Angewandte Mathematik und Optimierung Schriftenreihe
Applied Mathematics and Optimization Series
AMOS # 28(2015)

György Dósa, Armin Fügenschuh, Zhiyi Tan, Zsolt Tuza,
and Krzysztof Węsek

Semi-Online Scheduling on Two Uniform Machines with Known Optimum Part II: Tight Upper Bounds

Herausgegeben von der
Professur für Angewandte Mathematik
Professor Dr. rer. nat. Armin Fügenschuh

Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg
Fachbereich Maschinenbau
Holstenhofweg 85
D-22043 Hamburg

Telefon: +49 (0)40 6541 3540
Fax: +49 (0)40 6541 3672

e-mail: appliedmath@hsu-hh.de
URL: http://www.hsu-hh.de/am

Angewandte Mathematik und Optimierung Schriftenreihe (AMOS), ISSN-Print 2199-1928
Angewandte Mathematik und Optimierung Schriftenreihe (AMOS), ISSN-Internet 2199-1936

Semi-Online Scheduling on Two Uniform Machines with Known Optimum, Part II: Tight Upper Bounds

György Dósa[*] Armin Fügenschuh [†] Zhiyi Tan [‡]
Zsolt Tuza [§¶] Krzysztof Węsek [‖**]

Abstract

We consider a semi-online version of the problem of scheduling a sequence of jobs of different lengths on two uniform machines with given speeds 1 and s. Jobs are revealed one by one (the assignment of a job has to be done before the next job is revealed), and the objective is to minimize the makespan. In the considered variant the optimal offline makespan is known in advance.

The most studied question for this online-type problem is to determine the optimal competitive ratio, that is, the worst-case ratio of the solution given by an algorithm in comparison to the optimal offline solution. In this paper, we make a further step towards completing the answer to this question by determining the optimal competitive ratio for s between $\frac{5+\sqrt{241}}{12} \approx 1.7103$ and $\sqrt{3} \approx 1.7321$, one of the intervals that were still open. Namely, we present and analyze a compound algorithm achieving the previously known lower bounds. Additionally, we implemented our algorithm and give some experimental results on its average performance.

The algorithm applies the method of "safe sets", with the novel addition that the conditions are not required in a strict way but exceptions are accepted and handled with additional techiques.

[*]Department of Mathematics, University of Pannonia, Veszprém, Hungary, dosagy@almos.vein.hu, **corresponding author**

[†]Helmut Schmidt University / University of the Federal Armed Forces Hamburg, Holstenhofweg 85, 22043 Hamburg, Germany, fuegenschuh@hsu-hh.de

[‡]Department of Mathematics, Zhejiang University, Hangzhou, Peoples Republic of China, tanzy@zju.edu.cn

[§]Department of Computer Science and Systems Technology, University of Pannonia, Veszprém, Hungary, tuza@dcs.uni-pannon.hu

[¶]Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences, Budapest, Hungary

[‖]Helmut Schmidt University / University of the Federal Armed Forces Hamburg, Holstenhofweg 85, 22043 Hamburg, Germany, wesekk@hsu-hh.de

[**]Faculty of Mathematics and Information Science, Warsaw University of Technology, ul. Koszykowa 75, 00-662 Warszawa, Poland, wesekk@mini.pw.edu.pl

Keywords: Scheduling, semi-online algorithm, makespan minimization, mixed-integer linear programming.

1 Introduction

Combinatorial optimization problems come with various paradigms on how an instance is revealed to a solving algorithm. The very common *offline* paradigm assumes that the entire instance is known in advance. On the opposite end, one can deal with the pure *online* scheme, where the instance is revealed part by part, unpredictable to the algorithm, and no further knowledge on these parts is assumed. In between these two extremes, and also highly relevant for many practical applications, are *semi-online* paradigms, where at least some characteristics of the instance in general are assumed to be known, for example, the total instance size or distributions of some internal values.

As a continuation of our work [10], we consider a semi-online variant of a scheduling problem for two uniform machines, that is defined as follows. Suppose that two machines `M1` and `M2` are processing a sequence of incoming jobs of varying lengths. Machine `M1` has a speed of 1, so that a job of length ℓ is processed within ℓ units of time, whereas machine `M2` has a speed of $s \geq 1$, so that a job of length ℓ can be processed within $\frac{\ell}{s}$ units of time. The jobs must be assigned to the machines in an online fashion, so that the next job becomes visible only when the previous job has already been assigned. The goal is to find a schedule that minimizes the total makespan, that is, the point in time when the last job on either machine is finished. We assume that the optimal value of the makespan for the corresponding offline problem (where all jobs are known in advance), denoted by OPT is available to the scheduler, and can be taken into account during its assignment decisions.

We are interested in constructing an algorithm $\mathcal{A}$ that solves this semi-online problem, and achieves a small makespan. Of course, for a given instance I of the problem, the (offline) $\mathrm{OPT} = \mathrm{OPT}(I)$ value is a lower bound for the semi-online problem. Thus, we consider the competitive ratio $\frac{M_{\mathcal{A}}(I)}{\mathrm{OPT}(I)} \geq 1$, where $M_{\mathcal{A}}(I)$ is the makespan value achieved by algorithm $\mathcal{A}$ when applied to instance I, as a performance measure.

The competitive ratio $r_{\mathcal{A}}$ of an algorithm $\mathcal{A}$ is then defined as the worst case of this ratio, that is, the supremum over all possible problem instances:

$$r_{\mathcal{A}} = \sup\left\{\frac{M_{\mathcal{A}}(I)}{\mathrm{OPT}(I)} : I \text{ is an instance}\right\}.$$

One can try to bound the value of r from below by estimating the infimum of $r_{\mathcal{A}}$ over all algorithms $\mathcal{A}$, that is,

$$r^* := \inf\{r_{\mathcal{A}} : \mathcal{A} \text{ an algorithm}\}.$$

We call r^* the optimal competitive ratio. An algorithm $\mathcal{A}$ is said to be r-competitive, if for any instance I its performance is bounded by r from above: $\frac{M_{\mathcal{A}}(I)}{\mathrm{OPT}(I)} \leq r$. An optimal algorithm in this sense is r^*-competitive.

1.1 Survey of the literature

The problem of scheduling a set of jobs on m (possibly not identical) machines with the objective to minimize the makespan (maximum completion time), with the jobs being revealed one-by-one, is a classic online algorithmic problem. Starting with results of Graham [17], much work has been done in this field (see for example [1, 7, 12, 14, 15, 16]), although even if we restrict only to the case of identical machines, the optimal ratio is still not known in general.

From both the theoretical and practical point of view, it may be important to investigate semi-online models, in which some additional information or relaxation is available. In this work we consider the scheme in which only the optimal offline value is known in advance (OPT version); however it is worth mentioning a strong relation with another semi-online version of the described scheduling problem, in which only the sum of jobs is known (SUM version) [2, 4, 5, 11, 18, 19, 20]. Namely, for a given number m of uniform (possibly non-identical) machines the optimal competitive ratio for the OPT version is at most the competitive ratio of the SUM version (see Dósa et al. [11]; for equal speeds this was first implicitly stated by Cheng et al. [8]).

For a more detailed overview of the literature on online and various semi-online variants, we refer to the survey of Tan and Zhang [21].

Azar and Regev [6] introduced the OPT version on (two or more) identical machines under the name of bin stretching, and this case was studied further by Cheng et al. [8] and by Lee and Lim [19]. However, knowing the relation between the OPT and SUM versions, the first upper bound for two equal-speed machines follows from the work of Kellerer et al. [18] on the SUM version.

In this work we are interested in the OPT version on two uniform machines with non-identical speeds, therefore we summarize previous results for this case. Recall that speeds of machines are 1 and s. Known bounds on the optimal competitive ratio r^* are expressed in terms of s.

Studies on this version of the problem were initiated by Epstein [13]. She provided the following bounds:

$$r^*(s): \begin{cases} r^*(s) \in \left[\frac{3s+1}{3s}, \frac{2s+2}{2s+1}\right] & \text{for } s \in [1, q_E \approx 1.1243] \\ r^*(s) \in \left[s\left(\frac{3}{4}+\frac{\sqrt{65}}{20}\right), \frac{2s+2}{2s+1}\right] & \text{for } s \in \left[q_E, \frac{1+\sqrt{65}}{8} \approx 1.1328\right] \\ r^*(s) = \frac{2s+2}{2s+1} & \text{for } s \in \left[\frac{1+\sqrt{65}}{8}, \frac{1+\sqrt{17}}{4} \approx 1.2808\right] \\ r^*(s) = s & \text{for } s \in \left[\frac{1+\sqrt{17}}{4}, \frac{1+\sqrt{3}}{2} \approx 1.3660\right] \\ r^*(s) \in \left[\frac{2s+1}{2s}, s\right] & \text{for } s \in \left[\frac{1+\sqrt{3}}{2}, \sqrt{2} \approx 1.4142\right] \\ r^*(s) \in \left[\frac{2s+1}{2s}, \frac{s+2}{s+1}\right] & \text{for } s \in \left[\sqrt{2}, \frac{1+\sqrt{5}}{2} \approx 1.6180\right] \\ r^*(s) \in \left[\frac{s+1}{2}, \frac{s+2}{s+1}\right] & \text{for } s \in \left[\frac{1+\sqrt{5}}{2}, \sqrt{3} \approx 1.7321\right] \\ r^*(s) = \frac{s+2}{s+1} & \text{for } s \geq \sqrt{3} \end{cases}$$

where q_E is the solution of $36x^4 - 135x^3 + 45x^2 + 60x + 10 = 0$.

Ng et al. [20] studied this problem with comparison to the SUM version. They presented algorithms giving the upper bounds

$$r^*(s) \le \begin{cases} \frac{2s+1}{2s} & \text{for } s \in \left[\frac{1+\sqrt{3}}{2}, \frac{1+\sqrt{21}}{4} \approx 1.3956\right] \\ \frac{6s+6}{4s+5} & \text{for } s \in \left[\frac{1+\sqrt{21}}{4}, \frac{1+\sqrt{13}}{3} \approx 1.5352\right] \\ \frac{12s+10}{9s+7} & \text{for } s \in \left[\frac{1+\sqrt{13}}{3}, \frac{5+\sqrt{241}}{12} \approx 1.7104\right] \\ \frac{2s+3}{s+3} & \text{for } s \in \left[\frac{5+\sqrt{241}}{12}, \sqrt{3}\right] \end{cases}$$

and proved the following lower bounds:

$$r^*(s) \ge \begin{cases} \frac{3s+5}{2s+4} & \text{for } s \in \left[\sqrt{2}, \frac{\sqrt{21}}{3} \approx 1.5275\right] \\ \frac{3s+3}{3s+1} & \text{for } s \in \left[\frac{\sqrt{21}}{3}, \frac{5+\sqrt{193}}{12} \approx 1.5744\right] \\ \frac{4s+2}{2s+3} & \text{for } s \in \left[\frac{5+\sqrt{193}}{12}, \frac{7+\sqrt{145}}{12} \approx 1.5868\right] \\ \frac{5s+2}{4s+1} & \text{for } s \in \left[\frac{7+\sqrt{145}}{19}, \frac{9+\sqrt{193}}{14} \approx 1.6352\right] \\ \frac{7s+4}{7s} & \text{for } s \in \left[\frac{9+\sqrt{193}}{14}, \frac{5}{3}\right] \\ \frac{7s+4}{4s+5} & \text{for } s \in \left[\frac{5}{3}, \frac{5+\sqrt{73}}{8} \approx 1.6930\right] \end{cases}$$

Dósa et al. [11] considered this version together with the SUM version. Their results included the bounds

$$r^*(s) \ge \begin{cases} \frac{8s+5}{5s+5} & \text{for } s \in \left[\frac{5+\sqrt{205}}{18}, \frac{1+\sqrt{31}}{6} \approx 1.0946\right] \\ \frac{2s+2}{2s+1} & \text{for } s \in \left[\frac{1+\sqrt{31}}{6}, \frac{1+\sqrt{17}}{4} \approx 1.2808\right] \end{cases}$$

$$r^*(s) \le \begin{cases} \frac{3s+1}{3s} & \text{for } s \in \left[1, q_D \approx 1.071\right] \\ \frac{7s+6}{4s+6} & \text{for } s \in \left[q_D, \frac{1+\sqrt{145}}{12} \approx 1.0868\right] \end{cases}$$

where q_D is the unique root of the equation $3s^2(9s^2-s-5) = (3s+1)(5s+5-6s^2)$.

Finally, the recent manuscript [10] whose results complement this work of the present authors provided the following lower bounds:

$$r^*(s) \ge \begin{cases} \frac{6s+6}{4s+5} & \text{for } s \in \left[\frac{\sqrt{21}+1}{4} \approx 1.3956, \frac{\sqrt{73}+3}{8} \approx 1.443\right] \\ \frac{12s+10}{9s+7} & \text{for } s \in \left[\frac{5}{3}, \frac{13+\sqrt{1429}}{30} \approx 1.6934\right] \\ \frac{18s+16}{16s+7}, & \text{for } s \in \left[\frac{13+\sqrt{1429}}{30}, \frac{30+7\sqrt{186}}{74} \approx 1.6955\right] \\ \frac{8s+7}{3s+10}, & \text{for } s \in \left[\frac{30+7\sqrt{186}}{74}, \frac{31+\sqrt{8305}}{72} \approx 1.6963\right] \\ \frac{12s+10}{9s+7} & \text{for } s \in \left[\frac{31+\sqrt{8305}}{72}, \frac{4+\sqrt{133}}{9} \approx 1.7258\right] \end{cases}$$

Here we collected only a brief summary of known bounds; for further details about previous results we refer to Dósa et al. [10].

1.2 Our contribution

After the work of Dósa et al. [10], between $\frac{5}{3}$ and $\sqrt{3}$ there are two intervals, namely $\left[\frac{13+\sqrt{1429}}{30}, \frac{31+\sqrt{8305}}{72}\right] \approx \left[1.6934, 1.6963\right]$ that we call *narrow interval* and $[\frac{5+\sqrt{241}}{12}, \sqrt{3}] \approx [1.7103, 1.7321]$ that we call *wide interval*, where the question remained open regarding the tight value of the competitive ratio.

In the narrow interval the upper bound is very close to the lower bound (the biggest gap is still smaller than 0.000209), so in this paper we focus on the wide interval, for which we present an optimal compound algorithm which has a competitive ratio that equals the previously known lower bounds.

We apply the method of "safe sets" that was introduced by Ng et al. [20] and independently by Angelelli et al. [3] (called "green set" in the latter), and also used by Dósa et al. [11]. Once those sets are properly defined (cf. Figure 2), we try to assign the next job in the sequence to a machine where its completion time will be in some safe set. In case of the quoted papers, the safe sets are defined in such a way that the next property holds in *any* case: after some initial phase when the loads of both machines are low, a job will surely arrive that can be assigned into a safe set. In other words, the boundaries of the safe sets are *optimized* in the way that the best possible competitive ratio would be reached while the above property holds.

Now, we make a tricky modification extending the power of the method. We realize that, keeping the above property, the algorithm *cannot be optimal* in the considered interval of speeds, therefore we do not insist on this property for defining the boundaries of the safe sets. We are less restrictive as we allow the possibility that during the scheduling process, some relatively big job may arrive, which cannot be assigned within a safe set. But it turns out that this *rare* case can be handled by another kind of algorithm. So, for any incoming job first we try our algorithm "Final Cases" which uses the safe sets, to assign the actual job into a safe set if possible. If this is not possible, we apply our second algorithm "Initial Cases" to assign the job. In this way we apply novel ideas for this semi-online problem.

We further show that our algorithm matches the best known algorithm of Ng et al. [20] on the interval $[\frac{1+\sqrt{13}}{3}, \frac{5+\sqrt{241}}{12}] \approx [1.5352, 1.7103]$ in the sense of competitive ratio. For a visual comparison of the previously known results and our contribution we refer to Figure 1. Whenever the dotted line (that represents an upper bound) is on an unbroken line (that represents a lower bound), the optimal competitive ratio is known.

2 Notions and definitions

Let $q_0 = \frac{1+\sqrt{13}}{3} \approx 1.5352$, which is the positive solution of $\frac{6s+6}{4s+5} = \frac{12s+10}{9s+7}$.
Let $q_6 := \frac{5+\sqrt{241}}{12} \approx 1.7103$, which is the positive solution of $\frac{12s+10}{9s+7} = \frac{2s+3}{s+3}$.
Let $q_7 := \frac{4+\sqrt{133}}{9} \approx 1.7258$, which is the positive solution of $\frac{12s+10}{9s+7} = \frac{s+1}{2}$.

We note that the values q_6 and q_7 were already defined in the paper [10].

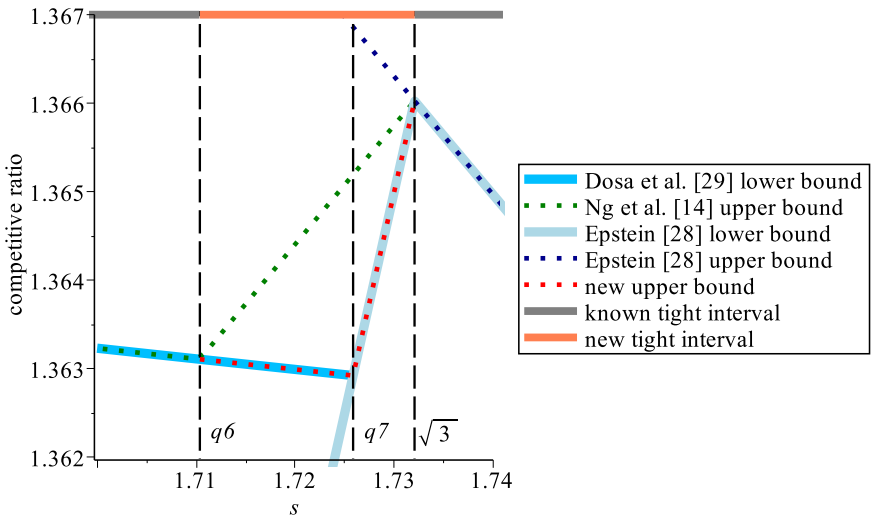

Figure 1: Known and new upper and lower bounds.

Then the wide interval is $[q_6, \sqrt{3}]$. For the remainder of this article we consider values of s from the wide interval only. We define

$$r(s) = \begin{cases} r_2(s) := \frac{12s+10}{9s+7} & \text{if } q_6 \le s \le q_7 \approx 1.7258 \quad \text{i.e. } s \text{ is regular} \\ r_5(s) = \frac{s+1}{2} & \text{if } q_7 \le s \le \sqrt{3} \quad \text{i.e. } s \text{ is large} \end{cases}$$

We remark that the value $r_2(s)$ is the same as in our preceding paper Dosa et al. [10]. The speeds to the left from the narrow interval (which are not considered in this paper) were called smaller regular speeds. The speeds to the right of the narrow interval were called bigger regular speeds, now we call these speeds simply as regular. The value $r_5(s)$ is Epstein's lower bound from [13] on the right side of the wide interval. Note also that the graph of $r_2(s)$ can be seen on the figure between q_6 and q_7, where the dotted line touches the unbroken line. Similarly, the graph of $r_5(s)$ appears between q_7 and $\sqrt{3}$, where the dotted line touches the unbroken line.

Let OPT and SUM mean, respectively, the known optimum value, and the total size of the jobs. Note that SUM $\le (s+1) \cdot OPT$, and the size of any job is at most $s \cdot OPT$. We denote the prescribed competitive ratio (that we do not want to violate) by r.

The optimum value is assumed to be known, and for sake of simplicity we will assume that OPT is equal to 1. (This can be assumed without loss of generality by normalization, i.e., dividing all of the job lengths by the optimal makespan.) Then we define five safe sets $S_i = [B_i, T_i]$ with size $D_i = T_i - B_i$ for $i = 1, \ldots, 5$ as follows (see also Figure 2):

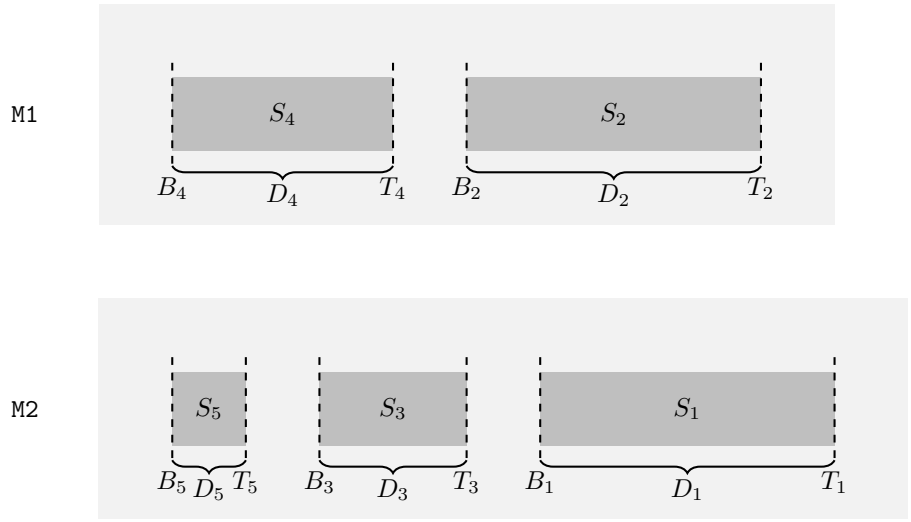

Figure 2: Safe sets.

1. $B_1 = s + 1 - r$ and $T_1 = rs$, thus $D_1 = (s + 1)(r - 1)$,
2. $B_2 = s + 1 - sr$ and $T_2 = r$, thus $D_2 = (s + 1)(r - 1)$,
3. $B_3 = 2s - 2r - rs + 2$ and $T_3 = s(r - 1)$, thus $D_3 = 2r - 3s + 2rs - 2$,
4. $B_4 = 4s - 2r - 3rs + 3$ and $T_4 = r - 1$, thus $D_4 = (3r - 4)(s + 1)$,
5. $B_5 = 6s-5r-4rs+6$ and $T_5 = 10s-7r-7rs+9$, thus $D_5 = 4s-2r-3rs+3$.

These sets define time intervals, and they are called "safe" because if the load of the machine is in this interval, this enables a "smart" algorithm (as the one we introduce later) to finish the schedule by not violating the desired competitive ratio. In other words, from the point of view of an algorithm (which wishes to keep the competitive ratio low), we want to assign the actual job in a way that the increased load of some machine will be inside a safe set.

3 Properties

Lemma 1 *$r_5(s) \geq r_2(s)$ for $s \geq q_7$.*

Proof. $r_5(s) - r_2(s) = \frac{s+1}{2} - \frac{12s+10}{9s+7} = \frac{9s^2-8s-13}{2(9s+7)} \geq 0$, which is true since $9s^2 - 8s - 13 \geq 0$ holds if and only if $s \leq \frac{4-\sqrt{133}}{9}$ or $s \geq \frac{4+\sqrt{133}}{9} = q_7$. ■

Lemma 2 *The following inequalities hold in the entire considered domain of the function r, i.e., for all $s \in [q_6, \sqrt{3}]$.*

1. $\frac{3s+2}{2s+2} < \frac{4}{3} < 1.35 < r(s) < \min\left\{\frac{4s+3}{3s+2}, \frac{s+2}{s+1}\right\} < \frac{2s+1}{s+1} < 2.$

2. $\frac{8s+7}{6s+5} \leq r(s).$

3. $\frac{s+3}{s+2} < \frac{7s+5}{5s+4} < \frac{s+1}{2} \leq r(s) < \frac{6s+6}{4s+5}.$

Proof. The rightmost part in Lemma 2.1, i.e. $\frac{2s+1}{s+1} < 2$, holds trivially. All other claims in 2.1 and 2.2 but the ones which regard $r_5(s)$ are already proven in [10], thus we give only this unproved part here. Moreover, we give the proof for 2.3, whose claims were not considered before.

1. The leftmost lower bound holds as $\frac{3s+2}{2s+2} < \frac{4}{3}$ is equivalent to $4(2s+2) - 3(3s+2) = 2 - s > 0$, and hence to $s < 2$. Further, it is easy to see that $r(s) = \frac{s+1}{2} > 1.35$, since $s > 1.7$ in the domain of r_5. Regarding the upper bound, $\frac{2s+1}{s+1} > \frac{s+2}{s+1}$ holds trivially since $s > 1$, thus it remains to show that $r < \min\left\{\frac{s+2}{s+1}, \frac{4s+3}{3s+2}\right\}$. Note that $\frac{4s+3}{3s+2} \geq \frac{s+2}{s+1}$ for positive s holds if and only if $(4s+3)(s+1) - (s+2)(3s+2) = s^2 - s - 1 \geq 0$, i.e., $s \geq \frac{1+\sqrt{5}}{2} \approx 1.618$. Therefore, for large s we need to show only that $r < \frac{s+2}{s+1}$. We have $\frac{s+1}{2} - \frac{s+2}{s+1} < 0$, which holds since $s^2 - 3 \leq 0$ is true.

2. For large s we get that $\frac{s+1}{2} \geq \frac{8s+7}{6s+5}$ holds if and only if $(6s+5)(s+1) - 2(8s+7) = 6s^2 - 5s - 9 \geq 0$, i.e., $s \geq \frac{5+\sqrt{241}}{12} \approx 1.7103 = q_6$ which is valid.

3. Regarding the leftmost inequality, $\frac{7s+5}{5s+4} - \frac{s+3}{s+2} = \frac{2(s-1)(s+1)}{(s+2)(5s+4)} > 0$ trivially holds. The next inequality holds since $\frac{s+1}{2} - \frac{7s+5}{5s+4} = \frac{5s^2-5s-6}{2(5s+4)} > 0$ holds if $s > \frac{5+\sqrt{145}}{10} \approx 1.7042$ (and this value is smaller than q_6). Regarding $r(s) \geq \frac{s+1}{2}$, for large speeds the inequality holds trivially (with equality) and for regular speeds we have already seen the validity of the inequality in Lemma 1. Thus we are done with the lower bound; let us see the upper bound. For regular s we have $\frac{6s+6}{4s+5} - \frac{12s+10}{9s+7} = \frac{6s^2-4s-8}{(9s+7)(4s+5)} \geq 0$, which is true, since $6s^2 - 4s - 8 \geq 0$ for $s \leq \frac{1-\sqrt{13}}{3}$ and $s \geq \frac{1+\sqrt{13}}{3} = q_0 \approx 1.535$. For large s we have $\frac{6s+6}{4s+5} - \frac{s+1}{2} = \frac{-4s^2+3s+7}{2(4s+5)} = \frac{(s+1)(7-4s)}{2(4s+5)} \geq 0$, which is true since $s \leq 1.75$.

■

In the next lemma we state some properties of the safe sets. Note that an alternative option to define the safe sets would be to require these properties below.

Lemma 3 *1.* $D_1 = D_2$,

2. $T_1 - T_3 = s$ *and* $T_2 - T_4 = 1$,

3. $B_3 = B_1 - D_1$,

4. $B_4 = B_2 - D_3$,

5. $B_5 = B_3 - D_4$,

6. $T_5 = B_5 + B_4$.

Proof. Proofs of the equalities in Lemma 3.1 to 3.4 were given in Dosa et al. [10]. Since these proofs use nothing else than the definition of the safe sets, we do not repeat them. For proving 3.5 and 3.6 we use again the definitions of the boundaries.

5. $B_5 + D_4 = (6s - 5r - 4rs + 6) + (3r - 4)(s + 1) = 2s - 2r - rs + 2 = B_3$.

6. $B_5 + B_4 = (6s - 5r - 4rs + 6) + (4s - 2r - 3rs + 3) = 10s - 7r - 7rs + 9 = T_5$.

■

The next lemma proves that the safe sets are well defined in the sense that they are disjoint sets, and follow each other in the described order on the machines.

Lemma 4 *The following inequalities hold:*

1. $0 \leq B_4 < T_4 < B_2 < T_2$,

2. $0 < B_5 < T_5 \leq B_3 < T_3 < B_1 < T_1$.

Proof. We note that in the paper Dosa et al. [10] we already introduced the first four safe set (in the same way), with the same properties. In this paper we need the fifth safe set as well, moreover the claims of the lemma hold also for large s values, thus we need to give the proof of the lemma again. In the calculations we generally use Lemma 2, unless stated otherwise.

1. From $r \leq \frac{4s+3}{3s+2}$ it follows that $0 \leq 4s + 3 - 3rs - 2r = B_4$. From $r > \frac{4}{3}$ and the definition we have that $0 < (3r-4)(s+1) = D_4 = T_4 - B_4$. From $r < \frac{s+2}{s+1}$ it follows $0 < (s + 1 - sr) - (r - 1) = B_2 - T_4$. By $r > 1$ we have that $0 < (s + 1)(r - 1) = T_2 - B_2$.

2. We observe that for positive s the inequality $0 < 6s - 5r - 4rs + 6 = B_5$ is equivalent to $r(s) < \frac{6s+6}{4s+5}$, which holds. Lemma 3.6 states that $T_5 - B_5 = B_4$, and thus using $B_4 > 0$ from Lemma 4.1 we have $T_5 - B_5 > 0$. From $r \geq \frac{8s+7}{6s+5}$ it follows that $0 \leq 5r - 8s + 6rs - 7 = (2s - 2r - rs + 2) - (10s - 7r - 7rs + 9) = B_3 - T_5$. From $r > \frac{3s+2}{2s+2}$ it follows that $0 < 2r + 2rs - 3s - 2 = D_3 = T_3 - B_3$. From $r < \frac{2s+1}{s+1}$ it follows that $0 < (s + 1 - r) - s(r - 1) = B_1 - T_3$. By $r > 1$ we have that $0 < (s + 1)(r - 1) = D_1 = T_1 - B_1$.

■

We will need some further properties regarding the safe sets. These properties make the later calculations easier.

Lemma 5 *1.* $D_1 = D_2 > \max\{B_2, D_3\}$,

2. $B_2 < 1$ *and* $B_1 < s$,

3. $T_3 - T_5 \geq B_2$,

4. $B_2 \geq B_3$,

5. $T_2 \geq B_1$,

6. $D_3 > B_4$,

7. $T_4 + D_3 > B_2$,

8. $2D_1 > s$,

9. $T_4 + D_1 > 1$,

10. $T_4 + T_2 \geq s$.

Proof. We generally use Lemma 2 for the lower or upper bounds on $r(s)$.

1. $D_1 = D_2$ holds directly by definition. For $D_2 > B_2$ we equivalently have $D_2 - B_2 = (s+1)(r-1) - (s+1-sr) = 2sr - 2s - 2 + r > 0$, and hence $r(2s+1) > 2s+2$, which holds. Finally, from $D_2 - D_3 = (rs + r - s - 1) - (2r - 3s + 2rs - 2) = -rs - r + 2s + 1 > 0$ we get $\frac{2s+1}{s+1} > r$, which is true.

2. $B_2 = s + 1 - sr < 1$, and $B_1 = s + 1 - r < s$ since $1 < r$.

3. We have $T_3 - T_5 - B_2 = s(r-1) - (10s - 7r - 7rs + 9) - (s + 1 - sr) = 7r - 12s + 9rs - 10 \geq 0$ if and only if $r \geq \frac{12s+10}{9s+7}$. This is trivially true for any $s \leq q_7$, and true for $s > q_7$ by Lemma 1.

4. We have $B_2 - B_3 = (s + 1 - sr) - (2s - 2r - rs + 2) = 2r - s - 1 \geq 0$ if and only if $r \geq \frac{s+1}{2}$, which holds.

5. $T_2 - B_1 = r - (s + 1 - r) = 2r - s - 1 \geq 0$.

6. $D_3 - B_4 = (2r - 3s + 2rs - 2) - (4s - 2r - 3rs + 3) = 4r - 7s + 5rs - 5 > 0$ if and only if $r > \frac{7s+5}{5s+4}$.

7. $T_4 + D_3 - B_2 > B_4 + D_3 - B_2 = 0$, by Lemmas 3.4 and 4.1.

8. $2D_1 - s = 2(s+1)(r-1) - s = 2r - 3s + 2rs - 2 > 0$ holds if $r > \frac{3s+2}{2s+2}$, which is true.

9. $T_4 + D_1 - 1 = (r-1) + (s+1)(r-1) - 1 = 2r - s + rs - 3 > 0$ if and only if $r > \frac{s+3}{s+2}$.

10. $T_4 + T_2 - s = (r-1) + r - s = 2r - s - 1 \geq 0$ since $r \geq \frac{s+1}{2}$.

■

4 Algorithm `FinalCases`

First the loads are zero. The actual load of the machines will be denoted as L_m ($m = 1$ or $m = 2$) just before assigning the next job. Thus, for example, if L_1 denotes the actual load of the first machine, then after assigning a job to this machine, the new load will again be denoted by L_1.

Here we define a servant algorithm, which works (and will be applied) *only if* the next job can be assigned to a machine whose increased load will be within some safe set. We call the algorithm `FinalCases`. We will say, for the sake of simplicity, that some step is *executed* if the condition of this step is satisfied and the actual job is assigned at this step. Otherwise we say that the step is only *examined*. In other words, entering some step, it is examined whether the condition of the step is fulfilled or not. If yes, the step is executed. If not, the step is not executed. Moreover, for the sake of simplicity, if some step is not executed, we do not write "else if" in the description of the algorithm; if it turns out that the condition of some step is not satisfied, then the algorithm simply proceeds with examining the next step.

Theorem 6 *Suppose that some of Steps 1 to 5 of Algorithm* `FinalCases` *is executed. Then all subsequent jobs are also scheduled by this algorithm, and the competitive ratio is not violated.*

Proof.

1. Suppose that Step 1 is executed. Then the load L_2 of the fast machine `M2` will be not more than $T_1 = rs$, thus we do not violate the competitive ratio r by the fast machine. On the other hand, the final load of the fast machine is at least $B_1 = s + 1 - r$, because we assigned job x_i to `M2`. Applying SUM $\leq s+1$, the final load L_1 of the slow machine `M1` cannot be more than r, since $L_1 = \text{SUM} - L_2 \leq (s+1) - (s+1-r) = r$, which means that the competitive ratio is not violated by the slow machine either.

2. Now suppose that Step 2 is executed. The proof is almost the same as for Step 1. The load of `M1` does not exceed T_2, so the competitive ratio is not violated by the slow machine. Moreover the final load of the slow machine is $L_1 \geq B_2 = s + 1 - sr$, thus $L_2 \leq \text{SUM} - L_1 \leq (s + 1) - B_2 = sr = T_1$, and we are done.

3. Suppose that Step 3 is executed. After assigning x_i to `M2`, $B_3 \leq L_2 \leq T_3$ holds. Then we possibly assign several jobs to `M1`. We claim that the increased load of `M1` cannot remain below B_2. Indeed, assume that it stays below B_2. Then $B_2 < 1$ from Lemma 5.2, and also $\frac{T_3}{s} = r - 1 < 1$ from the rightmost estimation in Lemma 2.1. Hence the makespan would be strictly less than OPT = 1; a contradiction. Thus there must arrive a job that ends the loop, i.e. some job x_j with $L_1 + x_j \geq B_2$. At this point the algorithm goes back to Step 1. We claim that with this job x_j the

Algorithm 1: `FinalCases`

Data: current loads L_1, L_2 for machines `M1`, `M2`; index i of current job x_i

if $L_2 + x_i \in S_1$ **then**
- $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
- $L_1 := L_1 + \sum_{j=i+1}^{N} x_j$ `// assign all subsequent jobs to M1`
- **stop** `// no more jobs, terminate`

if $L_1 + x_i \in S_2$ **then**
- $L_1 := L_1 + x_i$ `// assign job` x_i `to M1`
- $L_2 := L_2 + \sum_{j=i+1}^{N} x_j$ `// assign all subsequent jobs to M2`
- **stop** `// no more jobs, terminate`

if $L_2 + x_i \in S_3$ **and** $L_1 < B_2$ **then**
- $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
- **while** $L_1 + x_{i+1} < B_2$ **do**
  - $i := i + 1$ `// next job`
  - $L_1 := L_1 + x_i$ `// assign job` x_i `to M1`
- $i := i + 1$ `// next job`
- **goto** Step 1

if $L_1 + x_i \in S_4$ **and** $L_2 < B_3$ **then**
- $L_1 := L_1 + x_i$ `// assign job` x_i `to M1`
- **while** $L_2 + x_{i+1} < B_3$ **do**
  - $i := i + 1$ `// next job`
  - $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
- $i := i + 1$ `// next job`
- **if** $L_2 + x_i \in S_1$ **or** $L_1 + x_i \in S_2$ **or** $L_2 + x_i \in S_3$ **then**
  - **goto** Step 1
- **while** $L_2 + x_i < B_1$ **do**
  - $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
  - $i := i + 1$ `// next job`
- **goto** Step 1

if $L_2 + x_i \in S_5$ **and** $L_1 \leq B_4$ **then**
- $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
- **while** $L_1 + x_{i+1} < B_4$ **do**
  - $i := i + 1$ `// next job`
  - $L_1 := L_1 + x_i$ `// assign job` x_i `to M1`
- $i := i + 1$ `// next job`
- **if** $L_2 + x_i \in S_1$ **or** $L_1 + x_i \in S_2$ **or** $L_2 + x_i \in S_3$ **or** $L_1 + x_i \in S_4$ **then**
  - **goto** Step 1
- **while** $L_2 + x_i < B_1$ **do**
  - $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
  - $i := i + 1$ `// next job`
- **goto** Step 1

return `// back to the main program, if used as subroutine`

condition of Step 1 or Step 2 is satisfied, so the algorithm will assign all remaining jobs as explained above, and does not violate the competitive ratio.

Suppose that the condition of Step 2 is not satisfied, i.e., $L_1 + x_j \notin S_2$. Together with the previously satisfied condition $L_1 + x_j \geq B_2$, we deduce that $L_1 + x_j > T_2$, from which it follows that $x_j > D_2$. We show that in this case the condition of Step 1 is already fulfilled. Indeed, for the lower bound we have $L_2 + x_j > B_3 + D_2 = B_3 + D_1 = B_1$ (where from left to right we applied the condition of Step 3, the definitions of D_1 and D_2, and Lemma 3.3), while for the upper bound we have $L_2 + x_j \leq T_3 + x_j = s(r-1) + x_j = sr - s + x_j = T_1 - s + x_j \leq T_1$ (where from left to right we applied the condition of Step 3, the definitions of T_3 and T_1, and the inequality $x_j \leq s$ due to the fact that longer jobs would exceed OPT = 1 even on the fast machine). So we are entering Step 1 or Step 2.

4. Suppose that Step 4 is executed. After assigning x_i to `M1`, $B_4 \leq L_1 \leq T_4$ holds. Then we possibly assign several jobs to `M2`. We claim that the increased load of `M2` cannot remain below B_3. Indeed, assume that it stays below B_3. Then $L_1 \leq T_4 = r - 1 < 1$ from Lemma 2.1, moreover $\frac{B_3}{s} < \frac{B_1}{s} < 1$, where we use Lemmas 4.2 and 5.2. Hence the makespan would be strictly less than OPT = 1; a contradiction. Thus there must arrive a job that ends the loop, i.e., some job x_j with $L_2 + x_j \geq B_3$.

   If $L_2 + x_j$ is in S_1, or $L_1 + x_j$ is in S_2, or $L_2 + x_j$ is in S_3, we go back to Step 1. If Step 1 or Step 2 is executed, we are done. Otherwise the condition of Step 3 will be examined. We know that the condition $L_2 + x_j \in S_3$ is fulfilled. Observe that the second condition of Step 3, i.e. $L_1 \leq B_2$ also holds, since $L_1 \leq T_4$ still holds and we have $T_4 < B_2$ from Lemma 4.1. Thus Step 3 is executed, and we are done.

   Now assume that none of the conditions $L_2 + x_j \in S_1$, $L_1 + x_j \in S_2$, or $L_2 + x_j \in S_3$ is satisfied. Let us consider the size of the actual job, x_j. Since $L_2 + x_j \geq B_3$ (from the choice of x_j), but $L_2 + x_j$ is not in S_3, we deduce that $L_2 + x_j > T_3$. Hence together with $L_2 < B_3$ (also from the choice of x_j) it follows that $x_j > D_3$. Then, using $L_1 \geq B_4$, we get $L_1 + x_j > B_4 + D_3 = B_2$ by Lemma 3.4. Since $L_1 + x_j$ is not in S_2, we also deduce that $L_1 + x_j > T_2$ holds. On the other hand, the actual load L_1 of `M1` is at most T_4. Thus $x_j > T_2 - L_1 \geq T_2 - T_4 = 1$, where the equality comes from Lemma 3.2. Suppose that $L_2 + x_j > T_1$. Then $x_j > T_1 - T_3 = s$ (by the first part of Lemma 3.2) would follow, which would violate the value of OPT, because even the faster machine `M2` can process this job within this makespan. Hence $L_2 + x_j \leq T_1$. Together with the fact that $L_2 + x_j \notin S_1$, we have that $L_2 + x_j < B_1$. At this point x_j is assigned to `M2` by the algorithm.

   Now several subsequent jobs may be assigned to `M2`, while the load of `M2` remains below B_1. But, similarly to the previous steps, there must arrive a further job x_k that would exceed B_1. Indeed, assume that no

such jobs exists. Then $L_1 \leq T_4 = r - 1 < 1$ (by Lemma 2.1) and $L_2 \leq B_1 < s$ (by Lemma 5.2), so the makespan would stay below OPT = 1; a contradiction. Thus the assignment of jobs to `M2` is stopped, and the algorithm goes back to Step 1.

We claim that one of Step 1 or Step 2 will be executed. If Step 1 is not executed, then $L_2 + x_k \notin S_1$ and $L_2 + x_k > B_1$ from the previous loop. Together, $L_2 + x_k > T_1$. Since $L_2 < B_1$, we obtain $x_k > T_1 - L_2 > T_1 - B_1 = D_1$. Then we get $L_1 + x_k > B_4 + D_1 > B_4 + D_3$ by Lemma 5.1, and $B_4 + D_3 = B_2$ by Lemma 3.4, hence $L_1 + x_k > B_2$. Assume that Step 2 is not executed either. Then $L_1 + x_k \notin S_2$. Hence $L_1 + x_k > T_2$. From this is follows that $x_k > T_2 - L_1 \geq T_2 - T_4 = 1$, because $L_1 \leq T_4$ is still true and we have $T_2 - T_4 = 1$ (from Lemma 3.2). Then there are two jobs, say x_k and x_j, which are both bigger than 1, thus both have to be assigned to the fast machine in the optimal schedule. Therefore we have OPT $> \frac{2}{s}$, and $\frac{2}{s} > 1$ (from $2 > s$), which is a contradiction.

5. Finally, suppose that Step 5 is executed. We assign first the actual job to the machine `M2` and then we assign jobs to the machine `M1` until $L_1 + x_i < B_4$. Observe that L_1 cannot remain below B_4. Assume the opposite. Then $L_1 \leq B_4 < B_2 < 1$ by Lemma 5.2. Moreover, $L_2 \leq T_5 < B_1 < s$ by Lemma 2.1. Hence the makespan would be strictly less than OPT = 1; a contradiction. Thus there must arrive a job that ends the loop, i.e., some job x_j with $L_1 + x_j \geq B_4$.

   If any of the four conditions $L_1 + x_j \in S_4$, or $L_1 + x_j \in S_2$, or $L_2 + x_j \in S_3$, or $L_2 + x_j \in S_1$ is satisfied, we go back to Step 1. Note that at this moment $L_1 < B_4 < B_2$ and $L_2 \leq T_5 < B_3$ (applying Lemma 4). Hence it follows that some of Step 1 – Step 4 must be executed, and we are done. Therefore, suppose that none of the four conditions is satisfied. Let us consider the size of x_j.

   Since $L_1 + x_j \geq B_4$ (from the choice of x_j), but $L_1 + x_j$ is not in S_4, we deduce that $L_1 + x_j > T_4$. Hence together with $L_1 < B_4$ (also from the choice of x_j) it follows that $x_j > D_4$. Then $L_2 + x_j > B_5 + D_4 = B_3$, applying $L_2 \geq B_5$ and Lemma 3.4. Since $L_2 + x_j$ is not in S_3, it follows that $L_2 + x_j > T_3$. Together with $L_2 \leq T_5$, we get $x_j > T_3 - T_5$. Then $L_1 + x_j > (B_2 - T_3 + T_5) + (T_3 - T_5) = B_2$, applying $L_1 \geq 0 \geq B_2 - T_3 + T_5$ (Lemma 5.3). On the other hand, we know that $L_1 + x_j$ is not in S_2, thus it follows that $L_1 + x_j > T_2$. Consequently, using Lemma 3.2, we get $y > T_2 - T_4 = 1$.

   We know that $L_2 + x_j$ is not in S_1. Suppose that $L_2 + x_j > T_1$. Then $x_j > T_1 - L_2 > T_1 - T_3 = s$ would follow, applying Lemma 3.1, and $L_2 \leq T_5 < T_3$ by Lemma 4.1; a contradiction. Therefore at this point we assign x_j to machine `M2`, and the increased load of `M2` is strictly bigger than T_3 and strictly smaller than B_1.

   Now several subsequent jobs may be assigned to `M2`, while the load of `M2` remains below B_1. There must arrive a job, say x_k, such that

$L_2 + x_k \geq B_1$. Indeed, assume that it stays below B_1. Since we know that $L_1 \leq T_4$, we conclude similarly to the proof of the previous point that this would lead to a makespan strictly less than $1 = \text{OPT}$; a contradiction.

At this point the algorithm goes back to Step 1. We claim that either Step 1 or Step 2 will be executed. If Step 1 is not executed, then $L_2 + x_k > T_1$, since $L_2 + x_k \geq B_1$. This together with $L_2 < B_1$ implies that $x_k > D_1$. Therefore we get $L_1 + x_k > D_1 > B_2$, by Lemma 5.1. If Step 2 is not executed either, which means that $L_1 + x_k \notin S_2$ and hence $L_1 + x_k > T_2$, then $x_k > T_2 - L_1 \geq T_2 - B_4 > T_2 - T_4 = 1$, where we applied $L_1 < B_4$, $B_4 < T_4$ (by Lemma 4.1), and $T_2 - T_4 = 1$ (by Lemma 3.2).

Summarizing our analysis, we have two jobs, x_j and x_k, both greater than 1, thus both have to be assigned to the fast machine in the optimal schedule. Therefore we have $\text{OPT} > \frac{2}{s}$, and $\frac{2}{s} > 1$ (from $2 > s$), which is a contradiction. Therefore Step 1 or Step 2 has to be executed and we are done.

■

We have seen that Algorithm `FinalCases` solves the problem (does not violate the competitive ratio) if some step of the algorithm is executed. The problem is that it may happen — although only rarely — that no step can be executed because the condition of no step is satisfied. We must take care about these remaining cases. For this we define another algorithm in the next section.

We say that Algorithm `FinalCases` is *executable* if the condition of some step is satisfied. Summarizing our previous investigations, if Algorithm `FinalCases` is executable, then doing so we obtain a schedule which does not violate the competitive ratio.

5 Algorithm `InitialCases`

In order to handle the case where Algorithm `FinalCases` is not executable, we now give the algorithm `InitialCases` that calls `FinalCases` as a subroutine.

For proving that Algorithm `InitialCases` is r-competitive in the considered interval, we still need one more claim as below.

Lemma 7 *Suppose that machine `M1` is empty (i.e., $L_1 = 0$), and that the load L_2 of machine `M2` is at most B_5. If x is a job whose size satisfies $x \notin S_2$ and $L_2 + x \notin S_1$, then $x \leq T_3 - B_5$.*

Proof. Assume that $x > T_3 - B_5$. Since $T_3 - B_5 \geq B_2$ (by Lemma 5.3), it follows that $x > B_2$. Recall that there is no job assigned to `M1` so far. Since $x \notin S_2$, we obtain $x > T_2 \geq B_1$ (where the last estimation was shown in Lemma 5.5). From $x > B_1$ it then follows that $L_2 + x > L_2 + B_1 \geq B_1$. Together with $L_2 + x \notin S_1$, we deduce that $L_2 + x > T_1$. Hence $x > T_1 - L_2 \geq T_1 - B_5 > T_1 - T_3 = s = s \cdot \text{OPT}$ (by Lemma 4.2 and Lemma 3.2). This is a contradiction, since no job can be bigger than $s \cdot \text{OPT}$. ■

Algorithm 2: `InitialCases`

1 $L_1 := 0, L_2 := 0$ `// both machines are empty`
$i := 1$ `// start with first job`
while $L_2 + x_i < B_5$ **do**
  $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
  $i := i + 1$ `// next job`
call Algorithm `FinalCases`

2 $L_2 := L_2 + x_i$ `// assign job` x_i `to M2`
$j := i + 1$ `// next job`
while $L_2 + x_j < B_3$ **do**
  $L_2 := L_2 + x_j$ `// assign job` x_j `to M2`
  $j := j + 1$ `// next job`
call Algorithm `FinalCases`

3 $L_1 := L_1 + x_j$ `// assign job` x_j `to M1`
$k := j + 1$ `// next job`
while $L_2 + x_k < B_3$ **do**
  $L_2 := L_2 + x_k$ `// assign job` x_k `to M2`
  $k := k + 1$ `// next job`
call Algorithm `FinalCases`

4 $L_2 := L_2 + x_k$ `// assign job` x_k `to M2`
$\ell := k + 1$ `// next job`
while $L_2 + x_\ell < B_1$ **do**
  $L_2 := L_2 + x_\ell$ `// assign job` x_ℓ `to M2`
  $\ell := \ell + 1$ `// next job`
call Algorithm `FinalCases`

After this, we state the next theorem.

Theorem 8 *Algorithm* `InitialCases` *is r-competitive for any* $q_6 \leq s \leq \sqrt{3}$.

Proof.

1. If Algorithm `FinalCases` is called in Step 1 and there all jobs are assigned to machines (in Step 1 and Step 2 of Algorithm `FinalCases`), then `FinalCases` terminates and all jobs are within the safe sets, so the competitive ratio of r is not exceeded.

   At the end of Step 1, let us denote the actual job by x_i. It holds that $L_2 + x_i \geq B_5$, and before x_i came, L_2 was below B_5. Algorithm `FinalCases` was called at the end of Step 1, but none of the conditions of the five Steps 1–5 in Algorithm `FinalCases` was actually true (i.e., `FinalCases` was not executable). In particular, Step 5 of `FinalCases` was not executed. Since $L_1 = 0$ (machine `M1` is empty), and $B_4 > 0$ (from Lemma 4.1), it thus follows that $L_2 + x_i > T_5$. Together with $L_2 < B_5$ it follows from Lemma 3 that $x_i > T_5 - B_5 = B_4.6$. Note that at this point still there is no job assigned to `M1`. Since x_i is not assigned to `M1` (as `FinalCases` was not executable), in particular, Step 4 of `FinalCases` is not executable. Since $L_2 < B_5 < B_3$ (from Lemma 4.2), it means that $L_1 + x_i \notin S_4$. From $x_i > B_4$ (see above) it then follows that $x_i > T_4$.

   Suppose that $x_i > B_3$ holds (from which we derive a contradiction). Then it follows that $x_i > T_3 - B_5$, because otherwise, if $x_i \leq T_3 - B_5$, then $T_3 \geq x_i + B_5 > x_i + L_2 > x_i > B_3$, hence $L_2 + x_i \in S_3$. Since $L_1 = 0 < B_2$ (from Lemma 4.1), it follows that Step 3 of Algorithm `FinalCases` would be executed; a contradiction. Hence $x_i > T_3 - B_5$. Note that all assumptions of Lemma 7 are satisfied. Hence $x_1 \leq T_3 - B_5$; a contradiction. Therefore $x_i \leq B_3$.

   Thus we conclude from the previous two paragraphs that $T_4 < x_i < B_3$.

   Let us investigate how big the actual load of `M2` would be, if x_i was assigned to this machine; that is, we want to estimate $L_2 + x_i$. We are going to show that $T_5 < L_2 + x_i < B_3$, by excluding all other possibilities. To prove the lower bound, note that since the algorithm terminated the while-loop, we have $L_2 < B_5$ and $L_2 + x_i \geq B_5$. As we argued above, we know that $L_2 + x_i \notin S_5$, hence we have $L_2 + x_i > T_5$. To prove the upper bound, we need to exclude two more cases (see also Figure 2).

   (a) Suppose that $L_2 + x_i \in S_3 = [B_3, T_3]$. Since $L_1 = 0 \leq B_2$ (by Lemma 4.1), Step 3 of Algorithm `FinalCases` would have been executed; a contradiction. Thus $L_2 + x_i \notin [B_3, T_3]$.

   (b) Suppose that $L_2 + x_i > T_3$. Then $x_i > T_3 - L_2 > T_3 - B_5$ (since $L_2 < B_5$, see above). Note that all assumptions of Lemma 7 are satisfied. Hence $x_1 \leq T_3 - B_5$; a contradiction. Thus $L_2 + x_i \leq T_3$.

Consequently, $T_5 < L_2 + x_i < B_3$.

2. We enter Step 2. We assign x_i to `M2`. From the analysis above we know that the load L_2 after this assignment is above T_5 and below B_3.

   Then several jobs may come, and they are assigned to machine `M2`, while the load L_2 of `M2` remains below B_3. This termination point of the while-loop will come for sure: otherwise we would have an empty machine `M1`, and the total load of all jobs, all on machine `M2`, would be still below B_3. Since $B_3 < B_1 < s = s \cdot \mathrm{OPT}$ (by Lemma 5.2 and Lemma 4.2), this contradicts the assumption that the optimum value is OPT.

   Let x_j denote the job upon terminating the while-loop. Now we call Algorithm `FinalCases` with this index j. Assume `FinalCases` is not executable (otherwise we are done). It holds that $L_2 < B_3$ and $L_2 + x_j \geq B_3$. Furthermore, $L_1 = 0 \leq B_2$ (by Lemma 4.1), but Step 3 of Algorithm `FinalCases` was not executed, thus $L_2 + x_j \notin S_3$. Consequently, $L_2 + x_j > T_3$, and thus $x_j > D_3$. By Lemma 5.6 we have $D_3 > B_4$. Since no job is assigned to `M1`, and Step 4 of `FinalCases` was not executable, moreover $L_2 \leq B_3$, we have that $L_1 + x_j = x_j \notin S_4$. From $x_j > D_3 > B_4$ we deduce $x_j > T_4$.

   The assumption of $x_j \geq B_2$ will lead to a contradiction as follow. Since Step 2 of `FinalCases` was not executable, it holds that $L_1 + x_j = x_j \notin S_2$, hence $x_j > T_2$. In Lemma 5.5 we proved that $T_2 > B_1$, hence $x_j > B_1$. Since also Step 1 of `FinalCases` was not executable, it holds that $L_2 + x_j \notin S_1$. From $x_j > B_1$ we thus deduce that $L_2 + x_j > T_1$. Thus we estimate $x_j > T_1 - L_2 > T_1 - T_3 = s$, where the second estimation uses $L_2 < B_3 < T_3$ and the last inequality is due to Lemma 3.2. Hence $x_j > s = s \cdot \mathrm{OPT}$, so job x_j would be too large for an optimum value of OPT.

   Summing up, we conclude that $T_4 < x_j < B_2$ holds.

3. In Step 3 we assign x_j to `M1`, and since this is the only job which has been assigned to M_1 ever, the load L_1 of `M1` is between T_4 and B_2.

   Then again, several jobs may come, and they are assigned to machine `M2`, while the load L_2 of `M2` remains below B_3. This termination point of the while-loop will come for sure: otherwise we would have machine `M1` with a load lower than $B_2 < 1 = \mathrm{OPT}$ by Lemma 5.2, and the load of L_2 is below $B_3 < B_1 < s = s \cdot \mathrm{OPT}$ by Lemma 5.2. This contradicts the assumption that the optimum value is OPT.

   Let x_k denote the job upon terminating the while-loop. Now we call Algorithm `FinalCases` with this index k. Assume that `FinalCases` is not executable (otherwise we are done). In particular, Step 3 of `FinalCases` was not executable, and since $L_1 \leq B_2$, it follows that $L_2 + x_k \notin S_3$. Taking into account that $L_2 < B_3$ and $L_2 + x_k \geq B_3$, it follows that $L_2 + x_k > T_3$, hence $x_k > D_3$.

From Lemma 5.7 it follows that $x_j + x_k - B_2 > T_4 + D_3 - B_2 > 0$, thus $x_j + x_k > B_2$. Since Step 2 of `FinalCases` was not executable, it means that $L_1 + x_k = x_j + x_k \notin S_2$, hence $x_j + x_k > T_2$. Since $L_1 = x_j \leq B_2$, we have $x_k > D_2 = D_1$ (by the definition of D_1 and D_2).

Assume that $L_2 + x_k \geq B_1$. Since Step 1 of `FinalCases` was not executed, it would follow that $L_2 + x_k \geq T_1$. Thus taking into account that $L_2 < B_3$, we obtain the estimation $x_k \geq T_1 - L_2 > T_1 - B_3 > T_1 - T_3 = s = s \cdot \mathrm{OPT}$ (by Lemma 4.2 and Lemma 3.2), which contradicts the optimality of value OPT. Thus $L_2 + x_k < B_1$.

4. We start Step 4 with assigning x_k to `M2`. Then the new load L_2 is between T_3 and B_1.

   Then for the last time, several jobs may come, and they are assigned to machine `M2`, as long as the load L_2 of `M2` remains below B_1. The termination point of the while-loop will come for sure: otherwise we would have a machine `M1` with a load lower than $B_2 < 1 = \mathrm{OPT}$ (by Lemma 5.2), and the load of L_2 is below $B_1 < s = s \cdot \mathrm{OPT}$ by Lemma 5.8. This contradicts the assumption that the optimum value is OPT.

   Let x_ℓ denote the job upon terminating the while-loop. We will show that now `FinalCases` *is* executable, thus we are done. Assume the opposite: `FinalCases` is not executable.

   At this point we have $L_2 < B_1$ and $L_2 + x_\ell \geq B_1$. Since `FinalCases` is not executable, in particular, Step 1 of `FinalCases` was not executable, meaning $L_2 + x_\ell \notin S_1$. Hence $L_2 + x_\ell > T_1$, thus $x_\ell > D_1$.

   Using $x_j > T_4$ from Step 2 above, we can estimate $x_j + x_\ell > T_4 + D_1 > D_1 > B_2$ using Lemmas 4.1 and 3.1. Since at this point only x_j is assigned to `M1`, and Step 2 of `FinalCases` is not executable, that is $L_1 + x_\ell = x_j + x_\ell \notin S_2$, it also holds that $x_j + x_\ell > T_2$.

   We summarize: $x_i, x_j > T_4$, $x_k, x_\ell > D_2 = D_1$, moreover $x_j + x_k > T_2$ and $x_j + x_\ell > T_2$.

   Note that $x_k + x_\ell > 2D_1 > s = s \cdot \mathrm{OPT}$ (by Lemma 5.8). So it follows that x_k and x_ℓ must be assigned to different machines in any optimum schedule, because even the faster machine `M2` cannot handle both jobs within a makespan of OPT.

   First, consider an optimum schedule where x_k is assigned to the slower machine `M1` and x_ℓ is assigned to the faster machine `M2`. Assume that x_i is also assigned to `M1`. Then we can estimate the load of this machine: $L_1 \geq x_k + x_i > D_1 + T_4 > 1 = \mathrm{OPT}$ (by Lemma 5.9); a contradiction. Hence x_i cannot be assigned to `M1`. Similarly, if we assume that x_j is assigned to `M1`, we can deduce the very same estimation. Hence also x_j cannot be assigned to `M1`. So both x_i and x_j must be assigned to the faster machine `M2`.

   Second, consider an optimum schedule where x_ℓ is assigned to the slower machine. Then by repeating the same arguments as above, we can

deduce that also in this case, both x_i and x_j must be assigned to the faster machine `M2`.

Thus in any optimal schedule, both x_i and x_j are assigned to the fast machine `M2`, and one of x_k and x_ℓ is also assigned to the fast machine. Thus by Lemma 5.10 we get $s \cdot \mathrm{OPT} \geq \min\{x_i + x_j + x_k, x_i + x_j + x_\ell)\} = x_i + \min\{x_j + x_k, x_j + x_\ell\} > T_4 + \min\{T_2, T_2\} = T_4 + T_2 \geq s = s \cdot \mathrm{OPT}$; a contradiction.

It follows that our assumption was false, i.e., when job x_ℓ is revealed, `FinalCases` is executable. This completes the proof.

■

6 Computational Experiments

We implemented the algorithm `InitialCases` in Python 3.4 and ran computational experiments on a test set of randomly generated instances with different distributions of the job lengths. It is known that the offline version of the two-machine scheduling problem is NP-hard, see [9]. To compute the optimal makespan numerically, we formulate the offline scheduling problem as a mixed integer linear programming problem (see below), and then solve it with a linear programming based branch-and-cut approach using the numerical solver IBM ILOG Cplex 12.6.1. All experiments were conducted on a MacBookPro with MacOSX 10.10, a 2.8 GHz Intel Core i7 CPU and 16 GB 1600 MHz DDR3 RAM.

The mixed-integer model is stated as follows. Let J be an index set for the jobs in an instance. Each job $j \in J$ comes with a length $\ell_j \geq 0$. The speed of machine `M2` is specified by the parameter $s \geq 1$. For each machine $i \in I := \{\mathtt{M1}, \mathtt{M2}\}$ and each job $j \in J$ we introduce a binary decision variable $x_{i,j} \in \{0, 1\}$, where $x_{i,j} = 1$ if and only if job j is assigned to machine i. The real-valued variables $L_i \in \mathbb{R}_+$ for each machine $i \in I$ represent the load on this machine, i.e., the sum of all assigned jobs. Finally, variable $M \in \mathbb{R}_+$ denotes the makespan, i.e., the completion time of the latest job on either machine.

The scheduling problem is then algebraically described with the following constraints. Each job has to be assigned to exactly one of the two machines:

$$\sum_{i \in I} x_{i,j} = 1, \quad \forall\, j \in J. \tag{1}$$

The length of all jobs is then computed as

$$L_i = \sum_{j \in J} \ell_j x_{i,j}, \quad \forall\, i \in I. \tag{2}$$

The total makespan M is then the maximum of the makespans on each machine, which is derived from the individual loads $L_{\mathtt{M1}}, L_{\mathtt{M2}}$ and processing speeds $1, s$,

respectively:

$$L_{\texttt{M1}} \leq M, \tag{3}$$

$$\frac{1}{s} L_{\texttt{M2}} \leq M. \tag{4}$$

The objective is to minimize the total makespan:

$$\min \ M. \tag{5}$$

In a first computation experiment, we demonstrate how big an instance can be, in terms of the number of jobs, to allow the computation of the proven global optimum within a reasonable time. We generate a sequence of instances of growing sizes $|J| = 1, 2, 3, \ldots, 25$. For each instance size, we generate 100 instances, where each job length is chosen from the set $\{1, 2, \ldots, 5 \cdot 10|J|\}$ with a uniform distribution. We further vary the speed s of machine M2. To this end, we discretize the interval $[1, 2]$ into 25 steps. In total, we thus solve $25 \cdot 100 \cdot 25 = 65,000$ instances. The resulting CPU time is marked as a dot in Figures 3 and 4. The geometric and arithmetic mean is marked as a red or blue line, respectively.

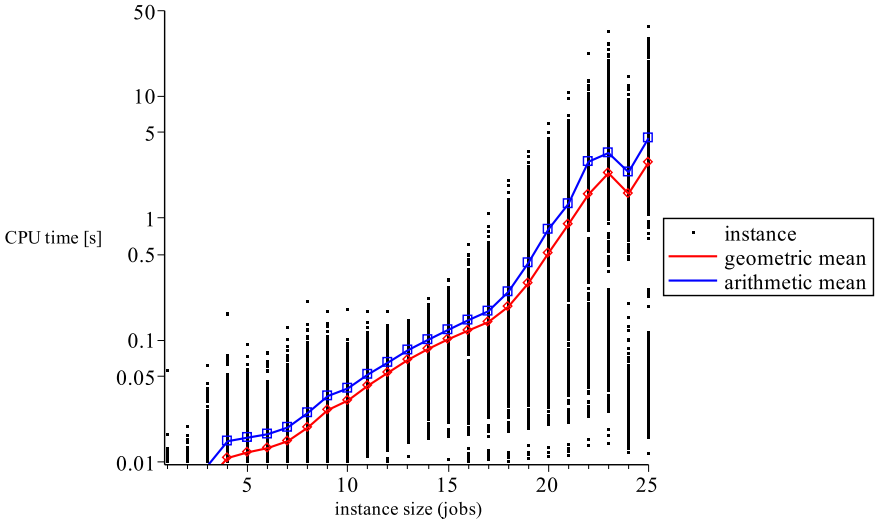

Figure 3: CPU times for varying instance sizes (number of jobs) $|J|$.

In Figure 3 we analyze the dependency of the CPU time on the instance size (number of jobs in J), independent of s. The solving time varies much; for any value of $|J|$ there are instances that can be solved almost immediately, but on the other side of the range, there was a single outlier (outside the range of the plot) that caused a running time of more than 1,500 seconds. As one can see, instances with up to 25 jobs can be solved within less than 5 second in average, but the general growth of the average solving time is (empirically) exponential

in $|J|$. More than 30 jobs cannot be handled with this approach anymore in reasonable time. It is worth noting that also for much larger instances the numerical solver is able to find a feasible solution to the problem that is often very close to optimality (with a gap of less than 0.01% between lower and upper bound), but then it fails to close this tiny gap even after a long time of branching. Summing up, we decided to limit the instance size to at most 25 jobs for the following experiments.

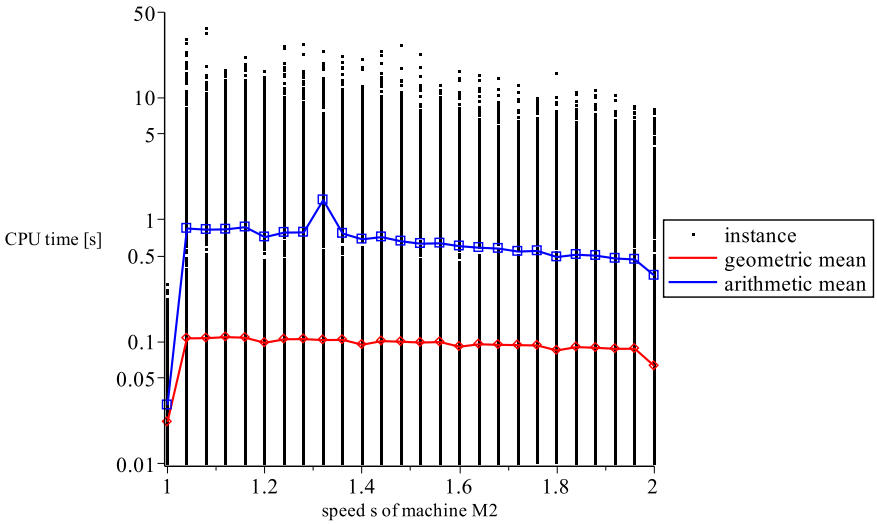

Figure 4: CPU times for different values of s.

A further evaluation of the same runs is shown in Figure 4, where we give the CPU time depending on the speed s, independent of n. Again, there is a large range of CPU times for each value of s, mainly because the instance size varies between 1 and 25 jobs. Interestingly, it is worth noting that for $s = 1$ the solving time is very low for any instance (less than 0.5 seconds even in the worst case). The reason might be that the scheduling problem for two identical processors is just a two-knapsack problem, that can be handled much easier with the existing branch-and-cut methods inside the solver. As soon as $s > 1$, this structure is no longer available, and much higher solving times can be expected. Also note that for larger values of s the average time (slightly) reduces, which can be best seen in the arithmetic mean values (blue line in Figure 4).

Having the offline makespan value M at hand, we divide each job length of this instance by M, so that the optimum value becomes 1. We implemented algorithm `InitialCases` (and thus `FinalCases` as a subroutine), and then scheduled the instances in an online way. This works of course very fast, so we do not need to worry about CPU times for this part. Hence, we are now more concerned with the resulting quality of the schedule, that is, we compare the semi-online makespan with the corresponding offline makespan (which was 1),

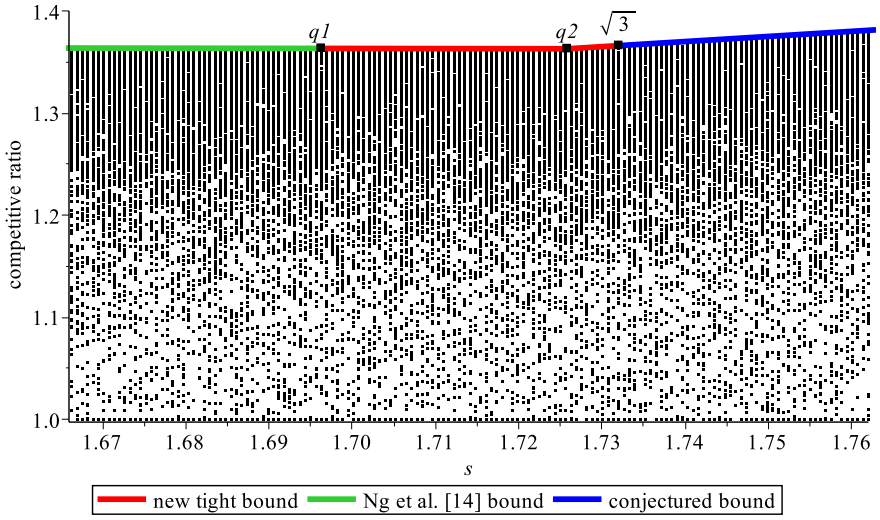

Figure 5: Comparing the competitive ratios from `InitialCases` (applied to random instances) with the bounds.

and we want to see how far away it actually is from the theoretical bound. For the interval $[q_6, \sqrt{3}]$, where we know that our algorithm has an optimal competitive ratio, the results can be found in Figure 5. Each black dot refers to the competitive ratio, when applied to a randomly generated instance with 5 to 25 jobs. The horizontal s-axis was split into 151 pieces, and for each value of s we generated 200 instances, so that in total $30,200$ instances were solved. As we can see, the actual ratios are quite near to the theoretically predicted bounds. For smaller instances, the ratio tends to be closer to 1, but for larger instances (more jobs), the ratio comes closer to the bound.

We can also analyze, which part of the two algorithms, `InitialCases` and `FinalCases`, was actually called how many times. In $30,200$ calls of `InitialCases`, already $29,057$ (96.21%) were finished in Step 1 of that algorithm. Then, another $1,138$ (3.77%) were finished in Step 2, and the remaining 5 (0.02%) in Step 3. The algorithm did not enter Step 4 at all. For `FinalCases`, we have 183 executions of Step 1, $30,017$ for Step 2 (which adds up to $30,200$ for Step 1 and Step 2 together), $10,439$ for Step 3, $21,021$ for Step 4, and $1,468$ for Step 5.

Of course, the algorithm `InitialCases` is applicable outside the interval $[q_0, \sqrt{3}]$, too. Figure 5 shows some instances with $s > \sqrt{3}$, and the bound r_5 seems to be an upper bound also for these values of s. Similarly (but not shown in this figure), we found that r_5 seems to be a valid upper bound for instances with $s < q_0$. From this we conjecture that our algorithm has the same bound r_5 even outside $[q_0, \sqrt{3}]$. Since it would not be an optimal competitive ratio, we refrain from giving a formal proof for this.

7 Conclusions

We gave a compound algorithm and showed that its competitive ratio equals the previously known lower bound for any speed $s \in [\frac{5+\sqrt{241}}{12}, \sqrt{3}] \approx [1.7103, 1.7321]$, i.e. on the "wide" interval. Although the considered interval is in fact "not too wide", we applied new ideas, to be able to get the tight ratio here.

Our new idea in the algorithm design is as follows. Instead of having a universal algorithm, we have two algorithms: one for the "good cases" and another for the problematic cases. If the incoming job is *good* in some sense for us, we assign it with the first algorithm. Otherwise, if the incoming job is *bad*, we assign it by the second algorithm. (Of course, we make only one common schedule, the next job is assigned by the rule of either the first, or the second algorithm, but not both.) The good or bad status of the incoming job depends on its size, and on the actual values of the loads of the machines as well.

If, at any time, a good job arrives, we win against the adversary list, as we are able to finish the schedule by the first algorithm, without violating the prescribed competitive ratio. And it turns out that in any sequence there *must* come a good job.

Except for the narrow interval (which is approximately $\big[1.6934, 1.6963\big]$) where the gap between the upper and lower bounds is very small, the question about the tight value of the competitive ratio for our problem remains open for speeds between $\frac{\sqrt{73}+3}{8} \approx 1.443$ and $\frac{5}{3}$. We think that our new idea can be helpful to get the tight ratio (or a ratio which is close to the tight ratio), where the question is actually open.

Moreover, we conjecture that the competitive ratio of our compound algorithm is $r_2(s)$ for all $s \in [1, q_7]$ and $r_5(s)$ for all $s \in [q_7, \infty)$. This guess is supported by our computational experiments.

Acknowledgements. Krzysztof Węsek's work was conducted as a guest researcher at the Helmut Schmidt University.

References

[1] S. Albers. Better bounds for online scheduling. *SIAM Journal of Computing*, 29:459 – 473, 1999.

[2] E. Angelelli, Á. B. Nagy, M. G. Speranza, and Zs. Tuza. The On-line Multiprocessor Scheduling Problem with Known Sum of the Tasks. *Journal of Scheduling*, 7:421 – 428, 2004.

[3] E. Angelelli, M. G. Speranza, J. Szoldatics, and Zs. Tuza. Geometric representation for semi on-line scheduling on uniform processors. *Optimization Methods & Software*, 25:421 – 428, 2010.

[4] E. Angelelli, M. G. Speranza, and Zs. Tuza. Semi on-line scheduling on three processors with known sum of the tasks. *Journal of Scheduling*, 10:263 – 269, 2007.

[5] E. Angelelli, M. G. Speranza, and Zs. Tuza. Semi-online scheduling on two uniform processors. *Theoretical Computer Science*, 393:211 – 219, 2008.

[6] Y. Azar and O. Regev. On-line bin-stretching. *Theoretical Computer Science*, 268(1):17 – 41, 2001.

[7] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35:108 – 121, 2000.

[8] T. C. E. Cheng, H. Kellerer, and V. Kotov. Semi-on-line multi-processor scheduling with given total processing time. *Theoretical Computer Science*, 337:134 – 146, 2005.

[9] E. G. Coffman, M. R. Garey, and D. S. Johnson. An Application of Bin-Packing to Multiprocessor Scheduling. *SIAM Journal of Computing*, 7(1):1 – 17, 1978.

[10] Gy. Dósa, A. Fügenschuh, Z. Tan, Zs. Tuza, and K. Węsek. Semi-Online Scheduling on Two Uniform Machines with Known Optimum Part I: Tight Lower Bounds. Technical report, Applied Mathematics and Optimization Series AMOS#27, Helmut Schmidt University/University of the Federal Armed Forces, Hamburg, Germany, 2015.

[11] Gy. Dósa, M. G. Speranza, and Zs. Tuza. Two uniform machines with nearly equal speeds: unified approach to known sum and known optimum in semi on-line scheduling. *Journal of Combinatorial Optimization*, 21:458 – 480, 2011.

[12] T. Ebenlendr and J. Sgall. A lower bound on deterministic online algorithms for scheduling onrelated machines without preemption. In *Proceeding of the 9th Workshop on Approximationand Online Algorithms*, Lecture Notes in Computer Science, pages 102 – 108, 2007.

[13] L. Epstein. Bin stretching revisited. *Acta Informatica*, 39:97 – 117, 2003.

[14] U. Faigle, W. Kern, and Gy. Turán. On the performance of on-line algorithm for particular problem. *Acta cybernetica*, 9:107 – 119, 1989.

[15] R. Fleischer and M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343 – 353, 2000.

[16] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceeding of the 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 564 – 565. ACM, New York/Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[17] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563 – 1581, 1966.

[18] H. Kellerer, V. Kotov, M. G. Speranza, and Zs. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21:235 – 242, 1997.

[19] K. Lee and K. Lim. Semi-online scheduling problems on a small number of machines. *Journal of Scheduling*, 16:461 – 477, 2013.

[20] C. T. Ng, Z. Tan, Y. He, and T. C. E. Cheng. Two semi-online scheduling problems on two uniform machines. *Theoretical Computer Science*, 410(8 – 10):776 – 792, 2009.

[21] Z. Tan and A. Zhang. Online and Semi-online Scheduling. In P. M. Pardalos et al., editor, *Handbook of Combinatorial Optimization*, pages 2191 – 2252. Springer Science+Business Media New York, 2013.