Semi-online scheduling with decreasing job sizes

Steve Seiden\textsuperscript{a,*,1}, Jiří Sgall\textsuperscript{b,c,2}, Gerhard Woeginger\textsuperscript{d,3}

\textsuperscript{a}Department of Computer Science, Louisiana State University, 298 Coates Hall, Baton Rouge, LA 70803, USA
\textsuperscript{b}Mathematical Institute, AS CR, Zitná 25, CZ-11567 Praha 1, Czech Republic
\textsuperscript{c}Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Praha, Czech Republic
\textsuperscript{d}Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria

Received 28 October 1998; received in revised form 1 July 2000

Abstract

We investigate the problem of semi-online scheduling jobs on \(m\) identical parallel machines where the jobs arrive in order of decreasing sizes. We present a complete solution for the preemptive variant of semi-online scheduling with decreasing job sizes. We give matching lower and upper bounds on the competitive ratio for any fixed number \(m\) of machines; these bounds tend to \(1 + \sqrt{3}/2 \approx 1.86303\), as the number of machines goes to infinity. Our results are also the best possible for randomized algorithms. For the non-preemptive variant of semi-online scheduling with decreasing job sizes, a result of Graham (SIAM J. Appl. Math. 17 (1969) 263–269) yields a \((4/3 - 1/(3m))\)-competitive deterministic non-preemptive algorithm. For \(m = 2\) machines, we prove that this algorithm is the best possible (it is \(7/6\)-competitive). For \(m = 3\) machines we give a lower bound of \((1 + \sqrt{37})/6 \approx 1.8046\). Finally, we present a randomized non-preemptive \(8/7\)-competitive algorithm for \(m = 2\) machines and prove that this is optimal. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Analysis of algorithms; Online algorithms; Competitive ratio; Scheduling

1. Introduction

Consider the following scheduling problem: we are confronted with a sequence of jobs with processing times \(p_1, p_2, \ldots, p_n\) that must be assigned to \(m\) machines. The load of a machine is the sum of the processing times of the jobs assigned to that machine. Our goal is to minimize the makespan, i.e., the maximum machine load. If the problem is online, then each job must be assigned without knowledge of successive jobs. (However, the order of jobs has no correlation with the time in the schedule, some future job may be assigned to start running earlier than the current one.) If the problem is semi-online with decreasing job sizes, then we know that \(p_i > p_{i+1}\) for all \(i \geq 1\). Hence, in this variant we do have some partial knowledge on the future jobs, which makes the problem easier to solve than standard online scheduling problems.
The quality of an online or semi-online algorithm \( A \) is measured by its competitive ratio, defined as the smallest number \( c \) such that for every list of jobs \( L \), \( A(L) \leq c \text{opt}(L) \), where \( A(L) \) denotes the makespan of a schedule produced by algorithm \( A \) for scheduling the list \( L \) of jobs, and \( \text{opt}(L) \) denotes the corresponding makespan of some optimal schedule.

We study semi-online scheduling with decreasing job sizes in the three main variants: preemptive, non-preemptive deterministic, and non-preemptive randomized scheduling. Note that for \( m = 1 \) the trivial deterministic non-preemptive algorithm that schedules all jobs on the single machine without introducing any idle time is optimal. Thus, we assume \( m \geq 2 \) for the rest of the paper. In the preemptive variant, a job may be preempted, i.e., split into pieces that may be spread over several machines and/or assigned to non-consecutive time slots. However, pieces of the same job must not overlap in time, even if scheduled on different machines. In other words, if a job is assigned to time slots \( (s_1, t_1], (s_2, t_2], \ldots, (s_k, t_k] \) on one or several machines, then any pair of these time slots must be disjoint. Upon arrival of each job we have to assign it to particular time slots, which in principle may create idle time that is used by later jobs. However, by Observation 4.1 in [7], which applies in our case as well, we may always modify any preemptive algorithm so that it never introduces idle times, for non-preemptive variants the same is true for trivial reasons.

(1) For the preemptive variant, we give matching lower and upper bounds on the competitive ratio for any fixed number \( m \) of machines. These bounds increase with \( m \) and tend to \( (1 + \sqrt{3})/2 \approx 1.36603 \), as the number of machines goes to infinity. Our results are also the best possible for randomized algorithms. Our upper bound does not really require that the job sizes are decreasing; even the knowledge of the size of the largest job would be sufficient.

(2) For the non-preemptive deterministic variant, we give a lower bound of \( 7/6 \) for \( m = 2 \) and a lower bound of \( (1 + \sqrt{37})/6 \) for \( m = 3 \). Graham [11] has shown that for decreasing job sizes, the greedy algorithm List is \((4/3 - 1/(3m))\)-competitive. Hence, for \( m = 2 \) machines we have matching bounds. For \( m \geq 3 \), the problem remains open. Quite surprisingly, the non-preemptive variant allows better competitive ratios than the preemptive variant does. In fact, even List achieves a better competitive ratio (than possible with preemption) for every \( m \geq 2 \) as well as in the limit. In the light of the corresponding results for classical online scheduling (see below) this was certainly unexpected and to our knowledge, it is also the first occurrence of this phenomenon.

(3) Finally, for the non-preemptive randomized variant, for \( m = 2 \) machines, we present a randomized \( 8/7 \)-competitive algorithm and prove that it is optimal. Our algorithm is barely random [15], i.e., a distribution over a fixed number of deterministic algorithms. Each job is scheduled in \( O(1) \) time, and the algorithm uses \( O(1) \) space. For classical online scheduling, on the other hand, the known optimal algorithm uses an unbounded number of random choices [3]. In fact, it also uses \( \Omega(i) \) time to schedule the \( i \)th job and \( \Omega(n) \) space overall. For \( m \geq 3 \), the problem remains open.

Related work. Other results on semi-online problems are obtained by Azar and Regev [2] who consider semi-online scheduling where the optimum makespan is known in advance, and by Kellener et al. [13] who consider several semi-online versions of the partition problem, which corresponds to scheduling on 2 machines. Liu et al. [14] investigate the problem where a schedule must be created with knowledge only of the number of jobs and the ordering of their processing times.

In the standard online problem (where the job sizes are not necessarily decreasing) all the three main variants, preemptive, non-preemptive deterministic, and non-preemptive randomized, have been investigated in the literature. The preemptive variant was solved completely by Chen et al. [7] who designed an online algorithm for preemptive scheduling on \( m \) machines with competitive ratio

\[
\beta(m) = \frac{m^m}{m^m - (m - 1)^m} \rightarrow \frac{e}{e - 1} \approx 1.58198. \tag{1}
\]

They also gave a matching lower bound construction which even holds for randomized online algorithms. It is a general phenomenon that for preemptive online scheduling problems randomization does not help, and as we noted above, the same is true also for the semi-online variant studied in this paper.

For the non-preemptive deterministic variant, Graham [10] defined a simple greedy algorithm List which places each job on the machine which is currently least
loaded. He showed that List is \((2 - 1/m)\)-competitive on \(m\) machines; this analysis is tight. Since Graham’s seminal work, many researchers [3–6,8,9,12,16–18] have investigated this problem. The best possible competitive ratio for a large number of machines is now known to lie in the interval \([1.852, 1.923]\); cf. [1].

Only little is known about non-preemptive randomized online scheduling. For \(m = 2\) machines, Bartal et al. [3] give a \(4/3\)-competitive randomized online algorithm. Chen et al. [5] and Sgall [19] independently show an online lower bound of \(\beta(m)\) for any \(m \geq 1\). Note that this bound exactly matches the bound of Bartal et al. for \(m = 2\). Seiden [16,17] presents randomized online algorithms which outperform their deterministic counterparts for \(m = 3, \ldots, 7\).

2. Preemptive scheduling

Throughout this section, we will assume, without loss of generality, that the first (and hence largest) job has size 1. First, to develop the intuition behind the optimal algorithm, we describe its action on the sequence of jobs of size 1; this is also the sequence that will be used in the lower bound argument.

Suppose we want to achieve competitive ratio \(c \geq 1\). We schedule the first \(m\) jobs during the time slots before time \(c\); we do that by filling the machines one by one (since \(c \geq 1\), each job is split onto at most two machines and the two time slots do not overlap). Let \(k = \lceil m/c \rceil\). For \(i \leq k\), the \((m + i)\)th job is scheduled so that \(c/m\) of its load is scheduled after time \(c\) on the first \(i\) machines, each receiving load \(c/m\), and the rest of the job is scheduled before time \(c\). The \(i\) largest loads of machines will be \(c(1 + i/m), c(1 + (i - 1)/m), \ldots, c(1 + 1/m)\), which guarantees that in the next round, the time slots do not overlap. After the \((m + k)\)th job, the new jobs are scheduled only during the slots after \(c\) in a similar manner, on at most \(k + 1\) machines. We need to choose the right value of \(k\) and the corresponding value of \(c\) so that this scheme works, i.e., \(c\) is large enough so that the load we need to schedule before time \(c\) is at most \(cm\).

For \(0 \leq k \leq m\), we define values
\[
\gamma_{m,k} = \frac{2m^2 + 2mk}{2m^2 + k^2 + k},
\]
\[
\gamma_m = \max_{k = 0, \ldots, m} \gamma_{m,k}.
\]

For any \(m \geq 2\), let \(k(m)\) be the largest integer satisfying
\[
\gamma_m = \gamma_{m,k(m)}.
\]

**Lemma 2.1.** For every \(m \geq 2\) the following conditions hold:

(i) \(1 \leq \gamma_m \leq (1 + 1/\sqrt{3})/2 \approx 1.36603\). Moreover, as \(m \to \infty\), \(\gamma_m \to (1 + 1/\sqrt{3})/2\).

(ii) \(1 \leq k(m) = [m/\gamma_m] < m\).

(iii) \(\gamma_m < \gamma_{m+1}\).

Proof. (i) Obviously \(1 \leq \gamma_{m,k} \leq (2+x)/(2+x^2)\), where \(x = k/m\). This rational function is maximized for \(x = \sqrt{3} - 1\) and has maximal value \((1 + \sqrt{3})/2\). Moreover, for \(m\) large and \(k = [m(\sqrt{3} - 1)]\), the value of \(\gamma_{m,k}\) converges to \((1 + 1/\sqrt{3})/2\).

(ii) If \(k + 1 \leq m/\gamma_m\) then
\[
\gamma_{m,k+1} = \frac{(2m^2 + 2mk) + 2m}{(2m^2 + k^2 + k)} + 2(k + 1) \geq \min \left\{ \frac{2m^2 + 2mk}{2m^2 + k^2 + k} + 1 \right\} = \gamma_{m,k},
\]
and thus \(k \neq k(m)\). Similarly, we can verify that, if \(k > m/\gamma_m\), then \(\gamma_{m,k-1} > \gamma_{m,k}\) and \(k \neq k(m)\). This implies that \(k(m) = [m/\gamma_m]\). We have \(\gamma_{m,0} = 1 \leq \gamma_{m,1}\) and thus \(k(m) \geq 1\). For \(m \geq 2\) we have \(\gamma_{m,1} > 1\), thus \(k(m) = [m/\gamma_m] < [m/\gamma_{m,0}] < m\).

(iii) Let \(k = k(m)\), and hence \(\gamma_m = \gamma_{m,k}\). We distinguish two cases.

First, assume \(k \geq (\sqrt{3} - 1)m\). In this case we prove that \(\gamma_{m,k} < \gamma_{m+1,k}\), which implies \(\gamma_m < \gamma_{m+1}\). Plugging in the definitions of \(\gamma_{m,k}\) and \(\gamma_{m+1,k}\) and simplifying shows that the \(\gamma_{m,k} < \gamma_{m+1,k}\) is equivalent to \(2m^2 - 2mk < (k + 1)^2\). However, the assumption implies that \(2m^2 - 2mk \leq k^2 < (k + 1)^2\), and we are done.

Second, assume \(k < (\sqrt{3} - 1)m\). We prove that \(\gamma_{m,k} < \gamma_{m+1,k+1}\), which also implies \(\gamma_m < \gamma_{m+1}\). Plugging in the definitions of \(\gamma_{m,k}\) and \(\gamma_{m+1,k+1}\) and simplifying shows that \(\gamma_{m,k} < \gamma_{m+1,k+1}\) is equivalent to
\[(k - m)(k^2 + 2km - 2m^2) + k(3k - m + 2) > 0\].

Our assumption implies that the first term is positive. From (ii) and (i) we obtain that \(k = [m/\gamma_m] \geq [m/2] \geq m/2 - 1\). This implies that the second term is non-negative, and the proof is complete. □
Theorem 2.2. For every \( m \geq 2 \), there exists a \( \gamma_m \) competitive deterministic algorithm for semi-online preemptive scheduling on \( m \) machines with decreasing job sizes.

Proof. Let \( c = \gamma_m \), and let \( k = k(m) \) be the largest integer for which \( \gamma_m = \gamma_{m,k} \). Consider the following algorithm: Suppose that at the current moment, jobs of total time \( T \) are already scheduled and that a job of size \( t \leq 1 \) arrives. We schedule it without introducing any idle time between consecutive job pieces. The assignment to the machines is described below in several cases; we prove later that the assigned time slots never overlap:

1. If \( T + t \leq m \), we schedule the job so that it is scheduled before time \( c \), on the first machine available. More precisely, if the first machine with the current load less than \( c \) has load at most \( c - t \), we schedule the whole job on it. If not, we schedule it on a portion of the job, so that the load becomes \( c \), and the rest on the next machine.

2. If \( T \geq m + i - 1 \) and \( T + t \leq m + i \) for some integer \( i, 1 \leq i \leq k \), schedule on each of the first \( i \) machines \( tc/m \) of the job, and schedule the remainder of the job before time \( c \) on the first available machine, as in Step 1.

3. If \( T \leq m + i \) and \( T + t > m + i \) for some integer \( i, 0 \leq i \leq k \), schedule on each of the first \( i \) machines \( tc/m \) of the job, on machine \( i + 1 \), \((T + t - m - i)/c \) of the job (or the whole remainder, if it is smaller — this can happen only if \( i = k \)), and schedule the remainder of the job before time \( c \) on the first available machine, as in Step 1.

4. If \( T \geq m + k \), schedule on each of the first \( k \) machines \( tc/m \) of the job, and the remainder on the machine \( k + 1 \).

At the moment where the algorithm has assigned a total load of \( m + i \), \( i = 0, \ldots, k \) (this moment may come in the middle of some jobs), the machines 1, 2, \ldots, \( i + 1 \) have loads exactly \( c(1 + i/m), c(1 + (i - 1)/m), \ldots, c \), respectively. This follows from adding the loads of all contributions. The only non-trivial part is to verify that the load of machine \( i + 1 \) is (not less than) \( c \). For \( i < k \), this is true already after total size \( m \) (from the fact that \( k = \lceil m/c \rceil \)). For \( i = k \), it follows from the calculation in the next paragraph.

Before time \( m + k \), the total size of portions of jobs scheduled before time \( c \) is

\[
m + k - \frac{k(k + 1)c}{2m} = cm
\]

using the definition of \( k \) and \( c = \gamma_k \). This implies the claim in the previous paragraph. Also, this implies that it is actually possible to fit all these portions before time \( c \). They are scheduled so that no two parts of the same job run at the same time on different machines, since \( c \geq 1 \).

After total load \( m \), the portion of a job scheduled on a single machine after time \( c \) is at most \( c/m \). Since the loads of any two of these machines differ by at least \( c/m \) at all times, these pieces do not overlap. For a job scheduled at a total load \( T < m + i \) but finished after \( m + i \), it is also necessary to argue that its portions assigned to machines \( i \) and \( i + 1 \) are not in parallel: machine \( i \) starts at load \( c(1 + (T + 1 - m - i)/m) \), while machine \( i + 1 \) starts at \( c \) and finishes at \( c(1 + (T + t - m - i)/m) \), which is smaller since \( t \). The last fact we need to verify is that the remainders of jobs scheduled on machine \( k + 1 \) in Step 4 are not scheduled in parallel with other portions of the same job. The remainder of a job of size \( t \) is at most \( t - ktc/m < tc/m \), because \( k = \lfloor m/c \rfloor \). Thus the loads of machines \( k \) and \( k + 1 \) differ by at least \( c/m \) after time \( m + k \).

The competitive ratio of the algorithm is implied by the above claims about the loads of the algorithm. \( \square \)

Theorem 2.3. No (deterministic or randomized) algorithm for semi-online preemptive scheduling on \( m \geq 2 \) machines with decreasing job sizes can have a competitive ratio less than \( \gamma_m \).

Proof. Fix \( k \leq m \) and a randomized algorithm. Suppose that the algorithm is \( c \)-competitive. We will prove that on a sequence of \( m + k \) unit jobs the competitive ratio is at least \( \gamma_{m,k} \). This implies \( c \geq \gamma_m \).

Let the randomized algorithm run on a sequence of \( m + k \) unit jobs. Let \( X_i \) be the last time when at least \( i \) jobs are running (after scheduling all \( m + k \) jobs); note that \( X_i \) is a random variable, since the algorithm is randomized. We claim that for \( 0 \leq i \leq k \),

\[
E[X_{k+1-i}] \leq \frac{c(m+i)}{m}.
\]

As long as \( k + 1 - i \) jobs are running, at least one of the first \( m + i \) jobs is running. Therefore, the average
makespan of the algorithm on the first $m + i$ jobs is at least $E[X_{i+1}]$. Since the optimal schedule for these $m + i$ jobs has makespan $(m + i)/m$, inequality (2) follows from the $c$-competitiveness of the algorithm. The total size of jobs scheduled by the algorithm is, for every random choice, at most

$$\sum_{i=1}^{m} X_i \leq \sum_{i=1}^{k} X_i + (m - k)X_{k+1}.$$ 

This size has to be at least $m + k$, and by using (2) we obtain

$$m + k \leq \sum_{i=1}^{k} X_i + (m - k)X_{k+1} \leq c \left( m + \frac{k(k + 1)}{2m} \right).$$

This is equivalent to $c \geq \gamma_{m,k}$. The lower bound follows. 

Corollary 2.4. There exists a deterministic algorithm for semi-online preemptive scheduling with decreasing job sizes whose competitive ratio is $(1 + \sqrt{3})/2 \approx 1.36603$. No randomized algorithm can achieve a better competitive ratio.

3. Non-preemptive deterministic scheduling

Theorem 3.1. Any deterministic algorithm for semi-online non-preemptive scheduling with decreasing job sizes has competitive ratio at least $7/6$ for $m = 2$, and competitive ratio at least $(1 + \sqrt{37})/6$ for $m = 3$. Moreover, for $m = 2$ there exists a matching $7/6$-competitive deterministic algorithm (due to [11]).

Proof. The case $m = 2$ follows as in [11]: Consider an instance with 2 jobs of size 3 followed by 3 jobs of size 2. If a deterministic algorithm puts the 2 jobs of size 3 on the same machine, no other jobs may arrive; hence, its competitive ratio is at least 2. If it puts the 2 jobs of size 3 on different machines, it is $7/6$-competitive on Graham’s sequence. Graham [11] shows that the competitive ratio of List is $4/3 - 1/(3m)$, this gives a tight upper bound of $7/6$ for $m = 2$.

Next, consider the case $m = 3$. We show a lower bound of $c = \frac{1 + \sqrt{37}}{6} \approx 1.18046$.

Let $x = (7 - 3c)/6 \approx 0.57643$. Consider the following job list: $x, x, 1 - x, 1 - x, 1/3, 1/3$ and $1/3$. The first three jobs must go on three pairwise distinct machines (otherwise the competitive ratio is at least $1/x = 6/(7 - 3c) > 1.73479 > c$). Consider the fourth job. If it is placed on the same machine as one of the first two jobs, then the algorithm’s makespan is 1, whereas the optimal offline makespan is $2(1 - x)$; note that $1/(2 - 2x) = 3/(3c - 1) = c$. If the fourth job is placed on the same machine as the third job, then the algorithm’s final makespan is at least $7/3 - 2x = c$. The optimal makespan for the entire list is 1. Summarizing, any online algorithm has competitive ratio at least $c$. 

4. Non-preemptive randomized scheduling

In this section, we discuss randomized semi-online non-preemptive scheduling with decreasing job sizes on $m = 2$ machines. We give matching lower and upper bounds of $8/7$ for this problem.

Theorem 4.1. No randomized algorithm for semi-online non-preemptive scheduling with decreasing job sizes on $m = 2$ machines can have a competitive ratio better than $8/7$.

Proof. We consider a problem instance with 2 jobs of size 3 followed by 3 jobs of size 2, as in [11]. If the algorithm places the first two jobs on the same machine with probability more than $1/7$, then the competitive ratio is at least $((1/7) \times 6 + (6/7) \times 3)/3 = 8/7$ and we are done. If not, the adversary gives the algorithm the remaining three jobs. The optimal solution places the 2 size 3 jobs together and the 3 size 2 jobs together, yielding a makespan of 6. The online algorithm can do this with probability at most $1/7$. If the online algorithm has placed the 2 size 3 jobs on separate machines, the best makespan it can achieve is 7. Therefore, the competitive ratio is at least $((1/7) \times 6 + (6/7) \times 7)/6 = 8/7$. 

Theorem 4.2. There exists a randomized $8/7$-competitive algorithm for semi-online non-preemptive scheduling with decreasing job sizes on $m = 2$ machines.

Proof. We call our machines $A$ and $B$. The algorithm places the first two jobs on machine $A$ with probability
1/7. With probability 6/7, it places the first on A and the second on B. Thereafter it schedules each job on
the machine with least load, i.e., it behaves like List [11].

Our randomized algorithm is a distribution over two
deterministic algorithms. We analyze it by considering it
to be a single deterministic algorithm which creates
two schedules. We show that the weighted average of
the makespans of the two schedules is at most 8/7
times the makespan of the optimal offline schedule. In
schedule π, the first two jobs are assigned to machine
A. In schedule σ, they are assigned to machines A and
B, respectively. The weights for the schedules are 1/7
and 6/7, respectively.

For the purposes of analysis we assume that there
are n ≥ 6 jobs. This assumption is valid since any
shorter job list may be extended with size 0 jobs. Def-
define \( P_i = \sum_{j=1}^{\infty} p_j \). Let \( a_i^x \) and \( b_i^x \) be the loads on ma-
chines A and B in Schedule \( x \in \{ \pi, \sigma \} \) after the first
\( i \) jobs have been scheduled, respectively. Let \( x_i^x = \max\{ a_i^x, b_i^x \} \). Let \( X_i^x \) be the machine to which job \( i \) is
assigned in Schedule \( x \). Let \( opt \) be the optimal offline
makespan for the first \( i \) jobs. We show that
\[
\frac{1}{7} x_i^\pi + \frac{6}{7} x_i^\sigma - \frac{8}{7} \text{opt}_n
\]
is at most 0.

By definition of the algorithm we have
\[
a_i^\pi = p_1, \quad b_i^\pi = 0, \quad x_1^\pi = a_1^\pi,
\]
\[
a_i^\sigma = p_1 + p_2, \quad b_i^\sigma = 0.
\]
\[
a_i^\sigma = p_1, \quad b_i^\sigma = 0.
\]
For \( 3 < i < 6 \) and \( x = \pi \) and for \( 2 < i < 6 \) and \( x = \sigma \)
we have the equations
\[
b_i^\pi = b_i^{\pi - 1} + p_i, \quad a_i^\sigma = a_i^{\sigma - 1},
\]
\[
a_i^\sigma = a_i^{\sigma - 1} + p_i, \quad b_i^\sigma = b_i^{\sigma - 1},
\]
\[
x_i^{\pi - 1} = \begin{cases} a_i^{\pi - 1} \geq b_i^{\pi - 1} \quad \text{if } X_i^\pi = B, \\ b_i^{\pi - 1} \geq a_i^{\pi - 1} \quad \text{otherwise.} \end{cases}
\]
We assume, without loss of generality, that \( p_1 = 1 \).
We have the constraints
\[
P_n \geq \sum_{j=1}^{6} p_j,
\]
\[
p_1 \geq p_2 \geq p_3 \geq p_4 \geq p_5 \geq p_6,
\]
opt_n \geq p_1, \quad \text{opt}_n \geq \frac{P_n}{2}.
\]
The optimal algorithm schedules at least three of the
first five jobs on the same machine, and thus we have
\[
\text{opt}_n \geq p_3 + p_4 + p_5.
\]
List is optimal for 4 jobs with decreasing processing
times. We therefore have
\[
\text{opt}_n \geq x_i^\sigma.
\]
Consider how the first five jobs are scheduled. In
Schedule \( \pi \), the first 2 jobs go on A, while the next 3
go on B. i.e.
\[
X_1^\pi = A, \quad X_2^\pi = A, \quad X_3^\pi = B, \quad X_4^\pi = B, \quad X_5^\pi = B.
\]
There are 2 possible states which we might be in after
the first 5 jobs. Either \( X_6^\pi = A \) or \( X_6^\pi = B \). In Schedule
\( \sigma \), the first job goes on A, while jobs 2 and 3 go on B,
i.e.
\[
X_1^\sigma = A, \quad X_2^\sigma = B, \quad X_3^\sigma = B.
\]
The values of \( X_4^\sigma, X_5^\sigma \) and \( X_6^\sigma \) might each have been
A or B, leading to a total of 8 possibilities. Since there
are 2 possibilities for Schedule \( \pi \) and 8 for Schedule
\( \sigma \), there are a total of 16. However, note that
\[
X_6^\sigma = A, \quad X_4^\pi = B, \quad X_5^\pi = B, \quad X_6^\pi = B
\]
and
\[
X_6^\pi = A, \quad X_4^\pi = B, \quad X_5^\pi = B, \quad X_6^\pi = A
\]
are impossible: \( X_6^\pi = A \) implies that \( p_3 + p_4 + p_5 \geq p_1 + p_2 \) and therefore \( p_3 + p_4 \geq p_1 \). But in both situations,
Schedule \( \sigma \) implies that \( p_1 \geq p_2 + p_3 + p_4 \geq 2p_1 + p_2 \),
a contradiction. Therefore, there are only 14 possible
ways in which the first 5 jobs are scheduled.

Now consider the situation in Schedule \( \pi \) for the
scheduling of jobs 6, . . . , n. Define
\[
Q = P_n - P_6, \quad y_5^\pi = P_5 - x_5^\pi = \min\{ a_5^\pi, b_5^\pi \}
\]
for \( x \in \{ \pi, \sigma \} \). There are three possibilities in each
schedule:

1. \( y_5^\pi + p_6 + Q \leq x_5^\pi \), in which case we have \( x_n^\pi = x_5^\pi \).
2. \( y_5^\pi + p_6 + Q > x_5^\pi \) and \( y_5^\pi + Q \leq x_5^\pi \), in which case we have \( x_n^\pi \leq (P_n - p_6)/2 + p_6 \).
3. \( y_5^\pi + Q > x_5^\pi \), in which case we have \( x_n^\pi \leq (P_n - p_6)/2 + p_6 \).
So there are \(3 \times 3 = 9\) ways in which jobs 6, \ldots, \(n\) might be scheduled in \(\pi\) and \(\sigma\). Using the above breakdown, there are \(14 \times 9 = 126\) possible sub-cases. Each of these sub-cases can be described by a linear program with objective function (4), constraints (5)–(13) and variables \(p_1, \ldots, p_6, P, Q, a_1^x, \ldots, a_6^x, b_1^x, \ldots, b_6^x, x_1^x, \ldots, x_5^x, y_5^x, x_n^x, \text{opt}_x\) for \(x \in \{\pi, \sigma\}\). All of these 126 linear programs have objective value at most 0, as we have verified by using a computer. The Mathematica program we used is available on the World Wide Web at: [http://www.csc.lsu.edu/~seiden/ordered.m](http://www.csc.lsu.edu/~seiden/ordered.m)

We conclude that the algorithm indeed is \(8/7\)-competitive. \(\square\)

References