Semi On-Line Scheduling on Two Uniform Processors

Enrico Angelelli, Maria Grazia Speranza
Università di Brescia, C.da S.Chiara 50 - Brescia, Italy, {angele, speranza}@eco.unibs.it

Zsolt Tuza
Computer and Automation Institute, Hungar. Acad. Sci., Budapest, Hungary, tuza@sztaki.hu
Department of Computer Science, University of Pannonia, Veszprém, Hungary

In this paper we consider the problem of semi on-line scheduling on two uniform processors, in the case where the total sum of the tasks is known in advance. Tasks arrive one at a time and have to be assigned to one of the two processors before the next one arrives. The assignment cannot be changed later. The objective is the minimization of the makespan. We derive lower bounds and algorithms, depending on the value of the speed \( s \) of the fast processor (the speed of the slow processor is normalized to 1). The algorithms presented for \( s \geq \sqrt{3} \) are optimal, as well as for \( s = 1 \) and for \( \frac{1+\sqrt{17}}{4} \leq s \leq \frac{1+\sqrt{3}}{2} \).

Keywords: Multi-processor Scheduling, Real-Time Scheduling, Theoretical Scheduling.

1 Introduction

The typical framework of on-line optimization problems requires to take decisions as data becomes available over time, while no information is given about the future. When all information is available at one time before optimization, the problem is called off-line. In on-line scheduling, tasks arrive one by one and no information is available about the number or the size of tasks that will arrive in the future. As soon as a task arrives it has to be assigned to a processor and the decision cannot be changed later. This framework captures an important feature of real problems, the absence of information. Nevertheless, even in real applications some information may be available to the decision makers. In this case the problem is called semi on-line. Though optimization algorithms for semi on-line problems cannot be as effective as those designed for off-line frameworks, even a little additional information may allow a dramatic improvement of the performance with respect to the pure on-line setting. Thus, it becomes of relevant interest to study the impact of information on the performance of algorithms for a given problem.

1.1 Problem definition and notation

In this paper we consider the semi on-line scheduling problem on two uniform processors in the case where the sum of the tasks is known in advance.

A set of ordered tasks \( \{p_1, p_2, \ldots, p_n\} \) such that \( \sum_{j=1}^{n} p_j = 1 + s \) arrive one at a time and have to be immediately assigned to one of two processors. The speed of the slower processor is normalized to 1, and the speed of the faster one is denoted by \( s \geq 1 \); as a particular case, for \( s = 1 \) we have two identical processors. We call the processors \( P_1 \) and \( P_2 \), referring in the subscript to their speed. The processing time of task \( p_j \) on processor \( P_h \) is \( p_j/h \).

Given a problem instance, we indicate by \( L_1 \) and \( L_s \) the loads of the processors just before the assignment of task \( p_i \) and by \( L_1 \) and \( L_s \) the final loads of the two processors. The objective of the problem is to minimize the makespan \( \max(L_1, L_s/s) \). Moreover, we use \( H \) to indicate both an
algorithm and the makespan generated by the same algorithm on the given instance. The optimum
off-line makespan is denoted by $Z^*$.

The performance of an on-line algorithm is measured by the competitive ratio. An on-line
algorithm $H$ for a minimization problem is said to be $r$-competitive if for any instance, given the
off-line optimum $Z^*$, the inequality $H \leq r \cdot Z^*$ holds. The competitive ratio $R_H$ is defined as
$\inf \{ r \mid H \text{ is } r\text{-competitive} \}$. An algorithm $H$ is said to be optimal if no other algorithm has a
better competitive ratio (see Sleator and Tarjan [12]).

1.2 Literature

The (semi) on-line scheduling problem where a sequence $I$ of tasks has to be assigned to a set of $m$
processors so that the maximum completion time (makespan) is minimized has attracted the interest
of many researchers. There are several variants for this problem. For example, the processors may
be identical or uniform, while the scheduling may be preemptive or non-preemptive. There is also a
number of different algorithmic approaches, deterministic and randomized algorithms, algorithms
designed for a fixed number of processors and algorithms for any number of processors. We refer

For the pure on-line problem, the well-known List algorithm introduced by Graham [8] has
been proved to be the best possible for the case of identical processors. The List algorithm assigns
the incoming task to the least loaded processor; and its natural extension to uniform processors
assigns the task to the processor where it can be completed first. Cho and Shani [4] proved that
List is $1+\sqrt{5}/2$-competitive for all $s$ and the bound is tight when $s = 1+\sqrt{5}/2$. Epstein et al. [5]
extended this result to show that List is $\min\{2s+1, 1+1/s\}$-competitive and optimal for all speeds $s$.
Epstein et al. [5] provided also randomized algorithms with better performance than List in
the case that no preemption is allowed. In the same paper, for the problem with preemption, an
optimal $1+s/(s^2+s+1)$-competitive algorithm for all $s \geq 1$ was proposed that cannot be beaten
by any randomized algorithm. This latter result was obtained independently by Wen and Du [13].
Several semi on-line scheduling problems have been studied for the case of two processors, both
identical and uniform. For identical processors, the case of given sum of the tasks and a number of
variants were studied in [10], [9], [1], [2] and [3]. For the case of two uniform processors, Epstein and
Favrholdt proposed optimal semi on-line algorithms on two uniform processors when tasks arrive
with non-increasing size, in both the non-preemptive case [6] and the preemptive case [7].

1.3 Our results

In this paper we provide both lower and upper bounds for the problem. All possible values of speed
$s$ for the fast processor are taken into account. For some ranges of $s$ the proposed algorithms are
proved to be optimal. Namely, the first algorithm is optimal with competitive ratio $4/3$ for the
special case $s = 1$ (identical processors), the second algorithm is optimal with competitive ratio $s$
in the range $1+\sqrt{7}/4 \leq s \leq 1+\sqrt{3}/2$ and, finally, the third algorithm is optimal for all $s \geq \sqrt{3}$ with
competitive ratio $1+1/s+1$. Moreover, we show that, despite the fact that in many semi on-line
problems the List algorithm implicitly exploits the available information and performs better than
in the pure on-line setting (see [6] and [9] for examples), in this case there is a need for specialized
algorithms. Indeed, when the total size of the tasks is fixed, the List algorithm does not improve
the bound $\min\{2s+1, 1+1/s\}$ showed in [5]. Furthermore, we note that, for $s \geq 1+\sqrt{5}/2$, the worst-case
ratio for List is the same as the one obtained by the trivial algorithm that blindly assigns all tasks
to the fast processor.
Figure 1: Bounds for the semi on-line problem.

In Section 2 we prove lower bounds for the problem and in Section 3 we propose three algorithms and prove their competitive ratio.

In Figure 1 we show the structure of the largest lower bounds and the best upper bounds we obtained depending on the value of the speed $s$. In Figure 2 we show the comparisons of the best upper bound for the semi on-line problem versus the best lower bound for the pure on-line problem. This proves the effectiveness of the information made available.

2 Lower bounds

In this section we present a number of lower bounds as functions of the speed $s$. The lower bounds are presented according to the range of speed $s$ for which they represent the best lower bound. Though, in each theorem statement we claim the full range of speed for which the lower bound holds.

In order to derive lower bounds on the algorithms for the problem, we define a number of parametric instances which we describe through the size of the tasks listed in parentheses in the order of arrival:

- $A = \{\delta, \delta, s - 2\delta, 1\}$
- $B = \{\delta, \delta, s - \delta, 1 - \delta\}$
- $C = \{\delta, \delta, s, 1 - 2\delta\}$
- $D = \{\delta, s - \delta, 1\}$

The instances depend on the parameter $\delta$ such that $\delta \leq \frac{1}{2}$. The off-line optimum is equal to 1 for all the instances above.

In order to simplify the derivation of the lower bounds, it is convenient to consider the following classes of algorithms:

$H_{10} \equiv \{\text{algorithms assigning task } p_1 = \delta \text{ to } P_1\}$
Figure 2: Semi on-line vs pure on-line problem.

\[ H_{20} \equiv \{ \text{algorithms assigning tasks } p_1, p_2 = \delta \text{ to } P_1 \} \]
\[ H_{02} \equiv \{ \text{algorithms assigning tasks } p_1, p_2 = \delta \text{ to } P_2 \} \]
\[ H_{11} \equiv \{ \text{algorithms assigning tasks } p_1, p_2 = \delta \text{ to } P_1 \text{ and } P_2 \text{, in any order} \} \]

The following relations between the set \( H \) of all algorithms and the above classes of algorithms will be used:

\[ H = H_{10} \cup H_{11} \cup H_{02} \]
\[ H = H_{11} \cup H_{20} \cup H_{02}. \]

With a little abuse of notation we indicate with \( H_{10}(D) \) any general lower bound that is valid for the makespan obtained by all algorithms in the set \( H_{10} \) on instance \( D \). We use a similar notation for all the other classes of algorithms and instances. We will use the following bounds:

\[ H_{10}(D) \geq \min(1 + s, \frac{1 + s - \delta}{s}, s, 1 + \delta) \]
\[ H_{11}(A) \geq \min(1 + s - \delta, \frac{1 + s - \delta}{s}, \max(\frac{s + 1}{s}, s - \delta), 1 + \delta) \]
\[ H_{11}(C) \geq \min(1 + s - \delta, \frac{1 + s - \delta}{s}, \delta + s, \frac{s + \delta}{s}) \]
\[ H_{20}(B) \geq \min(1 + s, \frac{1 + s - 2\delta}{s}, s + \delta, 1 + \delta) \]
\[ H_{02}(B) \geq \min(1 + s - 2\delta, \frac{1 + s}{s}, \max(s - \delta, \frac{s + \delta}{s}), s + \delta) \]
\[ H_{02}(C) \geq \min(1 + s - 2\delta, \frac{1 + s}{s}, s, \frac{s + 2\delta}{s}). \]

We are now ready to derive the lower bounds on any algorithm for the solution of the semi on-line problem on two processors.

2.1 The range \( 1 \leq s < \frac{1 + \sqrt{37}}{6} \approx 1.1805 \)

**Theorem 1** No algorithm can guarantee a competitive ratio better than \( \frac{s + 3}{3s} \) for \( s \in [1, \frac{3}{2}] \).

**Proof.** Let us fix \( \delta = \frac{s}{3} \) and view the set of algorithms \( H \) as \( H = H_{11} \cup H_{20} \cup H_{02} \). The lower bound is implied by the fact that \( H_{11}(C), H_{20}(B) \) and \( H_{02}(B) \) are bounded from below by \( \frac{s + 3}{3s} \). \( \square \)
2.2 The range \( \frac{1+\sqrt{37}}{6} \leq s < \frac{1+\sqrt{3}}{2} \)

**Theorem 2** No algorithm can guarantee a competitive ratio better than \( s \) for \( s \in [1, \frac{1+\sqrt{3}}{2}] \).

**Proof.** Let us fix \( \delta = (s^2 - s) \leq \frac{1}{2} \), for \( s \in [1, \frac{1+\sqrt{3}}{2}] \), and view the set of algorithms \( H \) as \( H = H_{10} \cup H_{11} \cup H_{02} \). The lower bound is implied by the fact that \( H_{10}(D), H_{11}(C) \) and \( H_{02}(C) \) are bounded from below by \( s \).

2.3 The range \( \frac{1+\sqrt{3}}{2} \leq s < \frac{1+\sqrt{5}}{2} \)

**Theorem 3** No algorithm can guarantee a competitive ratio better than \( 1 + \frac{1}{2s} = \frac{2s+1}{2s} \) for \( s \in [\frac{1+\sqrt{3}}{2}, 2] \).

**Proof.** Let us fix \( \delta = \frac{1}{2} \), for \( s \in [\frac{1+\sqrt{3}}{2}, 2] \), and view the set of algorithms \( H \) as \( H = H_{10} \cup H_{11} \cup H_{02} \). The lower bound is implied by the fact that \( H_{10}(D), H_{11}(C) \) and \( H_{02}(C) \) are bounded from below by \( \frac{2s+1}{2s} \).

2.4 The range \( \frac{1+\sqrt{5}}{2} \leq s < \sqrt{3} \)

**Theorem 4** No algorithm can guarantee a competitive ratio better than \( \frac{s+1}{2} \) for \( s \in [1, \sqrt{3}] \).

**Proof.** Let us fix \( \delta = \frac{s-1}{s} \), for \( s \in [1, \sqrt{3}] \), and view the set of algorithms \( H \) as \( H = H_{10} \cup H_{11} \cup H_{02} \). The lower bound is implied by the fact that \( H_{10}(D), H_{11}(A) \) and \( H_{02}(C) \) are bounded from below by \( \frac{s+1}{2} \).

2.5 The range \( \sqrt{3} \leq s \)

**Theorem 5** No algorithm can guarantee a competitive ratio better than \( 1 + \frac{1}{s+1} = \frac{s+2}{s+1} \) for \( s \geq \sqrt{3} \).

**Proof.** The proof is straightforward from the following two Lemmas.

**Lemma 6** No algorithm can guarantee a competitive ratio better than \( \frac{s+2}{s+1} \) for \( s \in [\sqrt{3}, 2] \).

**Proof.** Let us fix \( \delta = \frac{1}{s+1} \), for \( s \in [\sqrt{3}, 2] \), and view the set of algorithms \( H \) as \( H = H_{10} \cup H_{11} \cup H_{02} \). The lower bound is implied by the fact that \( H_{10}(D), H_{11}(A) \) and \( H_{02}(C) \) are bounded from below by \( \frac{s+2}{s+1} \).

**Lemma 7** No algorithm can guarantee a competitive ratio better than \( \frac{s+2}{s+1} \) for \( s \geq 2 \).

**Proof.** Let us fix \( \delta = \frac{1}{s+1} \) and consider the following instance, where the incoming task is generated according to the situation of the loads of the two processors:

a) if \( L_1^i < \delta \) and \( L_8^i < 1 - \delta \), then \( p_i = \min\{\delta - L_1^i, (1 - \delta) - L_8^i\} \);

b) if \( L_1^i = \delta \), then \( p_i = 1 \), \( p_{i+1} = s - (\delta + L_1^i) \) and \( p_{i+1} \) is the last task;

c) if \( L_1^i = 1 - \delta \), then \( p_i = s \), \( p_{i+1} = \delta - L_1^i \) and \( p_{i+1} \) is the last task.

According to rule a), both inequalities \( L_1^i + p_i \leq \delta \) and \( L_8^i + p_i \leq 1 - \delta \) hold, and after the assignment at least one of the two inequalities is strict and the sum of the loads of the two processors is still less than 1.
If the instance ends in case b), then $p_{i+1} > 1$ because $s \geq 2$ and the off-line optimum is equal to 1. The algorithm cannot obtain a value better than

$$\min \{1 + \delta, \frac{L_s^i + 1 + s - (\delta + L_s^i)}{s}\} = \min \{1 + \delta, \frac{1 + s - \delta}{s}\} = \frac{s + 2}{s + 1}.$$  

If the instance terminates in case c), then the off-line optimum is equal to $\frac{s}{s} = 1$ while the algorithm cannot obtain a value better than

$$\min \{s + L_1^i, \frac{1 + s - \delta}{s}\} \geq \min \{s, \frac{1 + s - \delta}{s}\} = \min \{s, \frac{s + 2}{s + 1}\}.$$  

In conclusion, the algorithm cannot obtain a ratio better than $\frac{s^2 + 2}{s + 1}$ when $s \geq 2$. □

3 Algorithms

In this section we first show that the List algorithm does not improve its performance when the total sum of the tasks is fixed with respect to the pure on-line problem, and then we present three different algorithms that in some intervals of the values of the speed $s$ are optimal. We classify the algorithms according to the values of speed $s$ for which they represent the best lower bound. Though, in each theorem statement, we claim the full range of speed for which the competitive ratio holds.

**Proposition 8** When the sum of the tasks is fixed, the List algorithm has competitive ratio $\min[1 + \frac{s}{s+1}, 1 + \frac{1}{s}]$.

**Proof.** It is already known that $\min[1 + \frac{s}{s+1}, 1 + \frac{1}{s}]$ is an upper bound for the performance of List algorithm in pure on-line setting. Thus, the bound holds for all particular cases like the fixed sum of the tasks. What we need to show is that List cannot guarantee a better performance. It is enough to consider the two instances:

- $I = \{s^2, (1 - s^2), s\}$ when $s < \frac{1 + \sqrt{5}}{2}$ and
- $I = \{1, s\}$ when $s \geq \frac{1 + \sqrt{5}}{2}$.

3.1 The range $1 \leq s < (1 + \sqrt{17})/4$

**Algorithm $H'$:**

If $p_i + L_s^i \leq s(1 + \frac{1}{2s + 1})$ then assign $p_i$ to $P_s$ else assign $p_i$ to $P_1$

**Theorem 9** Algorithm $H'$ is $(1 + \frac{1}{2s + 1}) = \frac{2 + 2s}{2s + 1}$ -competitive for all $s$ in the interval $1 \leq s \leq \frac{1 + \sqrt{17}}{4} \approx 1.2808$.

**Proof.** Let us fix the constant $c = \frac{1}{2s + 1}$. We first observe that $s \in [1, \frac{1 + \sqrt{17}}{4}]$ implies

$$1 + c = \frac{2 + 2s}{2s + 1} \geq s. \tag{1}$$

Let us denote with $p_k$ the first task such that $p_k + L_s^k > s(1 + c)$.  

MISTA 2007
If \( L_k^s \geq s - c \), then the total size of \( p_k \) and all the successive tasks is less than \((1+s)-(s-c)=1+c\). Thus, task \( p_k \) and all the others can be assigned to \( P_1 \) without violating the bound \( 1+c \).

If \( L_k^s < s - c \), then the incoming task \( p_k \) is necessarily greater than \( s(1+c)-(s-c)=c(s+1) \). Note that all the other tasks will be assigned to \( P_s \). In fact, from \( c = \frac{1}{2s+1} \) the inequality \((1+s)-c(s+1) \leq s(c+1) \) follows.

If \( p_k \leq 1+c \), then the bound \( H \leq (1+c) \) is guaranteed.

If \( p_k > 1+c \), then from inequality (1) it follows that the off-line optimum is at least \( p_k/s \), and so the ratio \( H/Z^\star \) is not larger than \( \frac{p_k}{p_k/s} = s \leq 1+c \). \( \square \)

3.2 The range \( (1+\sqrt{17})/4 \leq s < \sqrt{2} \)

Algorithm \( H'' \):

\[
\text{If } p_i + L_i^1 \leq s^2 \text{ then assign } p_i \text{ to } P_s \text{ else assign } p_i \text{ to } P_1
\]

**Theorem 10** Algorithm \( H'' \) is \( s \)-competitive for all \( s \geq \frac{1+\sqrt{17}}{4} \).

**Proof.** Let us consider the first task \( p_k \) such that \( p_k + L_k^s > s^2 \).

If \( L_k^s > 1 \), then the total size of \( p_k \) and all the successive tasks sum up to less than \( s \). This means that no task can force the algorithm to load \( P_1 \) more than \( s \) and \( P_s \) more than \( s^2 \). Hence, the ratio \( H''/Z^\star \) is not greater than \( s \).

If \( L_k^s < 1 \), then \( p_k > s^2 - 1 \). Let us denote by \( R \) the sum of the successive tasks (\( p_k \) excluded). We know that \( L_k^s + p_k + R = 1+s \), that is \( L_k^s + R = 1+s-p_k < 2+s-s^2 \leq s^2 \) (the latter inequality holds for \( s \geq \frac{1+\sqrt{17}}{4} \)). Thus, \( p_k \) will be assigned to \( P_1 \) and the load \( R \) will be assigned to \( P_s \). The algorithm will produce a makespan \( H'' = \max(p_k,s) \). If \( p_k \leq 1 \), then the ratio is \( H''/Z^\star = s \), otherwise the ratio is \( H''/Z^\star \leq \max(p_k,s)/p_k = \max(1,\frac{s}{p_k}) < s \). \( \square \)

3.3 The range \( \sqrt{2} \leq s \)

Algorithm \( H''' \):

\[
\text{If } p_i + L_i^1 \leq 1 + \frac{1}{s+1} \text{ then assign } p_i \text{ to } P_1 \text{ else assign } p_i \text{ to } P_s
\]

**Theorem 11** Algorithm \( H''' \) is \((1 + \frac{1}{s+1}) = \frac{s+2}{s+1}\)-competitive for all \( s \geq 1 \).

**Proof.** Let us denote by \( x \) the final load \( L_1 \) of processor \( P_1 \) in the heuristic solution. By the definition of the algorithm, we have \( x \leq \frac{s+2}{s+1} \). If \( \frac{1}{s+1} \leq x \leq 1 \), then \( H''' = (s+1-x)/s \leq (s + \frac{s}{s+1})/s = 1 + \frac{1}{s+1} \); and if \( x > 1 \), then \( H''' = x \). Thus, \( H''' \leq 1 + \frac{s+2}{s+1} \) whenever \( \frac{1}{s+1} \leq x \leq \frac{s+2}{s+1} \).

Suppose \( x < \frac{1}{s+1} \), and denote \( y = \frac{1}{s+1} - x \). Then each task assigned to \( P_s \) is larger than \( 1+y \). Hence, either \( Z^\star = H''' \) or \( Z^\star \) is obtained by assigning the smallest task exceeding 1 to \( P_1 \) and all the other tasks to \( P_s \). In the latter case we have \( Z^\star > 1+y \) for some value \( 0 < y \leq \frac{1}{s+1} \). Thus,

\[
\frac{H'''}{Z^\star} < \frac{s+1-x}{s(1+y)} = \frac{s + \frac{s}{s+1} + y}{s(1+y)} < 1 + \frac{1}{s + 1}
\]

since the worst upper bound for nonnegative \( y \) would occur with \( y = 0 \). \( \square \)
References


