the state of DOUBLE-FIT's machines just before j was dispatched. As j is dispatched to machine i_2 during the saturation phase, the load on i_2 must be below $3F_{\text{opt}}$ at that point. As jobs of class i for $i_1 < i \leq i_2$ were considered before class i_1-jobs, the load on i_2 was also below $3F_{\text{opt}}$ after scheduling class i jobs, and thus DOUBLE-FIT must have never switched to the Slow-fit phase while considering class i. □

Next we define the notion of *separated* machines, which will play a crucial role in the analysis.

Definition 3.5. Machines i_1 and i_2 ($i_1 < i_2$) are *separated* at epoch k if DOUBLE-FIT dispatched no jobs from classes $[1:i_1]$ onto machines $[i_2:m]$ at epoch k.

The following lemma shows that if two consecutive machines are separated, it is easy to relate epochs k and $k+1$.

Lemma 3.6. *If machines $i-1$ and i are separated at epoch k, then (3.1) implies (3.2) for machine i. Moreover this trivially holds for machine $i=1$.*

Proof. As machines $i-1$ and i are separated at epoch k, no jobs from class $[1:i-1]$ were dispatched to machines $[i:m]$ at epoch k. Thus

$$B_i(k) = A_i(k) + \sum_{i'=i}^{m} |J_{i'}|\,, \tag{3.3}$$

where $|J_i|$ represents the total size of all jobs in J_i.

As jobs from J_i cannot be scheduled on machines $[1:i-1]$ in an optimal schedule, we also obtain

$$B_i^{\text{opt}}(k) \geq A_i^{\text{opt}}(k) + \sum_{i'=i}^{m} |J_i|\,. \tag{3.4}$$

This implies that

$$B_i(k) = A_i(k) + \sum_{i'=i}^{m} |J_{i'}| \leq A_i(k) + B_i^{\text{opt}}(k) - A_i^{\text{opt}}(k) \leq B_i^{\text{opt}}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}\,, \tag{3.5}$$

where the last step follows by our assumption that (3.1) holds for (i,k).

Finally for $i = 1$, we observe that both (3.3) and (3.4) hold with equality, and hence the result holds trivially. □

We now have all the tools we need to prove Lemma 3.2.

Proof of Lemma 3.2. We use induction over i in the order of larger to smaller i. In particular, to prove that (3.2) holds for some pair (i,k), we assume that (3.1) holds for all (i',k) and that (3.2) holds for all (i',k) with $i' > i$. As the base case note that this is vacuously true for $i = m+1$ (as all relevant quantities are 0).

We consider three cases depending on how DOUBLE-FIT assigns jobs from classes $[1 : i-1]$ to machines $[i : m]$.

1. *No jobs from class* $[1 : i-1]$ *were dispatched to machines* $[i : m]$. In this case, machines $i-1$ and i are separated and (3.2) follows from Lemma 3.6.

2. *Jobs from classes* $[1 : i-1]$ *are only dispatched to machines* $[i : m]$ *during the saturation phase.* Let $i_{\max} \geq i$ denote the smallest index such that machines $i-1$ and $i_{\max}+1$ are separated (if no such machine exists, set $i_{\max} = m$). By the inductive hypothesis, we can assume that (3.2) holds for $i_{\max}+1$. In the case where $i_{\max} = m$, this holds vacuously. As jobs from classes $[1 : i-1]$ are assigned to $[i : m]$ (and hence to $i_{\max}$) during the saturation phase, Lemma 3.4 implies that all jobs in classes $[i : i_{\max}]$ were also dispatched during the saturation phase, which implies that all machines $[i : i_{\max}]$ are loaded below $4F_{\text{opt}}$. This gives us the following.

$$\begin{aligned}
B_i(k) &\leq 4F_{\text{opt}} \sum_{i'=i}^{i_{\max}} s_{i'} + B_{i_{\max}+1}(k) \\
&\leq 4F_{\text{opt}} \sum_{i'=i}^{i_{\max}} s_{i'} + B^{\text{opt}}_{i_{\max}+1}(k) + F_{\text{opt}} \sum_{i'=i_{\max}+1}^{m} s_{i'} \\
&= 3F_{\text{opt}} \sum_{i'=i}^{i_{\max}} s_{i'} + B^{\text{opt}}_{i_{\max}+1}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \\
&\leq B^{\text{opt}}_{i}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}\,,
\end{aligned}$$

where the second inequality follows from the inductive hypothesis for machine $i_{\max}+1$.

3. *Some job j from class* $[1 : i-1]$ *was dispatched to machines* $[i : m]$ *during Slow-fit phase (using threshold* $6F_{\text{opt}}$*).* We assume that $i > 1$, otherwise the result follows from case 1. Let $i_{\min} < i$ denote the largest index such that machines $[i_{\min} : i-1]$ have load more than $5F_{\text{opt}}$ and machine $i_{\min}-1$ has load at most $5F_{\text{opt}}$. If no such machine exists, set $i_{\min} = 1$. $i_{\min}$ is well-defined as $i > 1$ and machine $i-1$ must have load more than $5F_{\text{opt}}$ as job j from class $[1 : i-1]$ was assigned to a machine in $[i : m]$ during the Slow-fit phase.

Claim 3.7. *Machines* $i_{\min}-1$ *and* $i_{\min}$ *are separated or* $i_{\min} = 1$.

Proof. This is trivially true if $i_{\min} = 1$.

If $i_{\min} > 1$, suppose that some job j' from class $[1 : i_{\min} - 1]$ was dispatched to machines $[i_{\min} : m]$. Now j' cannot be dispatched during the Slow-fit phase as this would imply that the load on $i_{\min} - 1$ was more than $5F_{\text{opt}}$, which contradicts the choice of $i_{\min}$.

So all jobs in $[1 : i_{\min} - 1]$ that were assigned to $[i_{\min} : m]$ must have been assigned during the saturation phase. Let $i' \geq i_{\min}$ denote the largest index where such a job is assigned. By Lemma 3.4, it must be that all machines $[i_{\min} : i']$ were assigned load during the saturation phase and must have load at most $4F_{\text{opt}}$. This contradicts that $i_{\min}$ has load more than $5F_{\text{opt}}$. □

By Lemma 3.6 applied to $i_{\min}$, we get that (3.2) holds for machine $i_{\min}$ and thus

$$B_{i_{\min}}(k) \leq B^{\text{opt}}_{i_{\min}}(k) + F_{\text{opt}} \sum_{i'=i_{\min}}^{m} s_{i'} \,. \tag{3.6}$$

Furthermore, by choice of $i_{\min}$ all the machines in $[i_{\min} : i - 1]$ are loaded above $5F_{\text{opt}}$. This implies that

$$B_i(k) \leq B_{i_{\min}}(k) - 5F_{\text{opt}} \sum_{i'=i_{\min}}^{i-1} s_{i'} \,. \tag{3.7}$$

As every machine is loaded below $4F_{\text{opt}}$ in an optimal schedule, we also have

$$B^{\text{opt}}_{i_{\min}}(k) \leq B^{\text{opt}}_{i}(k) + 4F_{\text{opt}} \sum_{i'=i_{\min}}^{i-1} s_{i'} \,. \tag{3.8}$$

Adding (3.6) and (3.7) we obtain that

$$\begin{aligned} B_i(k) &\leq B^{\text{opt}}_{i_{\min}}(k) + F_{\text{opt}} \sum_{i'=i_{\min}}^{m} s_{i'} - 5F_{\text{opt}} \sum_{i'=i_{\min}}^{i-1} s_{i'} \\ &\leq B^{\text{opt}}_{i}(k) + F_{\text{opt}} \sum_{i'=i_{\min}}^{m} s_{i'} - F_{\text{opt}} \sum_{i'=i_{\min}}^{i-1} s_{i'} && \text{by (3.8)} \\ &= B^{\text{opt}}_{i}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \,, \end{aligned}$$

which implies that (3.2) holds for i. □

3.3 Proof of Lemma 3.3

We now prove Lemma 3.3, which is relatively easier.

Proof of Lemma 3.3. We will apply induction over i (in decreasing order of machines). Consider epoch k. We assume that (3.2) holds for all i' at epoch k, and that (3.1) holds for all $i' > i$ at epoch $k+1$. For the base case of $i = m+1$ the lemma follows trivially since all the relevant quantities are 0.

Consider some machine i. We consider two cases depending on the load of machine i after the jobs were dispatched at epoch k.

1. *Machine i has load at most $4F_{\text{opt}}$ after epoch k, i. e., $B_i(k) - B_{i+1}(k) \le 4F_{\text{opt}} \cdot s_i$.* At epoch $k+1$ before the jobs arriving during interval I_{k+1} are dispatched, the load of machine i will be at most F_{opt}. Thus we have that

$$\begin{aligned} A_i(k+1) &\le A_{i+1}(k+1) + F_{\text{opt}} \cdot s_i \\ &\le A_{i+1}^{\text{opt}}(k+1) + F_{\text{opt}} \sum_{i'=i+1}^{m} s_{i'} + F_{\text{opt}} \cdot s_i \\ &\le A_i^{\text{opt}}(k+1) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \,. \end{aligned}$$

Here the second inequality follows by the inductive hypothesis for machine $i+1$, and the third inequality follows as $A_i^{\text{opt}}(k+1)$ is non-decreasing as i decreases.

2. *Machine i is loaded above $4F_{\text{opt}}$ after epoch k, i. e., $B_i(k) - B_{i+1}(k) > 4F_{\text{opt}} \cdot s_i$.* In this case, some job j must have been dispatched to machine i during the Slow-fit phase. This only happens if j could not be dispatched during the saturation phase. In particular, this implies that all the machines $[i:m]$ (which is surely a subset of machines where j could have been scheduled) were loaded above $3F_{\text{opt}}$. So the total size of jobs on all machines $[i:m]$ decreases by exactly $3F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}$ during interval I_{k+1}.

   Thus we have that
$$A_i(k+1) = B_i(k) - 3F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \,. \tag{3.9}$$

   Similarly, as Opt can complete at most $3F_{\text{opt}} \sum_{i'=i}^{m} s_{i'}$ on machines $[i:m]$ during this interval, we have
$$A_i^{\text{opt}}(k+1) = B_i^{\text{opt}}(k) - 3F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \,. \tag{3.10}$$

   As (3.2) holds for each i at epoch k, we obtain that

$$\begin{aligned} A_i(k+1) &\le B_i(k) - 3F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \\ &\le B_i^{\text{opt}}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} - 3F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} && \text{by (3.2)} \\ &= A_i^{\text{opt}}(k+1) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \,, \end{aligned}$$

   and hence (3.1) holds for i at epoch $k+1$, which completes the proof. □

3.4 Removing the assumption of knowledge of Opt

We describe a variant of the standard doubling trick where we increase the online estimate of Opt by only 1.5 times at each step.

Consider some epoch k where the algorithm first fails with the current guess of F_{opt}. It must be that (3.2) does not hold. In particular, (3.1) holds at epoch k as (3.2) holds at $k-1$. Now, Lemma 3.2 implies that $F_{\text{opt}} < \text{Opt}$. We then abort epoch k, and do not schedule any jobs. Instead, we set $F'_{\text{opt}} = 1.5F_{\text{opt}}$ and redefine the new epoch to be the time $(k-1)F_{\text{opt}} + 3F'_{\text{opt}}$. Note that between these epochs $4.5F_{\text{opt}}$ time passes, so at the next epoch the load on all machines in the schedule of DOUBLE-FIT will be at most $6F_{\text{opt}} - 3F'_{\text{opt}} = 1.5F_{\text{opt}} = F'_{\text{opt}}$. This implies that for all i

$$A_i(k) \leq F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \leq A_i^{\text{opt}}(k) + F_{\text{opt}} \sum_{i'=i}^{m} s_{i'} \,.$$

The crucial point is that (3.1) holds for all machines i at this new epoch irrespective of the workload of the new restricted Opt (with parameter F'_{opt}). Thus, (3.2) holds if $F_{\text{opt}} \geq \text{Opt}$ and DOUBLE-FIT proceeds as normal.

4 Other lower bounds

We also show simple (but strong) lower bounds for weighted maximum flow-time and maximum stretch.

Lemma 4.1. *Any algorithm for minimizing maximum weighted flow-time on identical machines must have a competitive ratio of $\Omega(W)$, where W is the ratio between the largest and smallest weight.*

Proof. Consider the following instance on 2 machines. At time $t = 0$ we receive 2 jobs of size w with weight 1. Now, any algorithm has three options: (i) it commits to assigning both jobs to the same machine by time $w/2$, (ii) it commits to assigning both jobs on different machines by time $w/2$, or (iii) it has not committed at least one job to a machine yet by time $w/2$. Thus, the algorithm has not started processing this job yet.

In all three cases, we show that the algorithm will end up trailing by at least $\Omega(w)$ volume behind an optimal schedule. In option (i), at least w volume remains at time $t = w$, while Opt assigned one job to each machine and thus has no jobs left. In option (ii), another $3w/2$-sized job with weight 1 arrives at time $w/2$, so that by time $2w$ one of the algorithm's machines has load of at least $w/2$ (or the entire job still needs to be processed). Opt initially distributed both jobs to the same machine so it can distribute this job to its empty machine and has no jobs left by time $2w$. In option (iii), Opt distributes the jobs to different machines so that by time $w/2$ we are trailing $w/2$ volume.

Once we trail w volume behind optimum, at every unit time step we receive 2 unit-size jobs of weight w. If the trailing jobs are ever to be finished, at least $w/2$ delay is incurred on the weight w jobs, implying an objective value of $\Omega(w^2)$. Opt on other hand has value $O(w)$. □

If we remove weights in the above instance, this example also directly implies an $\Omega(S)$ lower bound on the competitive ratio for maximum stretch [3] where S is the ratio between the size of the largest and the smallest job.

We remark that a weaker lower bound of $\Omega(W^{0.4})$ for maximum weighted flow-time also follows from [10], using the analogy between delay factor and weighted maximum flow-time described in [3].

Concluding Remarks

Note that our algorithm DOUBLE-FIT is not immediate-dispatch, i. e., it does not dispatch a job to a machine immediately upon arrival. We are unable to extend the ideas here to obtain an $O(1)$-competitive immediate-dispatch algorithm, and it is not clear to us whether such an algorithm exists. Given that in the unrelated setting, there can be no $O(1)$-speed, $O(1)$-competitive immediate-dispatch algorithm [3] (while there is a $(1+\varepsilon)$-speed, $O(1/\varepsilon)$-competitive algorithm), it would be quite interesting to resolve this question.

Acknowledgments

The authors would like to thank Shuguang Li for noticing an error in a previous version of this paper.

References

[1] SUSANNE ALBERS: Online scheduling. In *Introduction to Scheduling* (Yves Robert and Frédéric Vivien, eds.), chapter 3, pp. 51–73. Chapman and Hall/CRC Press, 2010. Version available on author's website. [doi:10.1201/9781420072747-c3] 2

[2] CHRISTOPH AMBÜHL AND MONALDO MASTROLILLI: On-line scheduling to minimize max flow time: an optimal preemptive algorithm. *Oper. Res. Lett.*, 33(6):597–602, 2005. [doi:10.1016/j.orl.2004.10.006] 2

[3] S. ANAND, KARL BRINGMANN, TOBIAS FRIEDRICH, NAVEEN GARG, AND AMIT KUMAR: Minimizing maximum (weighted) flow-time on related and unrelated machines. In *Proc. 40th Internat. Colloq. on Automata, Languages and Programming (ICALP'13)*, pp. 13–24. Springer, 2013. [doi:10.1007/978-3-642-39206-1_2] 2, 11, 12

[4] S. ANAND, NAVEEN GARG, AND AMIT KUMAR: Resource augmentation for weighted flow-time explained by dual fitting. In *Proc. 23rd Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'12)*, pp. 1228–1241. ACM Press, 2012. ACM DL. 2

[5] YOSSI AZAR: On-line load balancing. In *Online Algorithms*, volume 1442 of *LNCS*, pp. 178–195. Springer, 1998. [doi:10.1007/BFb0029569] 2

[6] YOSSI AZAR, BALA KALYANASUNDARAM, SERGE A. PLOTKIN, KIRK PRUHS, AND ORLI WAARTS: On-line load balancing of temporary tasks. *J. Algorithms*, 22(1):93–110, 1997. Preliminary version in WADS'93. [doi:10.1006/jagm.1995.0799] 2

[7] NIKHIL BANSAL AND BOUKE CLOOSTERMANS: Minimizing maximum flow-time on related machines. In *Proc. 18th Internat. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'15)*, pp. 85–95, 2015. [doi:10.4230/LIPIcs.APPROX-RANDOM.2015.85] 1

[8] NIKHIL BANSAL AND JANARDHAN KULKARNI: Minimizing flow-time on unrelated machines. In *Proc. 47th STOC*, pp. 851–860. ACM Press, 2015. [doi:10.1145/2746539.2746601, arXiv:1401.7284] 3

[9] NIV BUCHBINDER AND JOSEPH NAOR: The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009. Preliminary version in FOCS'06. [doi:10.1561/0400000024] 2

[10] CHANDRA CHEKURI, SUNGJIN IM, AND BENJAMIN MOSELEY: Online scheduling to minimize maximum response time and maximum delay factor. *Theory of Computing*, 8(7):165–195, 2012. Preliminary version in ESA'09. [doi:10.4086/toc.2012.v008a007, arXiv:0906.2048] 11

[11] YOOKUN CHO AND SARTAJ SAHNI: Bounds for list schedules on uniform processors. *SIAM J. Comput.*, 9(1):91–103, 1980. [doi:10.1137/0209007] 4

[12] NAVEEN GARG: Minimizing average flow-time. In *Efficient Algorithms, Essays dedicated to Kurt Mehlhorn on the occasion of his 60th birthday*, pp. 187–198, 2009. Preliminary version in FOCS'07. [doi:10.1007/978-3-642-03456-5_13] 2

[13] SUNGJIN IM, BENJAMIN MOSELEY, AND KIRK PRUHS: A tutorial on amortized local competitiveness in online scheduling. *ACM SIGACT News*, 42(2):83–97, 2011. [doi:10.1145/1998037.1998058] 2

[14] BALA KALYANASUNDARAM AND KIRK PRUHS: Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. Preliminary version in FOCS'95. [doi:10.1145/347476.347479] 2

[15] MONALDO MASTROLILLI: Scheduling to minimize max flow time: Off-line and on-line algorithms. *Internat. J. Foundat. Comput. Sci.*, 15(2):385–401, 2004. Preliminary version in FCT'03. [doi:10.1142/S0129054104002480] 2

[16] KIRK PRUHS, JIŘÍ SGALL, AND ERIC TORNG: Online scheduling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (Joseph Y-T. Leung, ed.), chapter 15. CRC Press, 2004. Available at CRC Press. 2

AUTHORS

Nikhil Bansal
Professor
Eindhoven University of Technology
n.bansal@tue.nl
http://www.win.tue.nl/~nikhil/

Bouke Cloostermans
Ph. D. student
Eindhoven University of Technology
b.cloostermans@tue.nl
http://www.tue.nl/en/university/departments/mathematics-and-computer-science/the-department/staff/

ABOUT THE AUTHORS

NIKHIL BANSAL is a Professor in the Department of Mathematics and Computer Science at Eindhoven University of Technology. He received his Bachelors degree in Computer Science from IIT Mumbai (1999), and obtained his Ph. D. from Carnegie Mellon University in 2003 under the guidance of Avrim Blum. He is broadly interested in theoretical computer science with focus on the design and analysis of algorithms.

BOUKE CLOOSTERMANS received his Master's degree from Eindhoven University of Technology in 2014. He is currently a Ph. D. student at the same institution under the supervision of Nikhil Bansal.