General parametric scheme for the online uniform machine scheduling problem with two different speeds

Alexandre Dolgui, Vladimir Kotov, Aliaksandr Nekrashevich, Alain Quilliot

General parametric scheme for the online uniform machine scheduling problem with two different speeds

Alexandre Dolgui[a,*], Vladimir Kotov[d,c], Aliaksandr Nekrashevich[c], Alain Quilliot[b]

[a] *IMT Atlantique, LS2N - UMR CNRS 6004, La Chantrerie, 4, rue Alfred Kastler - B.P. 20722, F-44307 NANTES cedex 3, France*
[b] *ISIMA, LIMOS - UMR CNRS 6158, Complexe scientifique des Cézeaux, 63173 AUBIERE cedex, France*
[c] *Belarusian State University, FPMI DMA department, 4 Nezavisimosti avenue, 220030, MINSK, Belarus*
[d] *Vilnius Gediminas Technical University, Faculty of Fundamental Sciences, Department of Information Technologies, Saulėtekio al. 11, LT-10223 VILNIUS, Lithuania*

Abstract

In this paper, we consider the online uniform machine scheduling problem on m processors when speed $s_i = 1$ for $i = k+1, ..., m$ and $s_i = s$, $s > 1$, for $i = 1, ..., k$. The objective is to minimize makespan. We propose a parametric scheme with the worst-case performance 2.618 when $1 < s \leq 2$, and with the asymptotic worst-case performance $\frac{1}{2}(1+s+\sqrt{5-2s+s^2})$ for all $s > 1$ when the ratio m/k tends to infinity.

Keywords: approximation algorithms, online algorithms, scheduling

1. Introduction and Problem Formulation

We study a classic online uniform machine scheduling problem for m uniform processors $(M_1, \ldots, M_m)$ with speeds $(s_1, ..., s_m)$ without preemption. The objective is to minimize makespan. Jobs come sequentially one after another, and are immediately assigned to one of the processors. Jobs cannot change their processors afterward. No information about the future jobs is given. On the other hand, the order of the jobs is not connected to their starting time in the schedule. For example, a job that arrives later may start earlier than the current one. We only fix the assignment to the processors. Let us identify a job j with its processing time p_j. If job j is scheduled for processor M_i, then job processing takes $\frac{p_j}{s_i}$ time.

An algorithm's quality is measured by its worst-case performance. Denote by $F(L^A)$ the makespan of the schedule created by algorithm A, and denote by $F(L^{\text{opt}})$ the optimal offline makespan. We say that an algorithm A has the worst-case performance $c \in \mathbb{R}$ if, for any list of jobs L, $F(L^A) \leq c \cdot F(L^{\text{opt}})$.

The online scheduling problem for identical processors without preemption was first investigated by Graham (1969). He showed that the List Scheduling algorithm (LS), assigning the current next job to the least loaded processor, has the worst-case performance $(2 - \frac{1}{m})$.

Cho and Sahni (1980) proved the following worst-case performance of the LS for the system of uniform processors. For $m = 2$, the ratio is at most $\frac{1+\sqrt{5}}{2}$. If $m > 2$, the ratio is at most $1 + \frac{\sqrt{2m-2}}{2}$. In the special case when $s_1 = \ldots = s_{m-1} = 1$ and $s_m > 1$, they proved the worst-case performance of $3 - \frac{4}{m+1}$, for $m > 2$.

Li and Shi (1998) proved that the algorithm LS has the least possible worst-case performance for $m \leq 3$, and proposed an approximation algorithm for the particular case $s_i = 1, i = 1, \ldots, m-1$, and $s_m = 2, m \geq 4$. Their algorithm has the worst-case asymptotic performance of 2.8795 for $m \to \infty$. For the case $s_i = 1, i = 1, \ldots, m-1$, and $1 < s_m \leq 2, m \geq 4$, Cheng et al. (2006) designed an algorithm with the worst-case performance of 2.45. The case when the number of processors is two was studied in the work of Liu et al. (2009); Angelelli et al. (2008); Wen and Du (1998).

Related models. Some related models have been investigated in the past few years and semi-online algorithms have been studied. This term encompasses algorithms that are essentially online, but some partial information about the input is given to the scheduler in advance. The main motivation behind this approach is the observation that the classic competitive analysis is too pessimistic compared to practical results; in other words, the adversary who may arbitrarily determine the input sequence is too powerful. In practice, the inputs are not completely arbitrary, and it may be reasonable to restrict them.

Semi-online algorithms for scheduling with reassignment to two identical machines were considered in Min et al. (2011). Online algorithms with rearrangements for two related machines were studied in Wang et al. (2012); Cao and Liu (2010); Dosa et al. (2011). There are some related publications studying scheduling subject to eligibility constraints. Online scheduling on two uniform machines subject to eligibility constraints of type GradeOfService (GoS) was first considered in Liu et al. (2009). More recent results for this problem were presented in Lee et al. (2009).

Our contribution. In this paper, we consider the online uniform machine scheduling problem for the case $s_i = 1$ for $i = k+1, ..., m$ and $s_i = s$, $s > 1$, for $i = 1, ..., k$. The objective is to minimize makespan. We present a generalization of the approach proposed by Dolgui et al. (2015). Our parametric scheme has a better worst-case performance than the current known results starting from a big enough m with fixed k and s. We also prove that for the case $1 < s \leq 2$, the worst-case performance of our algorithm is at most of 2.618. The proposed parametric scheme requires only the ratio of the number of all the processors to the number of the fast ones instead of the exact numbers. For the case in which this ratio tends to infinity and the fast processors have the speed s, we obtain the asymptotic worst-case

*Corresponding author
Email addresses: alexandre.dolgui@imt-atlantique.fr (Alexandre Dolgui), kotovvm@yandex.by (Vladimir Kotov), nekrald@gmail.com (Aliaksandr Nekrashevich), alain.quilliot@isima.fr (Alain Quilliot)

performance of $\frac{1}{2}(1+s+\sqrt{5-2s+s^2})$. We also propose some algorithms that would help to find parameters for the proposed scheme. In our proofs and statements, we use the ideas of reserved classes and the dynamic lower bound of the optimal solution from Dolgui et al. (2015); Kellerer and Kotov (2013); Kellerer et al. (2015).

2. Notation and Lower Bounds

Denote the number of processors by m. Let us enumerate the processors with numbers from 1 to m. Denote the i-th processor by M_i. Denote by k the number of the processors with a speed $s > 1$. Let $L_{i,j}$ be the current load for the processor with number i before scheduling job j (with processing time p_j at step j). Denote by OM_j the optimal makespan after the first j jobs.

Denote by $q_1^{(j)}, \dots, q_j^{(j)}$ the processing times of jobs $1, \dots, j$ ordered at step j such that $q_1^{(j)} \geq q_2^{(j)} \geq \dots \geq q_j^{(j)}$. For simplicity, we assume that $q_y^{(j)}$ with non-positive indices y are equal to zero. Denote by z the upper integer part of s, i.e. z is a positive integer s.t. $(z-1) < s \leq z$.

Set $V_j^{(1)} = \min\left\{q_{(z-1)\cdot k+1}^{(j)}, \frac{q_{(z-1)\cdot(k-1)+1}^{(j)}+\dots+q_{(z-1)\cdot k+1}^{(j)}}{s}\right\}$, $V_j^{(2)} = \frac{p_1+p_2+\dots+p_j}{m-k+s\cdot k}$, $V_j^{(3)} = \frac{q_1^{(j)}}{s}$.

Property 1. *The optimal makespan after the first j jobs $OM_j \geq V_j^{(1)}$, $OM_j \geq V_j^{(2)}$, and $OM_j \geq V_j^{(3)}$.*

Proof. Values $V_j^{(2)}$ and $V_j^{(3)}$ are trivial lower bounds. Let us prove $OM_j \geq V_j^{(1)}$. Consider the $k(z-1)+1$ greatest jobs at step j and denote them by GJ_j. WLOG, for $j < k(z-1)+1$, it is possible to add some jobs with zero size. Then, in any scheduling, there are two cases. In the first, some job from GJ_j comes to a processor with speed one. It follows that the makespan of such scheduling is at least $q_{(z-1)\cdot k+1}^{(j)}$. In the second case, all of jobs come to processors with speed s. Since there are $k(z-1)+1$ jobs in GJ_j, there is a processor IM_j with speed s, such that at least z jobs are scheduled to it. Thus, the current load of IM_j is at least $\frac{q_{(z-1)\cdot(k-1)+1}^{(j)}+\dots+q_{(z-1)\cdot k+1}^{(j)}}{s}$. □

Denote by LB_j the maximal value among $V_j^{(1)}, V_j^{(2)}, V_j^{(3)}$. It is clear that LB_j is a lower bound for the optimal makespan OM_j. Trivial calculations show $LB_{j-1} \leq LB_j$.

3. Algorithm

Let us construct a parametric algorithm with the worst-case performance $B \geq s$. Let m_1, m_2, and R be non-negative integers, such that $k + m_1 + m_2 = m$, and $m_2 = R \cdot (z-1) \cdot k$.

Divide all processors into three classes. All k processors with speed s are assigned to the class **Fast**, m_1 processors are assigned to the class **Normal**, and the remaining m_2 processors are assigned to the class **Reserved**. By definition, m_2 is an integer multiple of $k \cdot (z-1)$. Note that the processors with high speed are always assigned to the class Fast. The classes Normal and Reserved are not fixed in advance; we only fix their cardinalities. In other words, processors from these classes may change their classes during the execution of the algorithm.

For the purpose of the algorithm description, we divide the list of Reserved processors into the groups $G_1, \dots, G_R$. Every group is an ordered list of $(z-1)\cdot k$ processors. The processors are numbered from 1 to $(z-1)\cdot k$ within the group.

Introduce a real number $\varphi \in [0,1]$ and the following classification for the current job at the stage j. The job j with a processing time p_j not exceeding $\varphi \cdot B \cdot LB_j$ is called **small**. Otherwise, the job is called **big**. We say that job j fits into processor i, if $L_{i,j} + p_j/s_i \leq B \cdot LB_j$.

Algorithm. Initialize $u = 1$. Then perform the following points sequentially for each new job j:

1. If job j fits into one of the processors of the Fast or Normal classes, assign it there and go to the next job.
2. If $u \leq (z-1)\cdot k$, carry out the following instructions: Assign the current job to the processor with number u in G_1; denote this processor by g_u. Denote by i the processor from the Normal class with the minimal current load. Swap processors g_u and i, i.e. processor i substitutes processor g_u in group G_1 (and obtains number u within the group). The processor i moves to the Reserved class. Correspondingly, g_u moves to the Normal class. After that increment $u := u + 1$.
3. If $u > (z-1)\cdot k$, move the group numeration $G_1, \dots, G_R$ cyclically left by 1, i.e. the group G_r becomes G_{r-1} if $r > 1$, and group G_1 becomes G_R. Afterward, set $u := 1$.

Notice that at point 3 we restore the property $u \leq (z-1)\cdot k$ if it was violated before. The algorithm can be implemented with complexity $O((j+m)\cdot\log(m+j))$ using some standard data structures, where j is the number of currently processed jobs.

The main idea of the algorithm is to determine a good proportion of m_1 and m_2 in order to guarantee the following conditions:

- A small job will fit into a processor from the Fast class or into a processor from the Normal class.
- If a big job does not fit into any of the processors from the Fast and Normal classes, it will fit into one of the processors from the Reserved class.

Moreover, we also want to minimize B. For this purpose, we introduce the following parametric scheme with parameters m_1, m_2, R, B, and φ. The first inequality of the scheme will force the first condition to hold. The second inequality of the scheme will force the second condition to hold.

3.1. Parametric scheme

As mentioned before, the algorithm is parametric in the terms of m_1, m_2, R, B, and φ, where m_1, m_2, R, and z are non-negative integers, $k + m_1 + m_2 = m$, z is the upper integer part of s, $s+1 > z \geq s$, and B, φ are non-negative real numbers, $B \geq s$, $\varphi \in [0,1]$, $m_2 = R \cdot (z-1) \cdot k$. Note that since $m_2 = R \cdot (z-1) \cdot k$, and $m_2 \leq m - k$, then R cannot be arbitrarily large.

We further consider only those parameters satisfying the following system:

$$\begin{cases} (B-1)\cdot s\cdot k + (1-\varphi)\cdot m_1 \cdot B \geq s\cdot k + m_1 + m_2 \\ (1-\varphi)\cdot B \leq (\varphi\cdot B)^R \cdot (B-s) \end{cases} \quad (1)$$

The first inequality in the system enables us to prove Lemma 1. The second inequality allows us to recalculate the dynamic lower bound estimation in Lemma 2. This helps us to prove the worst-case performance B for our algorithm.

The scheme and the algorithm generalize and extend the analysis of Dolgui et al. (2015). As in the latter study, we use the analysis with big and small jobs, a similar lower bound recalculation, as well as the separation of the processors into the Fast, Normal and Reserved classes.

However, there are some major differences. The division into big and small jobs is now parametric in the terms of φ. The analysis is modified to work with any speed s and parametric job separation. Two new approaches for the estimation are used. The first uses the Wolfram Mathematica. The second is essentially a numerical approach with a grid parameter search. The algorithm works for any speed $s > 1$, as well as for any m and k.

Note that the system presented in Dolgui et al. (2015) is obtained from (1) by setting $B = 2+\alpha, z = 2$ and $\varphi = 1/2$. Note also that from $s > 1$ it follows $z \geq 2$.

3.2. Basic properties of the algorithm

Property 2. *If job j does not fit into processor i from the Fast class, then $L_{i,j} > (B-1) \cdot LB_j$.*

Proof. This follows from $p_j/s \leq LB_j$ by the lower bound $V_j^{(3)}$. □

Lemma 1. *For the next current job j during the algorithm execution, there is a Fast processor i with load $L_{i,j} \leq (B-1) \cdot LB_j$, or a Normal processor w with load $L_{w,j} \leq (1-\varphi) \cdot B \cdot LB_j$.*

Proof. Assume the converse: at step j, all Fast processors have loads greater than $(B-1) \cdot LB_j$, and all Normal processors have loads more than $(1-\varphi) \cdot B \cdot LB_j$. Then, we obtain $(p_1 + \ldots + p_j) > k \cdot s \cdot (B-1) \cdot LB_j + m_1 \cdot (1-\varphi) \cdot B \cdot LB_j$. On the other hand, from the first equation from (1), we obtain $(p_1 + \ldots + p_j) > (s \cdot k + m_1 + m_2) \cdot LB_j$. This contradicts Property 1 (by the lower bound $V_j^{(2)}$). □

Corollary 1. *If job j is small, then there is a Fast or Normal processor i, such that job j fits into processor i.*

The next property proves that the system always has a solution.

Property 3. *The system* (1) *always has a feasible solution with fixed $k, m,$ and s.*

Proof. Consider $\varphi = 1, R = 0, m_1 = m - k$, and $m_2 = 0$. The only important inequality for this case is $(B-1) \cdot s \cdot k \geq s \cdot k + m - k$. This inequality means that $B \geq 2 + \frac{m-k}{s \cdot k}$. □

Note that if $R = 0$, the Reserved class will be empty, and only the first point of the algorithm will be executed.

Next we assume that (B, m_1, m_2, R, z) is a solution of (1).

3.3. Case $R \geq 1$

In this section, we suppose that the system (1) has a solution with some $R \geq 1$. Consider any such solution. Since groups are constantly renamed during the algorithm, it is useful to give them some stable notation. Denote the groups $F_1, \ldots, F_R$, where F_i corresponds to G_i in the source numeration(before the algorithm is launched). The only difference is that the numeration F_i will not change during the execution of the algorithm. Denote by $L_{i,j}^r$ the current load of the processor with index i within group $G_r, r = 1, \ldots, R$, before assignment of job j (at iteration j).

Note that for fixed i and r and different j, the value $L_{i,j}^r$ may correspond to different processors. Note also that, in general, $L_{i,j}^x \neq L_{i,j}^y$ when $x \neq y$, because of the correspondence to different groups.

Lemma 2. *For each job j, such that point 2 of the algorithm is executed and processor g_u in group G_1 is changed, it holds that $L_{u,j}^1 \leq (B-s) \cdot LB_j$.*

Proof. We proceed by induction. Assume that before the previous assignment to processor h from group F_i, the load of that processor h was at most $(B-s) \cdot LB_j$. Then, prove that before the next assignment, the load of h will not exceed $(B-s) \cdot LB_{j'}$. The base holds since initially all loads are zero. From the symmetry, it is sufficient to prove the statement for the first processor from the group F_1 at the moment when it is in G_1.

Denote by t_1^h the iteration in which we executed point 2 of the algorithm for the processor with number $u = h, h = 1, \ldots, (z-1) \cdot k$ in the group $G_1 = F_1$. After assignment of big job t_1^h to processor h in group G_1 with the property $L_{h,t_1^h}^1 \leq (B-s) \cdot LB_{t_1^h}$, we obtain that the new load L is within the following borders:

$$\varphi B \cdot LB_{t_1^h} < L \leq B \cdot LB_{t_1^h} \tag{2}$$

From Property 2 for any processor i from the Fast class, we obtain $L_{i,t_1^h} > (B-1) \cdot LB_{t_1^h}$. Thus, from Lemma 1 it follows that there is a processor i from the Normal class, such that $L_{i,t_1^h} \leq (1-\varphi) \cdot B \cdot LB_{t_1^h}$. Then, for the processor i mentioned in point 2 of the algorithm, it holds:

$$L_{i,t_1^h} \leq (1-\varphi) \cdot B \cdot LB_{t_1^h} \tag{3}$$

After the assignment and exchange of processors (but before the group renaming), we obtain:

$$L_{h,t_1^h+1}^1 \leq (1-\varphi) \cdot B \cdot LB_{t_1^h} \tag{4}$$

Note that the conditions (2), (3), and (4) hold for every iteration t_1^h, where $h = 1, \ldots, (z-1) \cdot k$. After $(z-1) \cdot k$ such iterations, we have the group renaming for all G_i. Now G_1 is the previous G_2, i.e. F_2. Therefore, the conditions mentioned above are respected for G_R.

Let t_2^1 be the iteration in which we assign a big job to the processor 1 in the new group G_1, i.e. immediately after renaming the groups at step $t_1^{(z-1) \cdot k}$. Similarly, define t_2^h. Then t_2^1 is big, and $p_{t_2^1} > \varphi \cdot B \cdot LB_{t_2^1}$.

We have $(z-1) \cdot k$ jobs t_1^h, and job t_2^1 with a processing time of at least $\varphi \cdot B \cdot LB_{t_1^1}$. Therefore, from the bound $V_j^{(1)}$ of Property 1, we obtain $\varphi \cdot B \cdot LB_{t_1^1} < LB_{t_2^1}$. We can proceed similarly for all h, i.e. the following holds:

$$\varphi \cdot B \cdot LB_{t_1^h} < LB_{t_2^h} \tag{5}$$

Therefore, combining (5) and (4) we obtain:

$$L_{h,t_1^h}^R \leq (1-\varphi) \cdot B \cdot LB_{t_1^h} \leq \frac{(1-\varphi) \cdot B \cdot LB_{t_2^h}}{\varphi \cdot B} \tag{6}$$

Analogously, we obtain $\varphi \cdot B \cdot LB_{t_2^h} < LB_{t_3^h}$, and so on. As a result, we proceed with a recalculation of the lower bound. After R such recalculations on the iteration for job j, we find that the load of processor 1 in the group F_1 is at most $\frac{(1-\varphi) \cdot B}{(\varphi \cdot B)^R} LB_j$. To finish the proof, we need to note that from the system (1) we have $\frac{(1-\varphi) \cdot B}{(\varphi \cdot B)^R} \leq (B-s)$. □

3.4. Particular case $R = 0$

Lemma 3. *In the case $R = 0$, the algorithm is correct, i.e. only point 1 will be executed. Moreover, then the minimal possible B is equal to $\max\left\{s, 2 + \frac{(s-1) \cdot (d-1)}{s+d-1}\right\}$, where $d = \frac{m}{k} \in \mathbb{Q}$ is the ratio of the number of all processors to the number of fast ones.*

Proof. Consider how the system (1) changes in the case when we fix $R = 0$. Then, $m_1 = m - k$, and $m_2 = 0$. We obtain:

$$\begin{cases} \varphi \cdot B \geq s \Leftrightarrow (1-\varphi) \cdot B \leq B - s \\ B \cdot s \cdot k + m_1 \cdot B - \varphi \cdot B \cdot m_1 \geq 2s \cdot k + m_1 \end{cases} \tag{7}$$

From Lemma 1 and $(1-\varphi) \cdot B \leq (B-s)$, using bound $V_j^{(3)}$ it follows that the algorithm will execute the first step only. Let us prove the part about the minimality of B. Since $\varphi B \geq s$, the best option for the second equation in (7) is when $\varphi B = s$. Thus, the minimal value B for (7) is equal to the result of the minimization of B subject to the following constraints (where $\varphi \in [0, 1]$):

$$\begin{cases} \varphi \cdot B = s \\ B \geq 2 + \frac{(s-1)\cdot(d-1)}{s+d-1} \end{cases} \tag{8}$$

The solution for this system is described by $\max\left\{s, 2 + \frac{(s-1)\cdot(d-1)}{s+d-1}\right\}$. In fact, from $\varphi \cdot B = s$ it follows that $B \geq s$. If the second equation of (8) gives the requirement for B to be at least s, then by adjusting φ we can make $\varphi \cdot B = s$. Otherwise, we can take $\varphi = 1$ and obtain s.

□

Note that the solution with $R = 0$ always exists. The minimization problem for B in the system (1) has a solution not worse than $\max\left\{s, 2 + \frac{(s-1)\cdot(d-1)}{s+d-1}\right\}$.

4. Main Result

4.1. Main theorem

Theorem 1. *For any fixed m, k, and s, the system* (1) *has at least one solution (B, m_1, m_2, z, R). For any solution of the system, the algorithm built upon it has the worst-case performance of at most B.*

Proof. It is enough to show that after each iteration of the algorithm, the load of each processor is at most $B \cdot LB_j$. In other words, each job has a processor into which it fits. The existence of the solution follows directly from Lemma 3. Lemma 3 also implies that all jobs for the case $R = 0$ have some processors they fit into, since only point 1 of the algorithm is executed. Now assume that $R \geq 1$. Property 1 implies that $p_j \leq s \cdot LB_j$ by the bound $V_j^{(3)}$. Then, from Lemma 2 it follows that if we reach point 2 of the algorithm, job p_j will fit into processor g_u. □

According to Theorem 1, it is intuitive to set the minimization problem over B with the conditions shown in the system (1). In fact, the less B is, the better the worst-case performance is. From Lemma 3 we obtain that $B \leq \max\left\{s, 2 + \frac{(s-1)\cdot(d-1)}{s+d-1}\right\}$, where $d = \frac{m}{k}$ is the ratio of the number of all the processors to the number of the fast ones. Note that from the second equation of the system (1), we obtain $B \geq s$.

Property 4. *Let us fix the parameters k, and s. Then, if there is a solution for $m = m_0$ with $B = B_0$, and $(1-\varphi)B_0 \geq 1$, a solution with $B = B_0$ exists for any $m \geq m_0$. In particular, this holds if we minimize over B the system* (1) *with the additional condition $\varphi \leq 1 - \frac{1}{s}$.*

Proof. In the system (1), we may use the same m_2 as for the solution with $m = m_0$. Then we need to check that the first inequality holds. The only changed variable is m_1. Since the coefficient $(1-\varphi) \cdot B$ for m_1 in the Left-Hand-Side(LHS) is at least 1, the inequality holds. The last statement of this property follows from $B \geq s$. □

4.2. Equivalent scheme

Since $m_1 + m_2 + k = m$, we can write an equivalent scheme for the system 1. Let us remember that $d = \frac{m}{k}$.

$$\begin{cases} B \cdot (s + (1-\varphi) \cdot (d - 1 - R \cdot (z-1))) \geq 2s + d - 1 \\ d \geq 1 + R \cdot (z-1) \\ (1-\varphi) \cdot B \leq (\varphi \cdot B)^R \cdot (B - s) \\ z = \lceil s \rceil, \varphi \in [0, 1], R \geq 0, R \in \mathbb{Z} \end{cases} \tag{9}$$

Note that for this new scheme, it is enough to know the ratio d instead of knowing both m and k. The equivalence here means that if we have some solution (B, φ, R) for (1), these values will work for (9), and vice versa.

Let us show that (9) is equivalent to (1). To see this, substitute $d = \frac{m}{k}$ into (9). Then the second inequality of (9) becomes $m \geq k + R \cdot k \cdot (z-1)$. Set $m_2 = R \cdot k \cdot (z-1)$, and set $m_1 = m - k - m_2$. Note that $m_1 \geq 0$. The first inequality in (9) becomes $2 \cdot s \cdot k + m - k \leq B \cdot s \cdot k + B \cdot (1-\varphi) \cdot (m - k - R \cdot k \cdot (z-1))$. This is equivalent to the first inequality of (1).

4.3. Particular case $R = 1$

Let $R = 1$. Then from $d \geq 1 + R \cdot (z-1)$, it follows that $d \geq z$. Then the conditions of the equivalent system are the following:

$$\begin{cases} B \geq s - 1 + 1/\varphi = f_1(s, \varphi, d) \\ B \geq \frac{2s+d-1}{s+(1-\varphi)\cdot(d-z)} = f_2(s, \varphi, d) \end{cases} \tag{10}$$

Here we have two limitations for B. Fix d, s, and z and allow φ to change. Then the upper condition decreases monotonously by φ, tending to positive infinity with $\varphi \to 0$. The lower condition increases monotonously with φ, thus the minimal value is obtained with $\varphi = 0$, and the maximal value is obtained with $\varphi = 1$.

Suppose that the maximal value of f_2 is less than the minimal value of f_1. The minimal value of f_1 equals to s and is reached when $\varphi = 1$. Then we have $2 + \frac{d-1}{s} \leq s$, which is equivalent to $d < (s-1)^2$. Thus, if $d < (s-1)^2$, then the minimal possible B is s.

Suppose $d > (s-1)^2$. Then the best value for B is obtained with the equality $f_1(\varphi) = f_2(\varphi)$. From the monotonous properties, this solution exists and is unique. Thus, we obtain the following second-order equation with respect to the variable φ:

$$s - 1 + \frac{1}{\varphi} = \frac{2s + d - 1}{s + (1-\varphi) \cdot (d - z)} \tag{11}$$

When $d \to \infty$, this equation is equivalent to $s - 1 + \frac{1}{\varphi} = \frac{1}{1-\varphi}$. After substituting the solution of the new equation to f_1, we obtain $\frac{1}{2}\left(1 + s + \sqrt{5 - 2s + s^2}\right)$. We proved the following result.

Lemma 4. *With $d \to \infty$, it holds that $B \leq \frac{1}{2}\left(1 + s + \sqrt{5 - 2s + s^2}\right)$.*

Lemma 5. *If $d \leq (s-1)^2$, then the worst-case performance $B \leq s$. If $d > (s-1)^2$, then the optimal B is not greater than the value obtained when equality* (11) *holds.*

Proof. This immediately follows from the previous discussion. □

The solution for (11) can be written explicitly. We have $\varphi = \frac{1-3d+2z-3s+d\cdot s-z\cdot s+s^2+\sqrt{M}}{2(d-z)\cdot(s-1)}$, where $M = -4(-d+z-s)\cdot(-d+z+d\cdot s-z\cdot s)+(-1+3d-2z+3s-d\cdot s+z\cdot s-s^2)^2$. Then the optimal B is at most $s-1+1/\varphi$.

4.4. Merging the estimations for $R = 0$ and $R = 1$ in the case $1 < s \leq 2$

Lemma 6. *Suppose that $1 < s \leq 2$. Then for any d, the optimal value of B is less than or equal to $\frac{3+\sqrt{5}}{2} \approx 2.618$.*

Proof. From $1 < s \leq 2$, it follows that $z = 2$. Consider two cases. In the first, $d \leq 3$ and in the second $d > 3$.

Suppose that $d \leq 3$. Then let us use the algorithm and its estimation for the particular case $R = 0$. Since the estimation is monotonous over d, the maximal value of B is obtained for $d = 3$. Here we obtain 2.5, which is less than 2.618.

Now assume that $d \geq 3$. Then use the algorithm and its estimation for the particular case $R = 1$. Since $d > (s-1)^2$ and $d > n$, from Lemma 5 it follows that the optimal B is less than or equal to the value obtained by solving equation (11).

This solution can be written explicitly with respect to φ. After natural substitution, we obtain a conditional optimization problem with conditions $1 < s \leq 2$ and $d \geq 3$. This is a standard extremum search problem, which can be solved analytically. We solve it with the Wolfram Mathematica and the notebook print is presented in the Online Appendix C. The maximal limitation for B is $\frac{3+\sqrt{5}}{2} \approx 2.618$, and is reached when $s = 2$ and $d \to \infty$.

□

5. Further Notes

The source code, as well as the Wolfram Mathematica notebook and files with online appendices, can be found on the Internet [1].

Some improvements in the proposed scheme, as well as some possible new research directions, are presented in the Online Appendix A. Since we do not have an explicit formula for B, it is natural to use a computational approach. Our computational approach and its results are presented in the Online Appendix B.

The table in the Online Appendix B presents the worst-case performance found with the computation approach for $s = 2$ depending on $m - k$ (the number of unit processors) and k (the number of fast processors).

6. Discussion

The obtained asymptotic worst-case performance when $d \to \infty$ is better than the result of Cho and Sahni (1980).

The table presented in the Online Appendix B shows better results than those achieved in Dolgui et al. (2015) (e.g. we have a better value for $m = 39, k = 1$, which is 2.41 in that paper and 2.372 in this paper). It is natural to expect that since the new scheme is more flexible (the scheme from Dolgui et al. (2015) is a particular case of our scheme), it will provide better results in other cases.

7. Conclusion

A parametric scheme is proposed, with a polynomial approximation algorithm based on the scheme. The asymptotic worst-case performance is estimated when $d \to \infty$. For the case $1 < s \leq 2$, we prove the worst-case performance of 2.618. This works for any ratio of the number of all the processors to the number of fast ones.

8. Acknowledgments

This research was supported in part by Labex IMOBS3 and FEDER Funding. Vladimir Kotov was also supported in part by BRFFI F15MLD-022.

References

Angelelli, E., Speranza, M., Tuza, Z., 2008. Semi-online scheduling on two uniform processors. Theoretical Computer Science 393, 211–219.

Cao, Q., Liu, Z., 2010. Online scheduling with reassignment on two uniform machines. Theoretical Computer Science 411, 2890–2898.

Cheng, T., Ng, C., Kotov, V., 2006. A new algorithm for online uniform-machine scheduling to minimize the makespan. Information Processing Letters 99, 102–105.

Cho, Y., Sahni, S., 1980. Bounds for list schedules on uniform processors. SIAM J. Comput. 9, 91–103.

Dolgui, A., Quilliot, A., Kotov, V., 2015. A parametric scheme for online uniform-machine scheduling to minimize the makespan. Buletinul Academiei de Stiinte a Republicii Moldova. Matematica 79(3), 102–109.

Dosa, G., Wang, Y., Han, X., Guo, H., 2011. Online scheduling with rearrangement on two related machines. Theoretical Computer Science 412, 642–653.

Graham, R., 1969. Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math 17, 263–269.

Kellerer, H., Kotov, V., 2013. An efficient algorithm for bin stretching. Operations Research Letters 41(4), 343–346.

Kellerer, H., Kotov, V., Gabay, M., 2015. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. Journal of Scheduling 18, 623–630.

Lee, K., Leung, J.-T., Pinedo, M., 2009. Online scheduling on two uniform machines subject to eligibility constraints. Theoretical Computer Science 410, 3975–3981.

Li, R., Shi, L., 1998. An on-line algorithm for some uniform processor scheduling. SIAM J. Comput. 27, 414–422.

Liu, M., Xu, Y., Chu, C., Zhengin, F., 2009. Online scheduling on two uniform machines to minimize the makespan. Theoretical Computer Science 410, 2099 – 2109.

Min, X., Liu, J., Wang, Y., 2011. Optimal semi-online algorithms for scheduling problems with reassignment on two identical machines. Information Processing Letters 111, 423–428.

Wang, Y., Benko, A., Chen, X., Dosa, G., Guo, H., Han, X., Lanyi, C., 2012. Online scheduling with one rearrangement at the end: Revisited. Information Processing Letters 112, 641–645.

Wen, J., Du, D., 1998. Preemptive on-line scheduling for two uniform processors. Operations Research Letters 23, 113–116.

[1] `https://github.com/nekrald/parametric_scheme_uniform_scheduling`