New bounds and algorithms for on-line scheduling: two identical processors, known sum and upper bound on the tasks

Enrico Angelelli[1†] and Maria Grazia Speranza[1] and Zsolt Tuza[2‡]

[1]*Dept. of Quantitative Methods, University of Brescia, C.da S. Chiara 50, I-25122 Brescia. Italy*
Email: `angele,speranza@eco.unibs.it`
[2]*Comp. and Autom. Inst., Hungarian Academy of Sciences, Budapest, Hungary and Dept. of Computer Science, University of Veszprem, Hungary. Email:* `tuza@sztaki.hu`

received Apr 21, 2005, accepted Dec 20, 2005.

In this paper we study a semi on-line version of the classical multiprocessor scheduling problem on two identical processors. We assume that the sum of the tasks and an upper bound γ on the size of each task are known. Each task has to be assigned upon arrival and the assignment cannot be changed later. The objective is the minimization of the maximum completion time on the processors. In this paper we propose new algorithms and improve known lower and upper bounds on the competitive ratio. Algorithms and bounds depend on the value of γ. An optimal algorithm, with respect to the competitive ratio, is obtained for $\gamma \in [\frac{1}{n}, \frac{2(n+1)}{n(2n+1)}] \cup \{\frac{2n-1}{2n(n-1)}\}$, where n is any integer value, $n \geq 2$.

Keywords: semi on-line scheduling, parallel processors, competitive analysis

1 Introduction

1.1 Description of the problem

In this paper we study semi on-line scheduling on two identical processors, where the sum of the tasks (normalized to 2, without loss of generality) and an upper bound γ on the size of the tasks are known in advance. Tasks arrive one at a time and must be immediately assigned to a processor. No changes are permitted on previous decisions. The objective is to minimize the makespan.

1.2 State of the art

The performance of on-line algorithms is often analyzed by studying their competitive ratio as introduced in [8]. We indicate with $A(I)$ the value of the solution obtained by an algorithm A on an instance I, and

[†]Corresponding author.
[‡]Research supported in part by the Hungarian Scientific Research Fund, grant OTKA T-049613.

with $T^*(I)$ the optimal value of an off-line algorithm, that is an algorithm that has perfect information on all the instance I. The *competitive ratio* r_A of algorithm A is defined as $r_A = \max_I \frac{A(I)}{T^*(I)}$. An on-line algorithm A is said to be *optimal* if no other algorithm A' has a competitive ratio $r_{A'} < r_A$.

For the on-line version of the multiprocessor scheduling problem on two processors the algorithm which assigns the incoming item to the least loaded processor (LS algorithm) is $\frac{3}{2}$-competitive and is optimal. The result is due to Graham [5], where it has been shown that the algorithm is $(2 - \frac{1}{m})$-competitive for m processors and to Faigle et al [4] who proved that the algorithm is optimal when $m = 2$ and $m = 3$.

Several semi on-line versions of the multiprocessor scheduling problem on two processors have been studied. Kellerer et al [7] have studied the cases where a buffer of fixed length is available to store the tasks, or the total size of the tasks is known, or two processors are available for the processing. For each of these problems an optimal $\frac{4}{3}$-competitive algorithm has been provided, which improves the $\frac{3}{2}$ of the on-line problem. A $\frac{4}{3}$ optimal algorithm has been obtained also for a randomized algorithm (see Bartal et al. [3]). Angelelli [1] has studied the semi on-line version on two processors where, in addition to the sum of the tasks, a lower bound β on the size of each task is known. He has shown that the performance of $\frac{4}{3}$ is improved by an optimal algorithm to $\max(2 - 2\beta, 1)$ if the lower bound β is greater than $\frac{1}{3}$. He and Zhang [6] have studied the semi on-line problem where both the bounds β and γ are known. They proved that the LS algorithm is optimal with performance $1 + \frac{\gamma-\beta}{2\beta}$.

The only semi on-line problem on two processors which has been studied in the literature and is still partially open is the problem with given sum of the tasks and known upper bound γ. This problem has been introduced in [2] where an optimal $\frac{4}{3}$–competitive algorithm has been presented for $\gamma \geq 1$. For $\gamma < 1$, the results are pictured in Figures 1, 2, and 3, where lower and upper bounds on the performance ratio of algorithms are illustrated. Namely, these figures show the bounds for $\gamma \in [\frac{1}{n}, \frac{1}{n-1}]$ for $n = 2$, $n = 3$ and n integer greater than 3, respectively.

1.3 Contribution of the paper

In this paper we improve on lower and upper bounds on the performance of algorithms for the problem described above. The results are summarized in Figures 4, 5, and 6. Our objective was to strengthen the bounds known for $\gamma \in [\frac{1}{2}, 1]$ to the intervals $[\frac{1}{n}, \frac{1}{n-1}]$ where n is any integer greater than 2. This has been completely achieved for the lower bounds. By comparing Figure 1 to Figures 5 and 6 we note that the shape given in Figure 1 has been reproduced in parametric form with respect to n on every interval $[\frac{1}{n}, \frac{1}{n-1}]$. Further relevant results, though partial, have been achieved for upper bounds. In Figure 4 we see that the upper bound on $[\frac{1}{2}, 1]$ has been improved; in Figure 5 and specially in Figure 6 we see the improvement over List Scheduling — the only algorithm which was previously available for small values of γ (large values of n). We note that the proposed algorithms are optimal for $\gamma \in [\frac{1}{n}, \frac{2(n+1)}{n(2n+1)}] \cup \{\frac{2n-1}{2n(n-1)}\} \subset [\frac{1}{n}, \frac{1}{n-1}]$.

It is still an open problem whether the algorithm, which is optimal for $\gamma = \frac{2n-1}{2n(n-1)}$, can be extended to a neighborhood of such a point.

1.4 Structure of the paper

In Section 2, we propose lower bounds on the performance of any algorithm for different values of γ in the range $[\frac{1}{n}, \frac{1}{n-1}]$, where n is any integer greater than or equal to 2. First, two lower bounds are proposed for $\gamma \in [\frac{2(n+1)}{n(2n+1)}, \frac{2n-1}{2n(n-1)}]$ and for $\gamma \in [\frac{2n-1}{2n(n-1)}, \frac{1}{n-1}]$. Next, the bounds are extended to the entire range

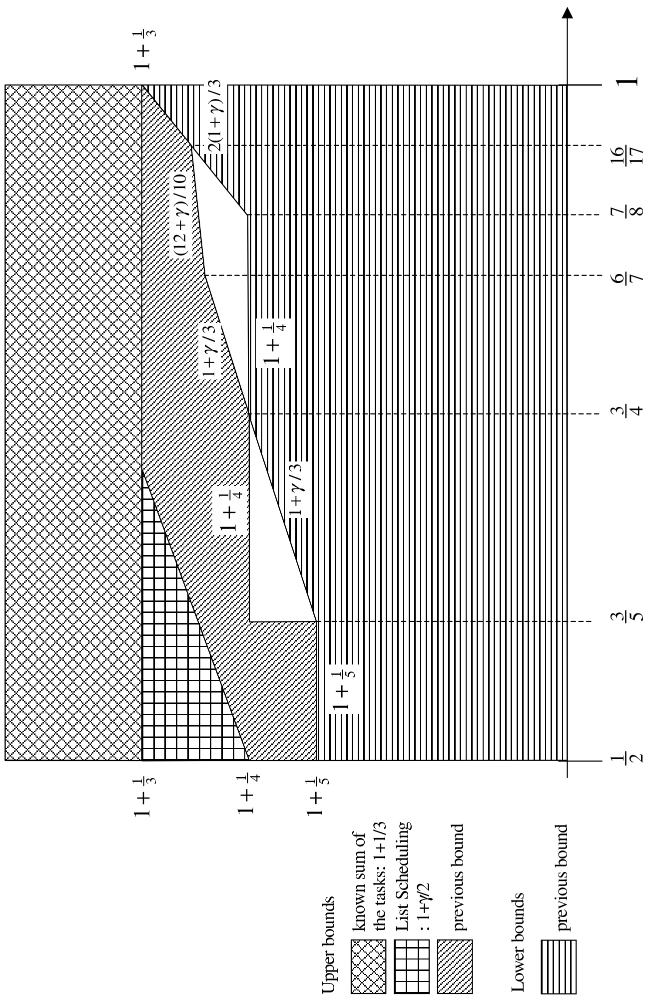

Fig. 1: Previous bounds for $\gamma \in [\frac{1}{2}, 1]$.

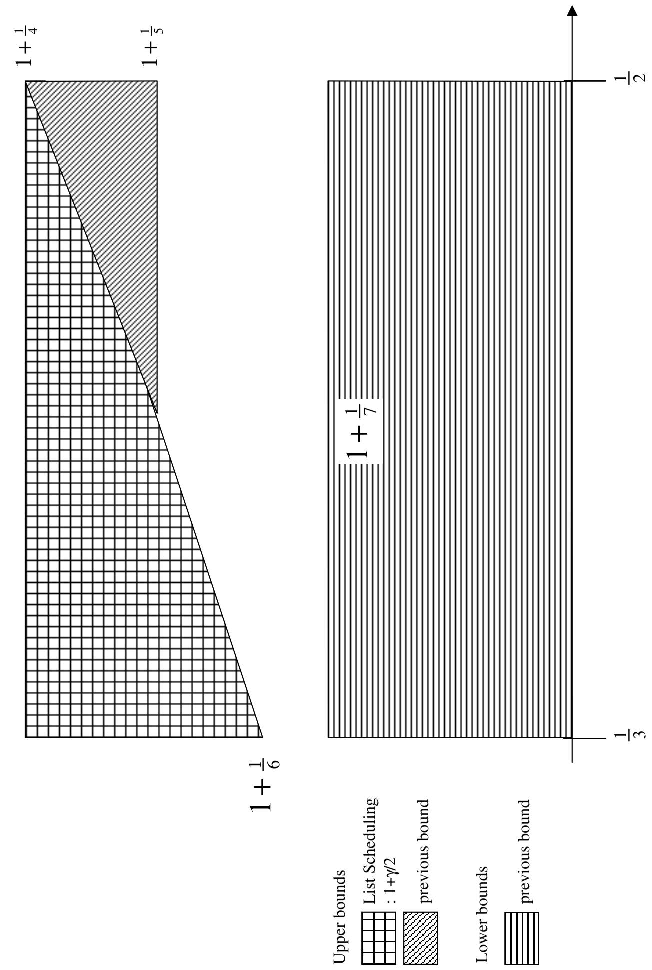

Fig. 2: Previous bounds for $\gamma \in [\frac{1}{3}, \frac{1}{2}]$.

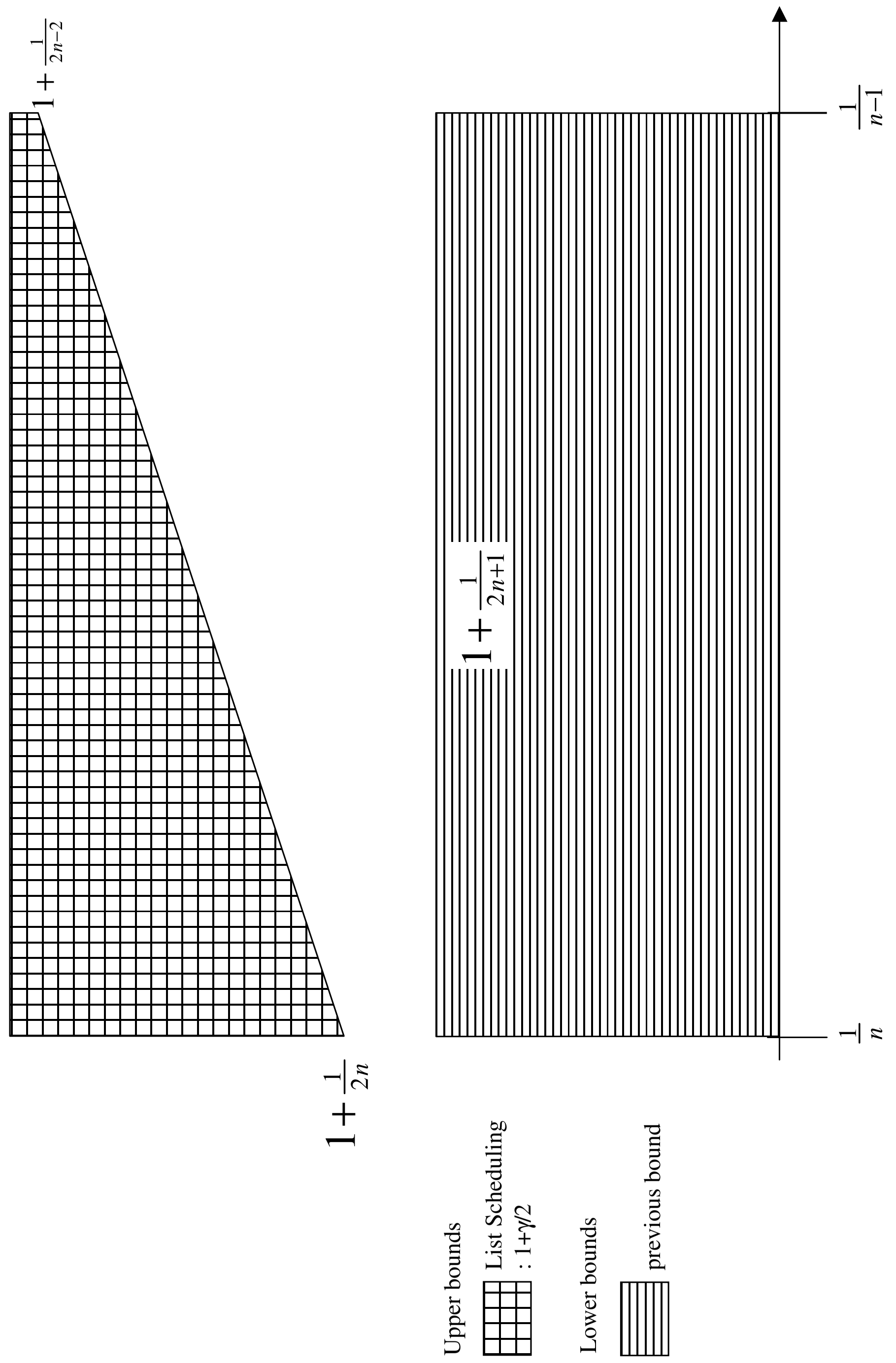

Fig. 3: Previous bounds for $\gamma \in [\frac{1}{n}, \frac{1}{n-1}]$ and $n > 3$.

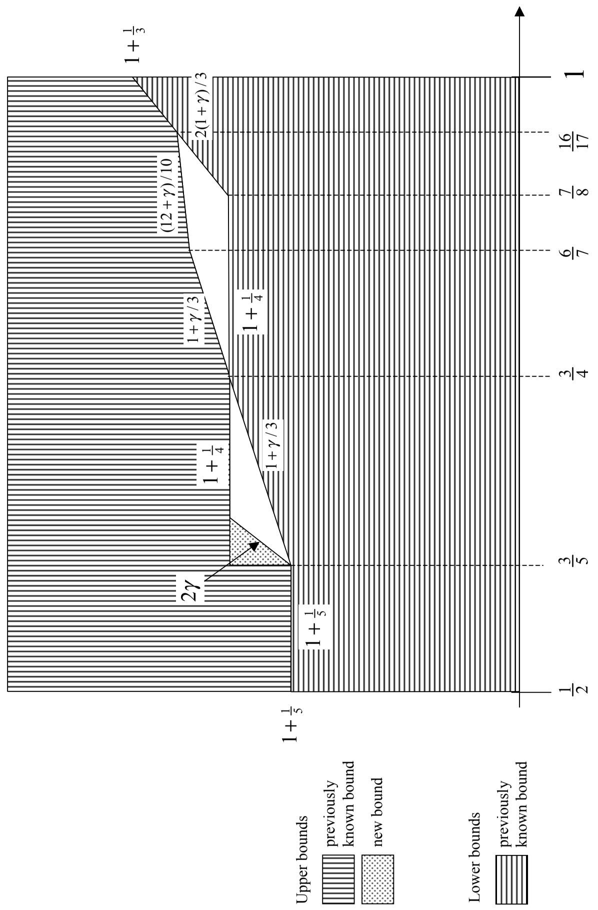

Fig. 4: Current bounds for $\gamma \in [\frac{1}{2}, 1]$.

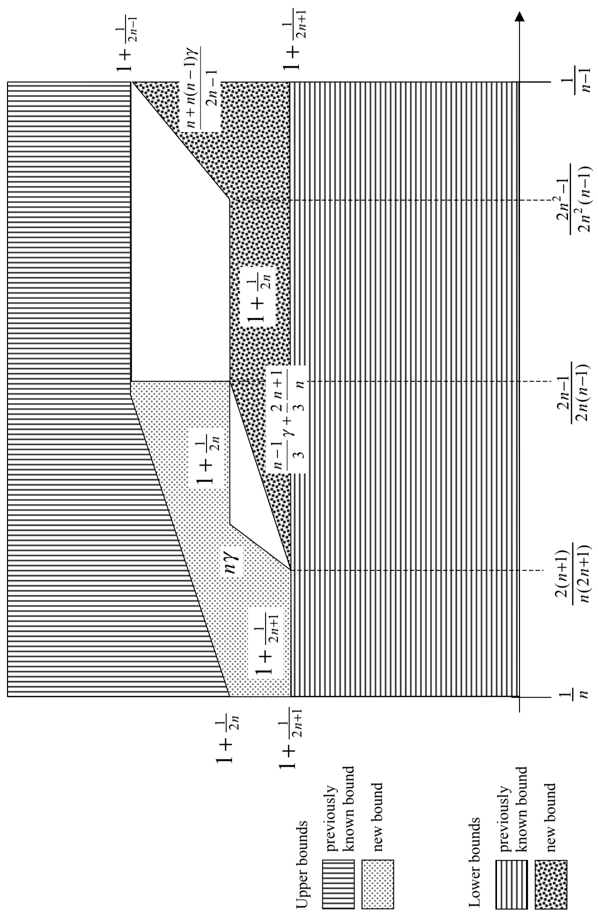

Fig. 5: Current bounds for $\gamma \in [\frac{1}{3}, \frac{1}{2}]$.

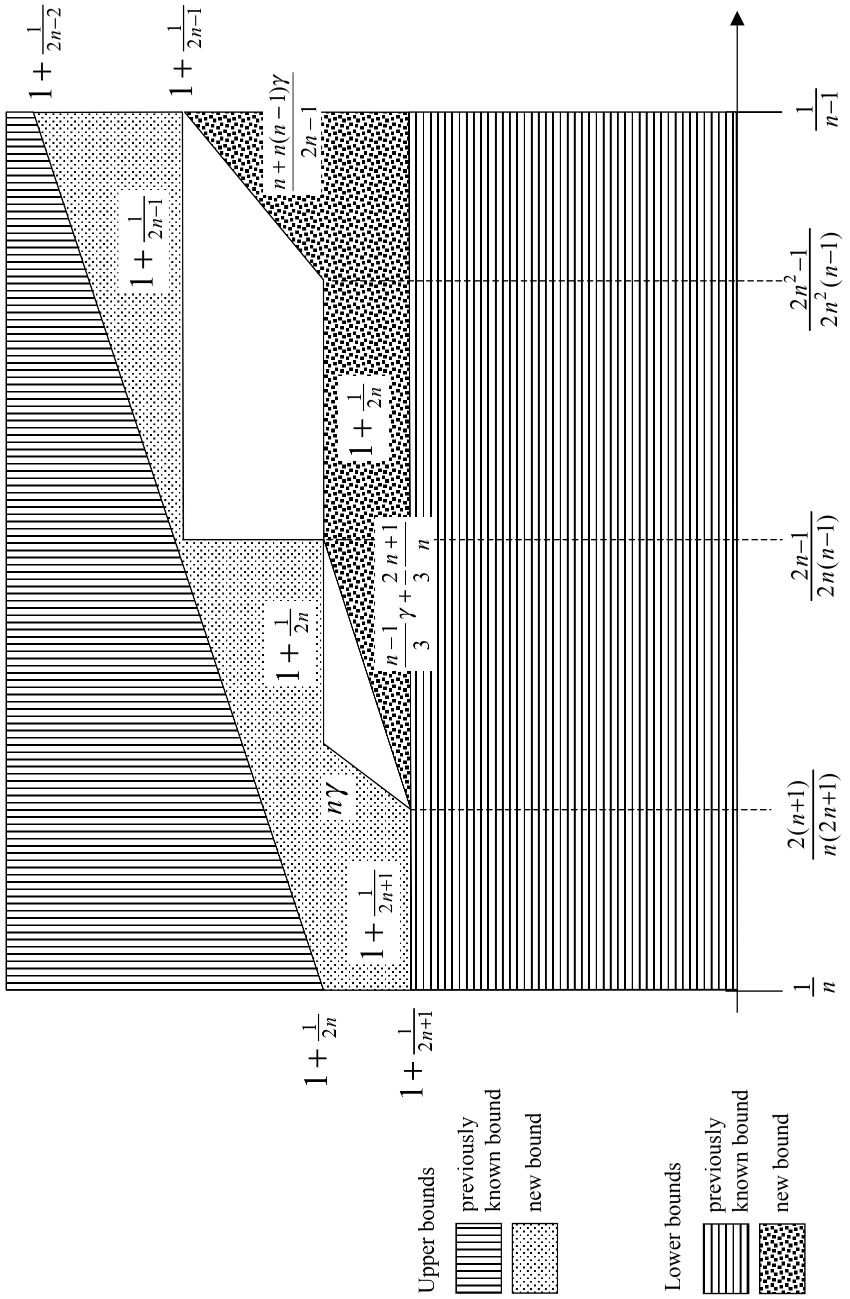

Fig. 6: Current bounds for $\gamma \in [\frac{1}{n}, \frac{1}{n-1}]$ and $n > 3$.

$[\frac{1}{n}, \frac{1}{n-1}]$. In Section 3, we propose three algorithms H', S, and H''. Algorithms H' and H'' provide the main results of the paper. Algorithm S is used as a "subprogram" for algorithm H''. Finally, some conclusions are given.

In this paper we denote with T^* and T, respectively, the off-line optimum makespan and the makespan obtained by an on-line algorithm.

2 Lower bounds

In this section we propose a lower bound on the performance of any algorithm for each value of γ in the interval $[\frac{1}{n}, \frac{1}{n-1}]$, where $n \geq 2$ can be any integer. We first consider lower bounds for $\gamma \in [\frac{2(n+1)}{n(2n+1)}, \frac{2n-1}{2n(n-1)}]$ and for $\gamma \in [\frac{2n-1}{2n(n-1)}, \frac{1}{n-1}]$, and then extend the bounds to the entire interval.

Lemma 1 *If $\gamma \in [\frac{2(n+1)}{n(2n+1)}, \frac{2n-1}{2n(n-1)}]$ for some fixed integer $n \geq 2$, then no on-line algorithm can have competitive ratio better than $\frac{n-1}{3}\gamma + \frac{2}{3}\frac{n+1}{n}$.*

Proof:

Let us consider two parametric instances I'_p, I''_p, where $p \in [0, \gamma]$ is a parameter whose value corresponds to the size of the first two tasks of each instance. The two instances are defined as follows:

$I'_p = \{p,\ p,\ \alpha,\ \alpha,\ 2(n-1)$ tasks of size $\gamma\}$ where the parameters p and α are fixed such that $p + \alpha + (n-1)\gamma = 1$ and $\alpha \leq p \leq \gamma/2$.

$I''_p = \{p,\ p,\ \beta,\ 2(n-1)$ tasks of size $\frac{1}{n}\}$ where the parameters p and β are fixed such that $2p + \beta + (n-2)\frac{1}{n} = 1$ and $p < \beta \leq \frac{1}{n}$.

While the off-line optimum for both I'_p and I''_p is $T^* = 1$, we note that in the optimal assignments for I'_p and I''_p the two tasks p are assigned, respectively, to different processors and to the same processor.

When two tasks of size p arrive first, any algorithm H has to decide whether it assigns them to different processors or to the same processor.

1) If algorithm H assigns the two tasks p to different processors, then on instance I''_p the makespan is

$$T \geq p + (n-1)\frac{1}{n} + \beta = 1 - p + \frac{1}{n}.$$

2) If algorithm H assigns the two tasks p to the same processor, then on instance I'_p the makespan is

$$T \geq 2p + (n-1)\gamma.$$

Thus, in either case $T/T^* \geq \min(1 - p + \frac{1}{n}, 2p + (n-1)\gamma)$. This lower bound clearly depends on the parameter p. By maximizing with respect to p, we obtain the bound

$$T/T^* \geq \frac{n-1}{3}\gamma + \frac{2}{3}\frac{n+1}{n}$$

for $p = \frac{n+1}{3n} - \frac{n-1}{3}\gamma$. Note that this value of the parameter fits all the implicit assumptions on p (i.e., $\alpha \leq p \leq \gamma/2, p < \beta \leq \frac{1}{n}$) when $\gamma \in [\frac{2(n+1)}{n(2n+1)}, \frac{2n-1}{2n(n-1)}]$.

□

Lemma 2 *If $\gamma \in [\frac{2n-1}{2n(n-1)}, \frac{1}{n-1}]$ for some fixed integer $n \geq 2$, then no on-line algorithm can have competitive ratio better than $\frac{n}{2n-1}(1+(n-1)\gamma)$.*

Proof:

Let us consider two parametric instances I'_p, I''_p, where $p \in [0, \gamma]$ is a parameter whose value corresponds to the size of the first two tasks of each instance. The two instances are defined as follows:

$I'_p = \{p,\ p,\ 2(n-1)$ tasks of size $\frac{1-p}{n-1}\}$, where the equality $p + (n-1)\frac{1-p}{n-1} = 1$ is an identity satisfied for all p.

Here we only have to impose the inequality $\frac{1-p}{n-1} \leq \gamma$, that is $p \geq 1-(n-1)\gamma$.

$I''_p = \{p,\ p,\ \alpha,\ \beta,\ (2n-3)$ tasks of size $\gamma\}$, where the parameters p, α and β are fixed such that $(n-1)\gamma + \alpha = 1$ and $p + p + (n-2)\gamma + \beta = 1$.

Obviously, we must guarantee that $\alpha, \beta \in [0, \gamma]$. While the condition $\alpha \in [0, \gamma]$ is ensured by the hypothesis $\gamma \in [\frac{1}{n}, \frac{1}{n-1}]$, the condition $\beta \in [0, \gamma]$ requires the assumption $p \in [\frac{1-(n-1)\gamma}{2}, \frac{1-(n-2)\gamma}{2}]$.

Summarizing the requirements on p induced by instances I'_p and I''_p, we assume the inequalities

$$p \geq 1-(n-1)\gamma \tag{1}$$

$$p \leq \frac{1-(n-2)\gamma}{2} \tag{2}$$

While the off-line optimum for both I'_p and I''_p is $T^* = 1$, we note that in the optimal assignments of I'_p and I''_p the two tasks p are assigned, respectively, to different processors and to the same processor.

Now, consider an algorithm which assigns the first two tasks to the same processor, and calculate the makespan it obtains on instance I'_p. Inequality (2) together with the condition $\gamma \geq \frac{1}{n}$ implies $p \leq \frac{1}{n}$. Thus, the inequalities

$$2p \leq 2\frac{1}{n} = 2\frac{1-\frac{1}{n}}{n-1} \leq 2(\frac{1-p}{n-1})1 < n(\frac{1-p}{n-1})$$

hold.

A set of $2n-2$ tasks of size $\frac{1-p}{n-1}$ must be assigned to the two processors which have loads $2p$ and 0, respectively. The makespan T produced by the algorithm is either $n\frac{1-p}{n-1}$ (case $2p \geq \frac{1-p}{n-1}$) or $1+p$ (case $2p < \frac{1-p}{n-1}$). Thus,

$$T \geq \min(n\tfrac{1-p}{n-1}, 1+p).$$

Consider now an algorithm which assigns the first two tasks to different processors, and calculate the makespan it obtains on instance I''_p. Observe that an odd number of tasks of size γ must be assigned to two processors whose loads are equal to p. Inequality (1) implies that the residual load $[\alpha + \beta]$ is not greater than γ. That is,

$$T \geq p + (n-1)\gamma.$$

Let us observe that $p + (n-1)\gamma \leq 1 + p$ for $\gamma \leq \frac{1}{n-1}$, and derive that no algorithm can guarantee a ratio better than $\min(p + (n-1)\gamma,\ n\frac{1-p}{n-1})$ on a set of instances whose off-line optimum is $T^* = 1$. The maximum value for such a lower bound is $\frac{n(1+(n-1)\gamma)}{2n-1}$, obtained when $p = \frac{n-(n-1)^2\gamma}{2n-1}$. In conclusion, for any algorithm, an instance exists such that

$$T/T^* \geq \frac{n + n(n-1)\gamma}{2n-1}.$$

□

Now we extend the previous lower bounds to the range $[\frac{1}{n}, \frac{1}{n-1}]$ for any fixed integer $n \geq 2$. The idea is that if a set of instances can be used to prove a lower bound $\underline{b}$ for a fixed γ', then the same set of instances can also be used to prove the same lower bound $\underline{b}$ for any other $\gamma > \gamma'$ because each task p satisfies $p \leq \gamma' < \gamma$.

Lemma 3 *If* $\gamma \geq \frac{1}{n}$ *for some fixed integer* $n \geq 2$*, then no algorithm can have competitive ratio better than* $1 + \frac{1}{2n+1}$.

Proof:
Define $m = n + 1$ and apply Lemma 2 for $\gamma = \frac{1}{m-1} = \frac{1}{n}$.

□

Remark: The same argument holds also for $n \geq 1$.

Lemma 4 *If* $\gamma \geq \frac{2n-1}{2n(n-1)}$ *for some fixed integer* $n \geq 2$*, then no algorithm can have competitive ratio better than* $1 + \frac{1}{2n}$.

Proof: Apply Lemma 1 for $\gamma = \frac{2n-1}{2n(n-1)}$.

□

In conclusion, by a selection of the highest lower bound available for each value of γ, the following general lower bounds can be drawn for $\gamma \in [\frac{1}{n}, \frac{1}{n-1}]$.

Theorem 5 *If* $\gamma \in [\frac{1}{n}, \frac{2(n+1)}{n(2n+1)}]$ *for some fixed integer* $n \geq 2$*, then no algorithm can have competitive ratio better than* $1 + \frac{1}{2n+1}$.

Proof:
This follows from Lemma 3.

□

Theorem 6 *If* $\gamma \in [\frac{2(n+1)}{n(2n+1)}, \frac{2n-1}{2n(n-1)}]$ *for some fixed integer* $n \geq 2$*, then no algorithm can have competitive ratio better than* $\frac{n-1}{3}\gamma + \frac{2}{3}\frac{n+1}{n}$.

Proof:
This is the statement of Lemma 1.

□

Theorem 7 *If $\gamma \in [\frac{2n-1}{2n(n-1)}, \frac{2n^2-1}{2n^2(n-1)}]$ for some fixed integer $n \geq 2$, then no algorithm can have competitive ratio better than $1 + \frac{1}{2n}$.*

Proof:
This follows from the bound given in Lemma 4, which is greater than the one in Lemma 2 for $\gamma < \frac{2n^2-1}{2n^2(n-1)}$.

□

Theorem 8 *If $\gamma \in [\frac{2n^2-1}{2n^2(n-1)}, \frac{1}{n-1}]$ for some fixed integer $n \geq 2$, then no algorithm can have competitive ratio better than $\frac{n}{2n-1}(1 + (n-1)\gamma)$.*

Proof:
This is the statement of Lemma 2 restricted to a shorter interval.

□

3 Algorithms

In this section we propose three algorithms H', S and H'' for $\gamma \in [\frac{1}{n}, \frac{1}{n-1}]$ where n is any integer greater than or equal to 2. While the focus is on the range $\gamma \in [\frac{1}{n}, \frac{1}{n-1}]$ for $n \geq 2$, some results for H' will be proved valid also for $n = 1$.

We shall apply the following notation. The current task is denoted by p. Processors are indicated by P_1 and P_2. We use index $i = 1, 2$ for processors P_1 and P_2, respectively, and then the variable P_{3-i} indicates the processor alternative to P_i. For the sake of avoiding too much notation, P_1 and P_2 are also used to represent the loads of the two processors.

3.1 Algorithm H'

1 – If $P_1 + p \leq 1 + \frac{1}{2n+1}$, then assign p to P_1

2 – else if $P_2 + p \leq 1 + \frac{1}{2n+1}$, then assign p to P_2

3 – else assign p to $\min(P_1, P_2)$

Lemma 9 *If $\gamma \geq \frac{2(n+1)}{n(2n+1)}$ for some fixed integer $n \geq 1$, then algorithm H' produces a makespan $T \leq n\gamma$.*

Proof:
Note that $\gamma \geq \frac{2(n+1)}{n(2n+1)}$ implies $1 + \frac{1}{2n+1} \leq n\gamma$. Thus we only need to consider instances such that $T > 1 + \frac{1}{2n+1}$. By construction, algorithm H' loads a processor more than $1 + \frac{1}{2n+1}$ only if neither rule (1) nor rule (2) is matched, that is, a task p' arrives when the two loads – say P_1' and P_2' – satisfy the inequality $p' + \min(P_1', P_2') > 1 + \frac{1}{2n+1}$. Thus, in this case we have

$$P_1' + P_2' + p' \leq 2 \tag{3}$$

$$P_1' + p' > 1 + \frac{1}{2n+1} = \frac{2n+2}{2n+1} \tag{4}$$

$$P_2' + p' > \frac{2n+2}{2n+1}, \tag{5}$$

where $P_2' = \sum_{i=1}^{k} p_i \le k\gamma$, for some $k \ge 0$

$$P_1' + p_i > \frac{2n+2}{2n+1}, \tag{6}$$

for all $i = 1, ..., k$.

Now we can derive a number of inequalities: from (3) and (5) we obtain

$$P_1' < \frac{2n}{2n+1}, \tag{7}$$

from (3) and (4) we obtain

$$P_2' < \frac{2n}{2n+1}, \tag{8}$$

from (6) and (7) we obtain for all i

$$p_i > \frac{2}{2n+1}, \tag{9}$$

from (4) and (7) we obtain

$$p' > \frac{2}{2n+1}, \tag{10}$$

and finally, we obtain

$$k < n, \tag{11}$$

from (9) and (8). In conclusion, $T = p' + \min(P_1', P_2') \le p' + P_2' \le k\gamma + \gamma \le n\gamma$. □

Lemma 10 *If $\gamma \le \frac{2(n+1)}{n(2n+1)}$ for some fixed integer $n \ge 1$, then H' produces a makespan $T \le 1 + \frac{1}{2n+1}$.*

Proof:
Suppose that an instance exists such that $T > 1 + \frac{1}{2n+1}$. By the same argument as in the proof of Lemma 9, we derive the contradiction $T \le n\gamma \le 1 + \frac{1}{2n+1}$.
□

Lemma 11 *If $\gamma \le \frac{1}{n-1}$ for some fixed integer $n \ge 2$, then H' has makespan $T \le 1 + \frac{1}{2n-1}$.*

Proof:
Define $m = n - 1$. Apply algorithm H' for $\gamma \le \frac{1}{m} \le \frac{2(m+1)}{m(2m+1)}$. Lemma 10 guarantees the makespan $T \le 1 + \frac{1}{2m+1} = 1 + \frac{1}{2n-1}$. □

Theorem 12 *If $\gamma \in [\frac{1}{n}, \frac{2(n+1)}{n(2n+1)}]$ for some fixed integer $n \ge 2$, then algorithm H' is $(1 + \frac{1}{2n+1})$-competitive; moreover, H' is optimal.*

Proof:
The competitive ratio follows from Lemma 10 and $T^* \geq 1$. Optimality follows from the lower bound in Theorem 5.

□

Theorem 13 *If* $\gamma \in [\frac{2(n+1)}{n(2n+1)}, \frac{1+2n}{2n^2}]$ *for some fixed integer* $n \geq 2$*, then algorithm* H' *is* $n\gamma$*-competitive.*

Proof:
The competitive ratio follows from Lemma 9 and $T^* \geq 1$.

□

Theorem 14 *If* $\gamma \leq \frac{1}{n-1}$ *for some fixed integer* $n \geq 2$*, then algorithm* H' *is* $(1 + \frac{1}{2n-1})$*-competitive; moreover,* H' *is optimal for* $\gamma = \frac{1}{n-1}$*.*

Proof:
The competitive ratio follows from Lemma 11 and $T^* \geq 1$. Optimality for $\gamma = \frac{1}{n-1}$ follows from the lower bound in Theorem 8.

□

3.2 Algorithm S

Theorem 12 guarantees that algorithm H' is optimal when $\gamma \in [\frac{1}{n}, \frac{2(n+1)}{n(2n+1)}]$ and Theorem 14 that algorithm H' is optimal if $\gamma = \frac{1}{n-1}$. Thus, we now need to focus on the range $\gamma \in (\frac{2(n+1)}{n(2n+1)}, \frac{1}{n-1})$ and look for an algorithm with a performance $1+\delta$ somewhere in between the lower bounds given by Theorems 6, 7 and 8 – which are certainly greater than $1 + \frac{1}{2n+1}$ – and the best performance guaranteed by List Scheduling and by the algorithm H'. Thus assume $\delta < \min(n\gamma - 1, \frac{\gamma}{2}, \frac{1}{2n-1})$.

It is easy to see that if a processor has load in the range $[1-\delta, 1+\delta]$ for a fixed $\delta < \gamma/2$, then it is possible – by assigning all the remaining tasks to the other processor – to control the makespan such that $T \leq 1+\delta$. On the other hand, if both processors have their loads in the range $(1+\delta-\gamma, 1-\delta)$, then it is not possible to find an assignment rule which guarantees $T \leq 1+\delta$ (a task of size γ will violate the bound $1+\delta$ on both processors). Now, the point is whether or not it is possible to find a set E of values such that if both processors are loaded less than $1+\delta$ and a processor has load in E, then the makespan can be controlled such that $T \leq 1+\delta$. With this objective in mind we define the *safe region* $E = \bigcup_{k=0}^{n} E^k$, where the sets $E^k = [1+\delta-2(k+1)\delta,\ 1+\delta-k\gamma]$, for $k = 0, ..., n$, are the component ranges.

Safe Algorithm S

(S.1) – If $p + P_i \in E$ for some $i = 1, 2$, then assign p to P_i

(S.2) – else if $P_i \in E^0$ for some $i = 1, 2$, then assign p to P_{3-i}

(S.3) – else if $P_i \in E$ for some $i = 1, 2$, then assign p to P_{3-i}

(S.4) – else assign p to $\min(P_1, P_2)$

Note that rules (S.2) and (S.3) are not redundant. Indeed, if $P_2 \in E^0$ and $P_1 \in E^1$, rule (S.2) assigns p to processor P_1, which is not guaranteed by rule (S.3) itself.

The following Lemma proves that the name *safe region* is appropriate.

Lemma 15 *If any of rules (S.1), (S.2) or (S.3) is matched by a task p when $P_1, P_2 \leq 1+\delta$, then all successive tasks will match either (S.2) or (S.3) and $T \leq 1+\delta$.*

Proof:

Let us assume the inequalities

$$P_1 \leq 1+\delta \tag{12}$$

$$P_2 \leq 1+\delta. \tag{13}$$

If rule (S.1) is matched (assume without loss of generality $p + P_1 \in E$), then task p is assigned to processor P_1. Inequalities (12) and (13) still hold, and obviously $P_1 \in E$ after the assignment of p to P_1.

If rule (S.2) is matched (assume $P_1 \in E^0$), then p is assigned to P_2. Inequalities $P_1 \geq 1-\delta$ and $P_1 + p + P_2 \leq 2$ imply $p + P_2 \leq 1+\delta$. Thus, inequalities (12) and (13) still hold and $P_1 \in E$.

If rule (S.3) is matched (assume $P_1 \in E \setminus E^0 = \bigcup_{k=1}^{n} E^k = \bigcup_{k=1}^{n} [1+\delta-2k\delta-2\delta,\ 1+\delta-k\gamma]$), task p is assigned to processor P_2. Now, we note first that $p < 2\delta$, since otherwise $p \in [2\delta, \gamma]$ implies $p + P_1 \in \bigcup_{k=0}^{n-1} E^k \subset E$ and rule (S.1) would apply. Secondly, processor P_2 is loaded less than $1-\delta$, since otherwise $P_2 \in E^0$ and rule (S.2) applies. Thus, inequalities (12) and (13) still hold and $P_1 \in E$.

In conclusion, if any of the rules (S.1), (S.2) or (S.3) is matched by a task p when $P_1, P_2 \leq 1+\delta$, all successive tasks will match either (S.2) or (S.3), while the inequalities (12) and (13) are guaranteed at each iteration.

□

The hypotheses upon which Lemma 15 is based are certainly satisfied when the initial load of the two processors is in the safe region, that is E contains 0. The following three lemmas exploit this lucky case, but unfortunately do not improve on the performance of algorithm H'.

Lemma 16 *If $\gamma \leq \frac{2(n+1)}{n(2n+1)}$ and $\delta = \frac{1}{2n+1}$ for some fixed integer $n \geq 2$, then algorithm S produces a makespan $T \leq 1 + \frac{1}{2n+1}$.*

Proof:

In this case $E^n = [0, \beta]$ where $\beta = 1 + \frac{1}{2n+1} - n\gamma \geq 0$. Since both processors start at 0, rule (S.3) is matched and Lemma 15 applies.

□

Lemma 17 *If $\gamma \geq \frac{2(n+1)}{n(2n+1)}$ and $\delta = n\gamma - 1$ for some fixed integer $n \geq 2$, then algorithm S produces a makespan $T \leq n\gamma$.*

Proof:

In this case $E^n = [\alpha, 0]$ where $\alpha = 2n - 2n^2\gamma + 2 - n\gamma \leq 0$. Since both processors start at 0, rule (S.3) is matched and Lemma 15 applies.

$\square$

Lemma 18 *If $\gamma \leq \frac{1}{n-1}$ and $\delta = \frac{1}{2n-1}$ for some fixed integer $n \geq 2$, then algorithm S produces a makespan $T \leq 1 + \frac{1}{2n-1}$.*

Proof:

In this case $E^{n-1} = [0, \beta]$ where $\beta = 1 + \frac{1}{2n-1} - (n-1)\gamma \geq 0$. Since both processors start at 0, rule (S.3) is matched and Lemma 15 applies.

$\square$

3.3 Algorithm H''

Since we are looking for a performance better than $1 + \min(n\gamma - 1, \frac{\gamma}{2}, \frac{1}{2n-1})$, and for $\delta < n\gamma - 1$ we have $\sup E^n < 0$, we will not take into account E^n anymore. Moreover, for $\delta < \frac{1}{2n-1}$ we have $\inf E^{n-1} > 0$, and this means that the initial load of the two processors is not in the safe region E, but in its complement with respect to $[0, 2]$. We shall call this complement the *risky region* A. That is, $A = A^0 \cup \bigcup_{k=1}^{n-1} A^k \cup A^n$, where the sets $A^n = [0, \inf E^{n-1})$, $A^k = (\sup E^k, \inf E^{k-1})$, for $k = 1, ..., n-1$, and $A^0 = (1 + \delta, 2]$ are the component ranges.

We already know that if the load of a processor happens to enter the safe region, then algorithm S guarantees, by Lemma 15, a makespan (and also a performance) $T \leq 1 + \delta$.

In the next lemma we investigate what happens if the arriving task p is such that Lemma 15 cannot be applied. In such a case, the loads of the two processors must be both in the risky region. Task p may be so small that it is not able to move the load of any processor from its current risky component to the next one. The following lemma takes into consideration the opposite case, i.e. when task p is big enough to force the load of a processor to move from a risky component to the next one. The claim is that if task p forces the load of one processor to move from range A^k to range A^{k-1}, skipping range E^{k-1}, then the same task, if assigned to the other processor, would force its load to exit its current risky range A^h and move either to the safe region E^{h-1} or to the risky range A^{h-1}.

Lemma 19 *Suppose that $P_1 \in A^k$ and $P_2 \in A^h$ for some $k, h > 1$ and $\delta \geq \max(\frac{1+(n-1)\gamma}{4n-1}, \frac{n-1}{2n-1}\gamma)$ for some fixed integer $n \geq 2$. If $p + P_1 \in A^{k-1}$, then $p + P_2 \notin A^h$. Conversely, if $p + P_2 \in A^{h-1}$, then $p + P_1 \notin A^k$.*

Proof:

The diameter of E^k is $d(E^k) = k(2\delta - \gamma) + 2\delta$ which decreases as k grows. The minimum is $2\delta n - n\gamma + \gamma$, for $k = n - 1$.

On the other hand, the diameter of A^n is $d(A^n) = \inf E^{n-1} = 1 + \delta - 2n\delta$, while the diameter of A^k is $k(\gamma - 2\delta)$ for $k = 1, ..., n-1$. The maximum is $(n-1)(\gamma - 2\delta)$, for $k = n - 1$. We see that $d(E^{n-1}) \geq d(A^n)$ if $\delta \geq \frac{1+(n-1)\gamma}{4n-1}$, and $d(E^{n-1}) \geq d(A^{n-1})$ if $\delta \geq \frac{n-1}{2n-1}\gamma$.

Thus, if $P_1 \in A^k$ and $p + P_1 \in A^{k-1}$, then $p > d(E^{k-1}) \geq d(E^{n-1}) \geq \max(d(A^{n-1}), d(A^n))$, and consequently $p + P_2 \notin A^h$.

□

Lemma 19 can be used to calculate lower bounds on some tasks of instances that keep an algorithm from loading the processors in the safe region.

Algorithm H''

H.1 – if $P_i \in E$ for some $i = 1, 2$, then run algorithm S

H.2 – if $p + P_i \in E$ for some $i = 1, 2$, then run algorithm S

H.3 – if $P_2 = 0$ and $p \in (1 + \delta - (n-1)\gamma, \frac{1+\delta-(n-2)\gamma}{2}]$, assign p to P_2

H.4 – if $p + P_1 \leq 1 + \delta$, assign p to P_1

H.5 – else assign p to $\min(P_1, P_2)$

Remark: If rule (H.1) or (H.2) is matched, then algorithm H'' is stopped and algorithm S takes decisions on task p and all the remaining tasks. Therefore, we can apply the bound derived for S, as follows.

Lemma 20 *If rule (H.1) or (H.2) is matched when $P_1, P_2 \leq 1 + \delta$, then $T \leq 1 + \delta$.*

Proof:
The result follows by Lemma 15.

□

It is interesting to analyze the performance of H'' on instances which force the loads to grow within the risky region and produce a makespan $T > 1 + \delta$. We have a result for a particular infinite sequence of values γ and δ.

Lemma 21 *Let $\gamma = \frac{2n-1}{2n(n-1)}$ and $\delta = \frac{1}{2n}$ for some fixed integer $n \geq 2$. If rules (H.1) and (H.2) are never matched when $P_1, P_2 \leq 1 + \delta$, then algorithm H'' produces a makespan T such that $T/T^* \leq 1 + \delta$.*

Proof:
If rules (H.1) and (H.2) are never matched, the makespan is necessarily greater than $1 + \delta$, because the loads of the two processors are forced to grow within the risky region A until at least one processor is loaded more than $1 + \delta$. We proceed as follows. We first prove that the makespan T is bounded from above by $1 + \frac{\gamma}{2} = \frac{4n^2-2n-1}{4n(n-1)}$, and then we prove that the off-line optimum T^* is bounded from below by $2\delta(n-1) + \frac{1+\delta-(n-2)\gamma}{2} = \frac{4n^2-4n+1}{4n(n-1)} > 1$.

Finally the ratio T/T^* is bounded from above by $\frac{4n^2-2n-1}{4n^2-4n+1} < 1 + \delta = 1 + \frac{1}{2n}$ for all n.

Let $\widetilde{p} \leq \gamma$ be the first task which forces $T > 1 + \delta$. When this task arrives, both processors have loads in the range $A^1 = (1 + \delta - \gamma, 1 - \delta)$, and $\widetilde{p} \in (2\delta, \gamma]$.

Upper bound for T. Assume, without loss of generality, that $P_1 < P_2$. If $P_1 > 1 - \frac{\gamma}{2}$, task $\widetilde{p}$ is assigned to P_1 and $T \leq 2 - P_2 < 1 + \frac{\gamma}{2}$, and otherwise if $P_1 \leq 1 - \frac{\gamma}{2}$, then $T \leq P_1 + \gamma \leq 1 + \frac{\gamma}{2}$.

Lower bound for T^*. We will show that instance I contains at least $2n-1$ "big" tasks: n of them are greater than $\frac{1+\delta-(n-2)\gamma}{2} = \frac{4n-3}{4n(n-1)}$ and the other $n-1$ are greater than $2\delta = \frac{1}{n}$. One of these big tasks is the already mentioned task $\widetilde{p} > 2\delta$. Note that $2\delta = \frac{1}{n} < \frac{4n-3}{4n(n-1)} = \frac{1+\delta-(n-2)\gamma}{2}$ for all n. This will be sufficient for proving that $\frac{4n^2-4n+1}{4n(n-1)}$ is a lower bound for the off-line optimum. Indeed, at least one processor must be assigned at least n big tasks, and the sum of them is bounded from below by $2\delta(n-1) + \frac{1+\delta-(n-2)\gamma}{2} = \frac{4n^2-4n+1}{4n(n-1)} > 1$ for all n.

Tasks on processor P_2. When $\widetilde{p}$ arrives, $P_2 \in A^1$. We claim that processor P_2 contains exactly $n-1$ tasks – say $p^{(i)}$ (for $i = 1, ..., n-1$) which are greater than 2δ. This is proved by the following argument. Only task $p^{(1)}$ can be possibly assigned to processor P_2 before processor P_1 reaches range A^1. In this case $p^{(1)} > 1 + \delta - (n-1)\gamma = 2\delta$.

All the other tasks which are assigned to P_2 (possibly including $p^{(1)}$) arrive when $P_1 \in A^1$. All these tasks must satisfy the inequality $p^{(i)} + P_1 > 1 + \delta$ which implies $p^{(i)} > 2\delta$, as $P_1 < 1 - \delta$. Moreover, since $p^{(i)} \in (2\delta, \frac{2n-1}{2n(n-1)}]$, we obtain that $n-2$ tasks of type $p^{(i)}$ are not sufficient to reach range A^1, while n tasks of type $p^{(i)}$ certainly make processor P_2 pass range A^1. In other words, processor P_2 has exactly $n-1$ tasks of size at most γ. The inequality $p^{(1)} + p^{(2)} > 1 + \delta - (n-2)\gamma$ implies that at least one of these two tasks is greater than $\frac{1+\delta-(n-2)\gamma}{2}$ (say, $p^{(1)} > \frac{1+\delta-(n-2)\gamma}{2}$).

Tasks on processor P_1. Similarly, processor P_1 must contain at least $n-1$ tasks which make the load of P_1 pass from A^k to A^{k-1} for every $k = n, ..., 1$. According to index k we enumerate such tasks $p_{(k)}$ in the reverse order of arrival. When $p_{(n-1)}$ arrives, processor P_2 is either empty or only loaded with a single task not greater than $\frac{1+\delta-(n-2)\gamma}{2}$. If $P_2 = 0$, then $p_{(n-1)} < \frac{1+\delta-(n-2)\gamma}{2}$ would be assigned to processor P_2 by rule (S.1) or (S.2). If $P_2 \in (1 + \delta - (n-1)\gamma, \frac{1+\delta-(n-2)\gamma}{2}] \subset A^{n-1}$, Lemma 19 and the assumption that the safe region cannot be reached imply $p_{(n-1)} > \frac{1+\delta-(n-2)\gamma}{2}$. In either case, $p_{(n-1)} > \frac{1+\delta-(n-2)\gamma}{2}$. Similarly, $p_{(k)} > \frac{1+\delta-(n-2)\gamma}{2}$ for $k = n-2, ..., 1$.

In conclusion, we have $2n-1$ big tasks. Among them, n tasks are greater than $\frac{1+\delta-(n-2)\gamma}{2}$, namely $p^{(1)}$ and the $p_{(k)}$ $(k = n-1, ..., 1)$. The other $n-1$ big tasks are greater than 2δ, these are $\widetilde{p}$ and the $p^{(i)}$ $(i = 2, ..., n-1)$. Thus, the off-line optimum is bounded from below by $2\delta(n-1) + \frac{1+\delta-(n-2)\gamma}{2}$.

□

Theorem 22 *If* $\gamma = \frac{2n-1}{2n(n-1)}$ *and* $\delta = \frac{1}{2n}$ *for some fixed integer* $n \geq 2$*, then algorithm* H'' *is* $(1 + \frac{1}{2n})$*–competitive; moreover,* H'' *is optimal.*

Proof:

If $\gamma = \frac{2n-1}{2n(n-1)}$ and $\delta = \frac{1}{2n}$, then either Lemma 20 or Lemma 21 applies. in both cases $T/T^* \leq 1 + \delta$. Optimality follows from Theorem 6.

□

In conclusion, for all integers $n \geq 2$, algorithm H' is optimal for $\gamma \in [\frac{1}{n}, \frac{2(n+1)}{n(2n+1)}]$ and $n\gamma$–competitive for $\gamma \in (\frac{2(n+1)}{n(2n+1)}, \frac{1+2n}{2n^2})$. Algorithm H'' is $(1 + \frac{1}{2n})$–competitive for $\delta = \frac{1}{2n}$ and $\gamma \in [\frac{1+2n}{2n^2}, \frac{2n-1}{2n(n-1)}]$;

in particular, it is optimal for $\gamma = \frac{2n-1}{2n(n-1)}$. For $\gamma \in (\frac{2n-1}{2n(n-1)}, \frac{1}{n-1}]$, the better performance is proven for algorithm H'.

References

[1] E. Angelelli. "Semi on-line scheduling on two parallel processors with known sum and lower bound on the size of the tasks", CEJOR, 8 (2000) 285–295.

[2] E. Angelelli, M.G. Speranza, Zs. Tuza. "Semi on-line scheduling on two parallel processors with upper bound on the items", Algorithmica, 37 (2003) 243–262.

[3] Y. Bartal, A. Fiat, H. Karloff, R. Vohra. "New algorithms for an ancient scheduling problem", J. Comput. Sys. Sci., 51 (1995) 359–366.

[4] U. Faigle, W. Kern, Gy. Turán. "On the performance of on-line algorithms for particular problems", Acta Cybernetica, 9 (1989) 107–119.

[5] R.L. Graham. "Bounds for certain multiprocessing anomalies", Bell System Technical Journal, 45 (1966) 1563–1581.

[6] Y. He, G. Zhang. "Semi on-line scheduling on two identical machines", Computing, 62 (1999) 179–187.

[7] H. Kellerer, V. Kotov, M.G. Speranza, Zs. Tuza. "Semi on-line algorithms for the partition problem", Oper. Res. Letters, 21 (1997) 235–242.

[8] D. Sleator, R.E. Tarjan, "Amortized efficiency of list update and paging rules", *Communications of ACM* 28 (1985) 202–208.