European Journal of Operational Research 297 (2022) 844–852

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A first Fit type algorithm for the coupled task scheduling problem with unit execution time and two exact delays

József Békési[a,1,*], György Dósa[b,2], Gábor Galambos[c]

[a] Department of Computer Algorithms and Artificial Intelligence, University of Szeged, POB 652, Szeged H-6701, Hungary
[b] Pannon University, Egyetem str. 10., Veszprém H-8200, Hungary
[c] Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, POB 361, Szeged H-6701, Hungary

ARTICLE INFO

Article history:
Received 28 March 2020
Accepted 1 June 2021
Available online 9 June 2021

Keywords:
Combinatorial optimization
Scheduling
Approximation algorithm

ABSTRACT

The considered coupled task problem (CTP) is to schedule n jobs, each consisting of two (sub)tasks, on a single machine. Exact delay times are between the subtasks of a job and the makespan has to be minimized. It has been proven that the problem is strongly $\mathcal{NP}$-hard in general case (see Orman and Potts (1997)), even if the lengths of the subtasks are identical. This paper considers a special case of CTP where there are jobs with two different delay times only. The complexity status of this problem is unknown. We will present an algorithm – called First Fit Decreasing (FFD) – and we will prove that its approximation ratio is in the interval $(1.57894, 1.57916)$.

1. Introduction

The single-machine coupled task problem (CTP) is defined as follows: An instance is given by $I_n = \{J_1, J_2, \ldots, J_n\}$, where each J_i is a job consisting of two tasks, which are denoted by $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_n\}$, respectively. Tasks have to be executed in a given sequence and an exact delay time (idle time) – l_i, $(i = 1, \ldots, n)$ – is required between their executions. For job J_i the processing times of the two tasks will be denoted by $p(a_i)$ and $p(b_i)$, resp. The machine can process at most one task at a time and no preemption is allowed. In a solution for job J_i $S(a_i)$ and $S(b_i)$ denote the starting times of the two tasks, and a solution is given by the first task's starting time of each job. The completion time of J_i is $C(J_i) = S(a_i) + p(a_i) + l_i + p(b_i)$. A solution – also called schedule – is feasible, if for any time-slot at most one task is performed by the machine. The quality of a (feasible) solution σ is evaluated by the maximum completion time (also called makespan):

$$C^{\sigma}_{\max} = \max\{C(J_i) | J_i \in I_n\}.$$

The objective is to find a feasible schedule that minimizes the maximum completion time.

We will usually refer to jobs and tasks just by their indices and – in order to simplify notations – we will use the identical notations a_i and b_i for the processing times of the tasks.

The CTP appears in numerous practical problems. The most citations can be found for pulsed radar systems where a radar has to keep track of different targets by transmitting a pulse, and receiving its reflection (see eg. Orman & Potts, 1997; Orman, Potts, Shahani, & Moore, 1996). The interval between the transmission and reception depends on the distance of the target. Similar examples from the field of agriculture and chemistry can also be cited (Ageev & Baburin, 2007; Ageev & Ivanov, 2016).

1.1. Related works

It is apparent that the pioneer complexity paper of Orman & Potts (1997) gave an intensive movement to the research of this topic, and an extensive study of the CTP with different objective functions has therefore been developed in last few years. Recently, a state of art survey was published by Khatami, Salehipour, & Cheng (2020), which is adequate for establishing the knowledge of the enquirer reader. For the general case, we will use the standard three-field notation – introduced by Graham, Lawler, Lenstra, & Rinnooy Kan (1979) – as follows: $1|Coup\text{-}Task, exact\ l_i|C_{\max}$.

The complexity of this problem was deeply analyzed in Orman & Potts (1997) and – in the general case – shown to be strongly $\mathcal{NP}$-hard. To avoid technical complications for special cases in which all jobs are identical, in their paper (Orman & Potts, 1997)

* Corresponding author.
E-mail addresses: bekesi@inf.u-szeged.hu (J. Békési), dosagy@almos.vein.hu (G. Dósa), galambos@jgypk.szte.hu (G. Galambos).

[1] J. Békési was supported by the EU-funded Hungarian grant "Integrated program for training new generation of scientists in the fields of computer science" [grant number EFOP-3.6.3-VEKOP-16-2017-0002]; and the National Research, Development and Innovation Office NKFIH [grant number SNN 129178].
[2] Gy. Dósa was supported by the EU-funded Hungarian grant [grant number EFOP-3.6.1-16-2016-00015]; and the National Research, Development and Innovation Office NKFIH [grant number SNN 129364].

https://doi.org/10.1016/j.ejor.2021.06.002

Table 1
Worst case performances of analysed approximation algorithms.

Problem	$\rho(A)$	T.compl.	Inapprox.	Ref.
l_j, a_j, b_j	≤ 3.5	$O(n \log n)$	$2-\varepsilon$	Ageev & Kononov (2007)
$l_j, a_j \leq b_j$	≤ 3	$O(n \log n)$	$2-\varepsilon$	Ageev & Kononov (2007)
$l_j, a_j = b_j$	≤ 2.5	$O(n \log n)$	$2-\varepsilon$	Ageev & Kononov (2007)
$l_j = L, a_j, b_j$	≤ 3	$O(n \log n)$	$1.5-\varepsilon$	Ageev & Ivanov (2016)
$l_j = L, a_j \leq b_j$	≤ 2	$O(n \log n)$	$1.5-\varepsilon$	Ageev & Ivanov (2016)
$l_j = L, a_j = b_j$	≤ 1.5	$O(n \log n)$	$1.25-\varepsilon$	Ageev & Ivanov (2016)
l_j, UET	$= 1.75$	$O(n \log n)$		Ageev & Baburin (2007) and Békési et al. (2009)
$l_j \in \{L_1, L_2\}$, UET	[1.5789,1.5791)	$O(n)$		this paper

Orman and Potts assumed that processing times and idle times are inputs for all jobs, thus they gave an input size of $O(n)$ rather than $O(\log n)$. The latter scheduling papers accepted this assumption, and we also use this condition for all inputs. The problem remains $\mathcal{NP}$-hard even in numerous special cases where there are restrictions either for the processing times of the tasks or for the delay times, and only a few of them can be solved in polynomial time (see Li & Zhao, 2007; Orman & Potts, 1997). One exception to a rule is the Identical Coupled Task Problem (ICTP) where $a_i = a$, $l_i = L$, and $b_i = b$ for each job. This problem was studied extensively later and different pseudo-polynomial algorithms have been developed, – e.g. Ahr, Békési, Galambos, Oswald, & Reinelt (2004) – but its complexity is still open.

For $\mathcal{NP}$-hard CTP problems, different approximation algorithms have been developed, and their effectiveness are measured by the *performance ratio* which is defined as follows. Let I_n be an instance of n jobs. Denote by $C^A_{\max}(I_n)$ and $C^*_{\max}(I_n)$ the makespans produced by algorithm A and an optimal schedule of I_n, respectively. Algorithm A is called *ρ-approximation algorithm* for some $\rho \geq 1$ if

$$C^A_{\max}(I_n) \leq \rho\, C^*_{\max}(I_n)$$

for every instance of jobs. The smallest possible value ρ is called *worst-case (performance) ratio* of algorithm A, and is denoted by ρ_A.

We summarized the recent results concerning the approximation algorithms for different $\mathcal{NP}$-complete problems in Table 1. As the table shows there is only one algorithm which is concerned with the general case, and most of the algorithms give only upper bounds for the performance ratios. The non-approximability results concerning the different cases are also included the table.

1.2. Our results

Among the special cases, in Orman & Potts (1997) the problem $1|$*Coup-Task, exact* $l_i, a_i = b_i = p|C_{\max}$ was also examined. Orman et al. proved that for these inputs the problem is $\mathcal{NP}$-hard if the delay times are arbitrary. If we suppose that $a_i = b_i = 1$, then we call this problem *Unit Execution Time* – UET – problem.)

For the case $1|$*Coup-Task, exact* $l_i = L, a_i = b_i = p|C_{\max}$ Orman and Potts proved that the problem is in $\mathcal{P}$, and gave an optimal algorithm with time complexity $O(n)$. (The exact description of the algorithm see later.) Similarly to the ICTP, the complexity of $1|$*Coup-Task, exact* $l_i \in \{L_1, L_2\}, a_i = b_i = p|C_{\max}$ is still open. Here, L_1 and L_2 denote two different delay times. In this paper, we investigate this problem. The basis of our motivation is that in practical problems there are almost always special conditions. In the case of an aircraft-carriers, there is a bounded number of airplanes with different ranges, and it is plausible that only two different planes are used in a deployment. As another example let us consider the car production assembly-line. If we suppose, that during the polishing process, two different phases are required and two basic types of polishing row-materials exist (metal and normal), then the schedule can also be handled by such model.

We introduce the *First-Fit Decreasing* (*FFD*) approximation algorithm for the problem

$1|$*Coup-Task, exact* $l_i \in \{L_1, L_2\},\ a_i = b_i = 1|C_{\max}$.

The FFD algorithm was originally defined by Johnson (1973) as a bin packing approximation algorithm. We define the appropriate version of this algorithm to our problem in consideration. Naturally, the algorithm is designed to solve problems with arbitrary number of different delays but in this paper we will investigate it only for the UET problem with two distinct delays among tasks.

Algorithm First-Fit Decreasing

Step 1. Sort the jobs in nonincreasing order according to the idle times. After ordering we suppose that $l_1 \geq l_2 \geq \ldots \geq l_n$. Let $i = 1$.
Step 2. Schedule job J_i from the earliest time which results in a feasible schedule.
Step 3. $i = i + 1$. If $i \leq n$ then goto Step 2. Otherwise END.

The time complexity of the algorithm is $O(n)$ (see Claim 4.2).

We will investigate the algorithm *FFD* from the worst-case point of view, and prove that

$$1.57894\ldots \approx \frac{30}{19} \leq \rho_{FFD} < \frac{\sqrt{11}+3}{4} \approx 1.57916\ldots.$$

In what follows, in Section 2, we start with preliminaries, where we give some basic definitions and show the lower bounds that will be used in the proofs. Section 3 deals with instances with equal delays. We prove that the *FFD* gives an optimal solution to this problem, and we give the exact value of the makespan. Section 4 contains the complete analysis of *FFD* for the instances that have two different delay times. Finally, we provide some concluding remarks in Section 5.

2. Preliminaries

Hereinafter we will consider UET problems. Let us consider an arbitrary feasible schedule σ. We will say that in σ J_i and J_j are *consecutive* if $S(b_i) < S(a_j)$, the jobs are *nested* if $S(a_i) < S(a_j) < S(b_j) < S(b_i)$, and the jobs are *interleaved* if $S(a_i) < S(a_j) < S(b_i) < S(b_j)$.

A similar definition can be done for the set of jobs: e.g. let S_1 and S_2 be sets of jobs. In schedule σ S_2 is *nested* in S_1 if $\forall J_i \in S_1$ and $\forall J_j \in S_2$, $S(a_i) < S(a_j) < S(b_j) < S(b_i)$.

If a task is performed in the time-period $[i-1, i]$, then we say that the task occupies the position i. In schedule σ the subset $\{J_i, \ldots, J_m\}$ of interleaved jobs form a *block*, if $\forall j, k,\ i \leq j < k \leq m$, $L_j = L_k$, and $S(a_j) < S(a_k)$. Jobs J_i and J_m are the *first-* and the *last-item* of the block, resp. We call the sum of the remaining idle times within a block *gap*. If there is no gap within the block, then we say that the block is *complete*, otherwise it is *incomplete*.

From the literature, there are some lower bounds known to estimate the optimal schedule for the CTP.

$$C^*_{\max}(I_n) \geq LB1 = 2n. \tag{1}$$

The next one gives an improvement for those cases where $\sum l_i \geq n(n-1)$ (see Békési, Galambos, Oswald, & Reinelt, 2009).

$$C^*_{\max}(I_n) \geq LB2 = 2n + \left\lceil \frac{1}{n}\left(\sum_{i=1}^{n} l_i - n(n-1)\right)\right\rceil. \quad (2)$$

If we apply the lower bound (2) for inputs with two different delays we can use the following estimation.

$$C^*_{\max}(I_n) \geq LB3 = n_1 + n_2 + \frac{n_1L_1 + n_2L_2}{n_1 + n_2} + 1, \quad (3)$$

where n_1 and n_2, mean the number of jobs with longer and shorter delays, respectively. At the end, if we have inputs with two different delays then the first part of the input – with n_1 jobs and L_1 delays – gives also a lower bound.

$$C^*_{\max}(I_n) \geq LB4 = n_1 + L_1 + 1. \quad (4)$$

3. Theorems for scheduling jobs with equal delays

Let I_n be an instance with jobs of equal delay times, and let us denote the common delay time by L. Such instance will be called *L-uniform* instance. We will say that the subset $\{J_i, \ldots, J_m\}$ of jobs are *continuously* scheduled if $\forall j,\ i \leq j < m, S(a_{j+1}) = S(a_j) + 1$. It is clear that for an L-uniform instance *FFD* schedules the items in a block continuously. Furthermore, let J_k and J_{k+1} be the last item of block B, and the first item of block $B+1$, respectively. Then if $S(b_k) + 1 = S(a_{k+1})$, then the two blocks are also continuously scheduled.

Let I_n be an L-uniform instance. Let $k = \lfloor n/(L+1) \rfloor$, and $n = n_c + n_r$, where $n_c = k(L+1)$, and $n_r = n - k(L+1) \leq L$. In Orman & Potts (1997) an optimal algorithm has been defined for the problem $1|CTP, a_i = b_i = p, l_i = L|C_{\max}$. If we apply this algorithm for the case $a_i = b_i = 1$, then we get the following algorithm.

Algorithm Greedy

Step 1. Compute $k = \lfloor n/(L+1) \rfloor$.

Step 2. Form k (complete) blocks of jobs, where each block contains $L+1$ jobs.

Step 3. If $n_r > 0$, then form an incomplete block containing all the remaining jobs where the first tasks are scheduled continuously.

Step 4. Schedule the complete blocks and the incomplete blocks – if any – continuously.

This algorithm in Orman & Potts (1997) has been analysed and it was proven to provide an optimal schedule. The value of $C^*_{\max}(I_n)$ was not given. Here, we give a simpler – but more formalized – proof, and we give $C^*_{\max}(I_n)$ explicitly.

Theorem 3.1. *Algorithm Greedy generates an optimal schedule for the* $1|Coup - Task, a_i = b_i = 1, l_i = L|C_{\max}$ *problem in* $O(n)$ *time, and*

$$C^*_{\max}(I_n) = \begin{cases} k(L+1) + n, & \text{if } n_r = 0; \\ (k+1)(L+1) + n, & \text{otherwise.} \end{cases}$$

Proof. We already know that Greedy generates an optimal schedule, so we prove only the validity of the formula.

First consider the case when all blocks are complete, i.e. $n_r = 0$. Then $C^*_{\max}(I_n) = 2k(L+1)$ and $n = k(L+1)$ thus the claim holds.

Now suppose that the last block is not complete. The contribution of k complete blocks to the makespan is $2(L+1)k$. The length of the last block is $L + 2 + (n_r - 1)$ where $n_r = n - k(L+1)$. Thus we get

$$\begin{aligned} C^*_{\max}(I_n) &= 2(L+1)k + L + 2 + n - k(L+1) - 1 \\ &= 2(L+1)k + L + 1 - k(L+1) + n \\ &= (k+1)(L+1) + n. \end{aligned}$$

□

Since for L-uniform instances an *FFD* schedule is identical to *Greedy*, the following corollary is a consequence of Theorem 3.1.

Corollary For L-uniform instances

$$C^{FFD}_{\max}(I_n) = \begin{cases} k(L+1) + n, & \text{if } n_r = 0; \\ (k+1)(L+1) + n, & \text{otherwise.} \end{cases} \quad (5)$$

We realize that $C^{FFD}_{\max}(I_n)$ is a function of n, and L. Let us denote this function by

$$C^{FFD}_{\max}(I_n) = f(n, L). \quad (6)$$

Considering the formula of (5), we see that moving from n to $(n+1)$, $f(n, L)$ grows by 1 if the last block is not complete (for n), otherwise (if the last block is complete for n) the value of (5) is growing by $L+2$. In both cases, by $n \to (n+1)$, the value of (5) is growing by at least 1. Thus, if we increase n by $d \geq 1$, the value of (5) is growing by at least d. Thus we have

$$f(n-d, L) \leq f(n, L) - d \quad (7)$$

for any $1 \leq d \leq n$.

4. Theorems for two different delays

Hereinafter, we suppose that instances contain only jobs with two different delay times L_1 and L_2, where $L_1 > L_2$. Jobs with delay time L_1, or L_2 are called *long*, or *short*, resp. Let $I_n = (I^1, I^2)$ be the concatenation of instances I^1 and I^2, with n_1 long and n_2 short jobs ($n = n_1 + n_2$), resp. To avoid the complicated notations sometimes we write instead of I_n simply I.

If we have two different delay times, then Step 1 in algorithm *FFD* can be performed in $O(n)$ time creating two "heaps" for the long and short jobs, respectively.

In Section 2 we defined the set of interleaved jobs. If we have jobs with two different gaps, then a (set) of short jobs can be in interleaved position either with a (set) of long jobs or with a (set) of short jobs. In these cases, we speak about *long-interleaved* or *short-interleaved* jobs, resp. If for a short job J_i, $S(a_i) > S(b_{n_1})$ $(i > n_1,)$ then we call the job *long-consecutive*.

We define the algorithm Separate, denoted by *SEP*. This algorithm applies the *FFD* rule independently for the subinstances I^1, and I^2, and concatenates the two schedules.

Algorithm Separate

Step 1. $t = 1$. Schedule all jobs in I^1 from the position t using *FFD*.

Step 2. $t = C^{FFD}_{\max}(I^1) + 1$. Schedule all jobs in I^2 starting at position t by the algorithm *FFD*.

First, we prove a simple Claim that we use several times thereinafter.

Claim 4.1. *If the FFD schedule of input* I_n *contains long-interleaved short jobs, and there is at least one empty position just after the second task of the last long job, then the idle time (denoted by* τ*) in the FFD schedule is at most* L_2.

Proof. During the proof we use the following – easily provable – facts. If the conditions are true then (a) there is at least one nested complete short block, (b) there is no incomplete nested short block, (c) the first long-interleaved short job starts just after the second task of the last nested short job.

Because of the conditions, there is at least one empty position just after the second task of the last long job. Let us denote the number of complete nested blocks of short jobs by $n_{2,cn}$. There are two cases.

Case A. Suppose that $n_2 \geq (L_1 - n_1 + 1) - 2(L_2 + 1)n_{2,cn}$.

In this case, the gap within the incomplete long block will be filled with nested jobs and the first tasks of interleaved short jobs.

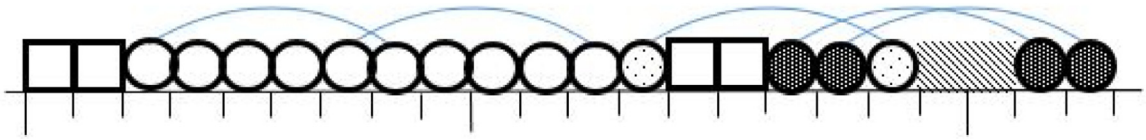

Fig. 4.1. An example for two different types of interleaved short jobs. Boxes denote the long jobs and circles correspond to short jobs.

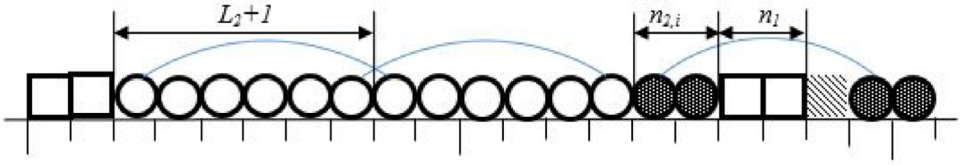

Fig. 4.2. Schedule of short interleaved jobs if second tasks are not consecutive.

Therefore, there is no gap within the incomplete block of the long jobs. So, idle time can occur only after the incomplete block of the long jobs. By this reason, the sum of idle times in the *FFD* schedule – independently whether or not long-consecutive short jobs exist – is at most L_2.

Case B. Suppose that $n_2 < (L_1 - n_1 + 1) - 2(L_2 + 1)n_{2,cn}$.

Now, there is no long-consecutive short job. A gap remains within the incomplete block of the long jobs, and there is also an idle time after the second task of the last long job. Let us denote these gaps by τ_1, and τ_2, resp. Both gaps start up within the two tasks of the first interleaved short job (see Fig. 4.2), thus $\tau = \tau_1 + \tau_2 < L_2$. □

Lemma 4.1. *For any input I with two different delays $C^{SEP}_{\max}(I) \geq C^{FFD}_{\max}(I)$.*

Proof. For scheduling the long jobs *FFD* and *SEP* are identical, so $C^{FFD}_{\max}(I^1) = C^{SEP}_{\max}(I^1)$. Furthermore, the two algorithms remain identical if there are only complete blocks from long jobs. So we can suppose that an incomplete block from long jobs exists. Let us apply the equality (6) for the instances I^1 and I^2 with parameters (n_1, L_1), and (n_2, L_2) We get

$$C^{SEP}_{\max}(I) = C^{FFD}_{\max}(I^1) + C^{FFD}_{\max}(I^2) = f(n_1, L_1) + f(n_2, L_2) = t + f(n_2, L_2).$$

Denote the number of nested short jobs by $n_{2,ne} \geq 0$. Now we distinguish several cases.

Case A. There is no (long-)interleaved short job. Then *FFD* schedules $n_2 - n_{2,ne}$ jobs from time t, so

$$C^{FFD}_{\max}(I) = t + f(n_2 - n_{2,ne}, L_2) \leq t + f(n_2, L_2) - n_{2,ne} \leq C^{SEP}_{\max}(I).$$

Case B. There is at least one long-interleaved short job. Suppose the last interleaved short job ends at position $p > t$. It follows that positions $j = t, \ldots, p-1$ are either idles or they are occupied by the second tasks of some interleaved short jobs.

Case B.1. There is no empty position between t and $p-1$. This means that there are $n_{2,i} = p - t$ interleaved short jobs.

Let $n_{2,lc} \geq 0$ be the number of long-consecutive short jobs. Since there are also $n_{2,ne}$ nested small jobs, $n_{2,lc} = n_2 - n_{2,ne} - n_{2,i}$. The long-consecutive short jobs are scheduled by *FFD* from position p. Then

$$\begin{aligned} C^{FFD}_{\max}(I) &= p + f(n_2 - n_{2,ne} - (p - t), L_2) \\ &\leq p + f(n_2, L_2) - n_{2,ne} - (p - t) \\ &= t + f(n_2, L_2) - n_{2,ne} \leq SEP(I). \end{aligned}$$

Case B.2. There is at least one empty position between t and $p-1$. Now, we can use the Claim 4.1, and so, $\tau \leq L_2$.

In this case the first interleaved short job ends strictly later than $t+1$, and the incomplete long block will be filled with nested jobs and the first tasks of interleaved short jobs. By this reason, the sum of idle times in the *FFD* schedule is at most L_2.

On the other hand, in the *SEP* schedule all positions in the gap of the incomplete long block remain idle. Since in the *FFD* schedule there is at least one complete block of nested jobs, the length of the gap is at least $2(L_2 + 1) > L_2$. Thus the total length of idle intervals in the *SEP* schedule is bigger than L_2. Then clearly $C^{FFD}_{\max}(I) < C^{SEP}_{\max}(I)$. □

At this point we are able to easily determine the time complexity of algorithm *FFD*. We assume that L_1 and L_2 are fixed.

Claim 4.2. *The time complexity of the FFD algorithm is $O(n)$.*

Proof. The ordering in Step 1 needs only $O(n)$ time, since there are only two different types of jobs. In Step 2, *FFD* first schedules the long jobs in $O(n)$ time. Then come the short jobs. From this point we store (in a vector) that what time-slots are occupied and what time slots are free. Note, that any time slot will be checked at most once: if the time slot will be occupied by the current job, this time slot will be never checked again. Otherwise, if the time slot is free but cannot be occupied by the current (short) job, it never will be able to be occupied by some other short job.

We know from Lemma 4.1 that $C^{FFD}_{\max}(I) \leq C^{SEP}_{\max}(I) \leq 2n + L_1 + L_2$ holds, it means that at most so many time slots are tried (each one is tried at most once). We conclude that the complexity is $O(n)$. □

Let us note that since we have only two distinct delays, the instance size is $\log(\max L_1, L_2)$. It implies that an O(n) time algorithm is not a polynomial time algorithm.

4.1. Upper bounds

In this subsection, we give an upper bound for the performance ratio of *FFD*. This ratio depends strongly on the structure of an instance i.e. does the *FFD* schedule contain complete blocks, are there nested short jobs, or interleaved short jobs, etc? We will investigate *FFD* schedules with different structures. In the following, if for two instances I and I', $C^{FFD}_{\max}(I)/C^{*}_{\max}(I) \leq C^{FFD}_{\max}(I')/C^{*}_{\max}(I')$, then we say that $C^{FFD}_{\max}(I')$ *dominates* $C^{FFD}_{\max}(I)$. Our upper bound proof will be divided into three parts.

- First, we will consider instances for which $C^{FFD}_{\max}(I)/C^{*}_{\max}(I) \leq \frac{3}{2}$.
- Secondly, we investigate inputs for which there is a dominant instance with a given structure.
- Finally, we give an upper bound for the set of dominant instances.

In the sequel, we will use the following notations. Let k_b, k_s, r_b, and r_s be such integers, for which $n_1 = k_b(L_1 + 1) + r_b$ where

$1 \le r_b \le L_1$ and $n_2 = k_s(L_2+1)+r_s$ where $0 \le r_s \le L_2$. Then

$$k_b(L_1+1) = n_1 - r_b \quad \text{and} \quad k_s(L_2+1) = n_2 - r_s. \tag{8}$$

4.1.1. Inputs with performance ratio at most $\frac{3}{2}$.

Lemma 4.2. *If I is such an input for which there is at least one complete block of long jobs in the FFD schedule, then $\frac{C^{FFD}_{\max}(I)}{C^{*}_{\max}(I)} \le 3/2$.*

Proof. Since there is a complete block of long jobs, we have

$$L_1 + 1 \le n_1. \tag{9}$$

Applying (6) for the instances I^1 and I^2 separately, and taking into account that for L-uniform instances $C^{FFD}_{\max}(I) = C^{SEP}_{\max}(I)$, we get that

$$C^{SEP}_{\max}(I) = f(n_1, L_1) + f(n_2, L_2). \tag{10}$$

On the other hand, applying Theorem 3.1 for I^1, we have

$$C^{*}_{\max}(I) \ge C^{*}_{\max}(I^1) = f(n_1, L_1).$$

Case A. If $k_s = 0$ (i.e. there is no complete block of short jobs in the *SEP* schedule), then we first apply the lower bound (4). Then using the inequality (9) and the lower bound (1), we have

$$\begin{aligned} C^{SEP}_{\max}(I) - C^{*}_{\max}(I) &\le f(n_2, L_2) = L_2 + 1 + n_2 \\ &\le L_1 + n_2 < n_1 + n_2 = n \le C^{*}_{\max}(I)/2, \end{aligned}$$

thus $C^{SEP}_{\max}(I) \le (3/2)C^{*}_{\max}(I)$ and we are done.

Case B. If $k_s > 0$ (i.e. there is at least one complete block also from the short jobs in the *SEP* schedule), then

$$L_2 + 1 \le n_2. \tag{11}$$

Let us start with (10) and apply (8)- (11). Then we get

$$\begin{aligned} C^{SEP}_{\max}(I) &= f(n_1, L_1) + f(n_2, L_2) \\ &\le (k_b+1)(L_1+1) + n_1 + (k_s+1)(L_2+1) + n_2 \\ &= n_1 - r_b + (L_1+1) + n_1 + n_2 - r_s + (L_2+1) + n_2 \\ &\le 2n_1 + (L_1+1) + 2n_2 + (L_2+1) < 3n_1 + 3n_2 = 3n \\ &\le (3/2)C^{*}_{\max}(I) \end{aligned}$$

where the last inequality follows from the lower bound (1). □

Lemma 4.3. *Let us suppose that $k_b = 0$, i.e. there is no complete block from the big jobs. If there is at least one complete block of short jobs, then $\frac{C^{FFD}_{\max}(I)}{C^{*}_{\max}(I)} \le 3/2$.*

Proof. Let us denote the number of short jobs in complete blocks by $n_{2,c}$. Then

$$n_2 \ge n_{2,c} \ge L_2 + 1. \tag{12}$$

If all of the short jobs are nested jobs then the *FFD* schedule is optimal. So, we can suppose that there is at least one short job i with $S(b_i) \ge S(b_{n_1}) + 1$. Let us denote the sum of idle times in the *FFD* schedule by τ. If $\tau \le n$ then $C^{FFD}_{\max}(I) \le 3n$. Thus it is enough to show that $\tau \le n$.

Case A. There is at least one long-interleaved short job, and these jobs are scheduled so that the second tasks are *not* consecutive to the last task of the long jobs (see Fig. 4.2). Now, we apply Lemma 4.1, which results that $\tau \le L_2$. From (12) we get $L_2 < n_2 < n$. Therefore $\tau < n$, and we are done.

Case B. There is at least one long-interleaved short job, and these jobs are scheduled so that the second tasks are consecutive to the last task of the long jobs. This schedule of the interleaved jobs does not result in gap just after the second task of the last long job (see Fig. 4.3).

Let us consider the tasks which are scheduled in the gap of the last long-interleaved short job. These tasks are the second tasks of the last incomplete block of nested jobs (if any), the second tasks of the long jobs and second tasks of the long-interleaved jobs except the last one. Since there is no gap within this time interval, here there are L_2 tasks. Note that we also have at least one complete block of L_2+1 short jobs (which are nested jobs or long-consecutive jobs), and these jobs belong to another subset of I^2. This means, that the number of jobs is at least $n \ge 2L_2 + 1$.

On the other hand, gap (idle time interval) can happen only in the incomplete block of long-consecutive jobs and within the first tasks and second tasks of the long jobs. The size of both gaps is at most L_2. Thus, the total gap is at most $\tau \le 2L_2 \le n$ and we are done.

Case C. There is no interleaved short job in the *FFD* schedule at all. Similar to the Case B, we get that the total gap is at most $\tau \le 2L_2$.

Now, we can suppose that the number of long-consecutive jobs is $n_{2,lc} > 0$, otherwise the schedule is optimal. Any long-consecutive job could not be scheduled as long-interleaved job, thus $n_{2,icn} + n_1 > L_2$, where $n_{2,icn}$ is the number of jobs in the incomplete block of the nested short jobs (see again Fig. 4.3). Since in the schedule there are further short jobs which form at least one complete block, $n \ge n_{2,icn} + n_1 + (L_2+1) > 2L_2 + 1 > \tau$. □

Lemma 4.4. *If there are neither nested nor interleaved short jobs in the input I, then $C^{FFD}_{\max}(I)/C^{*}_{\max}(I) \le 3/2$.*

Proof. If the conditions are true then all short jobs are long-consecutive and $C^{FFD}_{\max}(I) = C^{SEP}_{\max}(I)$. Since there is no interleaved job, therefore $L_2 \le n_1 - 1$. So, we get

$$\begin{aligned} C^{FFD}_{\max}(I) &= (n_1 + L_1 + 1) + (n_2 + L_2 + 1) = n + L_1 + L_2 + 2 \\ &\le n + (L_1 + n_1 + 1) = \frac{1}{2}LB1 + LB4 \le (3/2)C^{*}_{\max}(L). \end{aligned}$$

□

In the next lemma, we exclude the case when there is no long-consecutive short job at all.

Lemma 4.5. *If there is no long-consecutive short job in the input I, then $C^{FFD}_{\max}(I)/C^{*}_{\max}(I) \le 3/2$.*

Proof. We claim that $C^{FFD}_{\max}(I) \le L_1 + n + 2$. Let us consider the status when all the long jobs are just scheduled. Now, the first n_1 time slots are occupied, and the time slots are busy from L_1+1 to $L_1 + 2 + (n_1 - 1) = L_1 + n_1 + 1$.

Now let us see how the short jobs are scheduled.

First, suppose that there is no nested short job. Then all short jobs are interleaved jobs and since $L_2 < L_1$, the second tasks of the short jobs can be assigned continuously from the time $L_1 + n_1 + 1$, and so we have $C^{FFD}_{\max}(I) = (L_1 + n_1 + 1) + n_2$.

Otherwise, there are several nested jobs. Let us denote by $n_{2,ne}$ and $n_{2,i}$ the number of nested short jobs and the number of interleaved jobs, resp. Then $n_2 = n_{2,ne} + n_{2,i}$ and so $n_{2,i} < n_2$. Therefore,

$$C^{FFD}_{\max}(I) = (L_1 + n_1 + 1) + n_{2,i} < (L_1 + n_1 + 1) + n_2 = L_1 + n + 1,$$

and so

$$\frac{3}{2}C^{*}_{\max}(I) \ge LB4 + LB1/2 = (L_1 + n_1 + 1) + n \ge C^{FFD}_{\max}(I).$$

□

4.1.2. Inputs with dominant instances

From the Lemma 4.4 we know that if there are neither nested nor interleaved short jobs then $C^{FFD}_{\max}(I)/C^{*}_{\max}(I) \le \frac{3}{2}$. In this section we will investigate those cases when one of them occurs. Let

$$LB = \max\{LB1, LB3, LB4\}.$$

Lemma 4.6. *Suppose that I is an input that contains nested short jobs in its FFD schedule. Then there exists a dominant instance I' for which the FFD schedule does not contain nested jobs.*

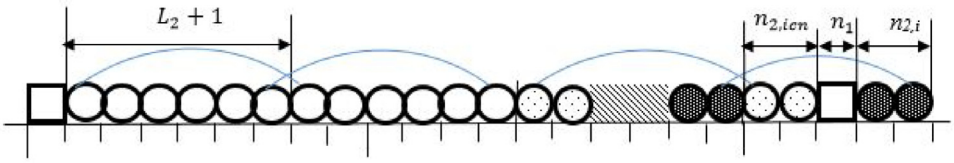

Fig. 4.3. Schedule of short interleaved jobs if first tasks are consecutive.

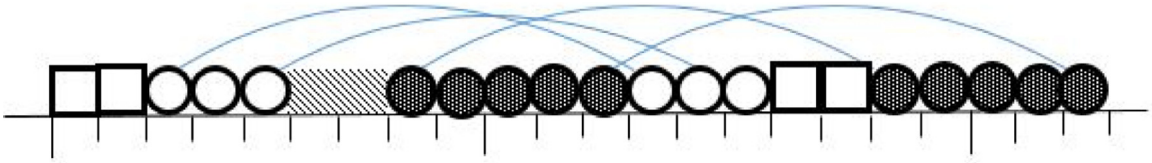

Fig. 4.4. An instance with jobs, $L_1 = 14$, $n_1 = 2$, $L_2 = 9$,$n_{2,i} = 5$, and $z = 3$.

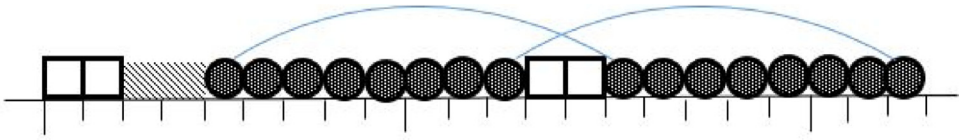

Fig. 4.5. The modified instance with $L'_1 = 11$, $n_1 = 2$, $L_2 = 9$, $n'_{2,i} = 8$.

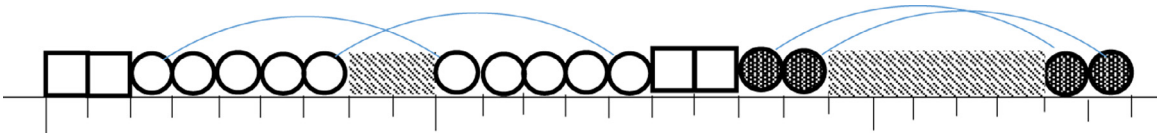

Fig. 4.6. An instance without interleaved jobs $L_1 = 13$, $n_1 = 2$, $L_2 = 6$, $n_2 = 7$,$z = 5$.

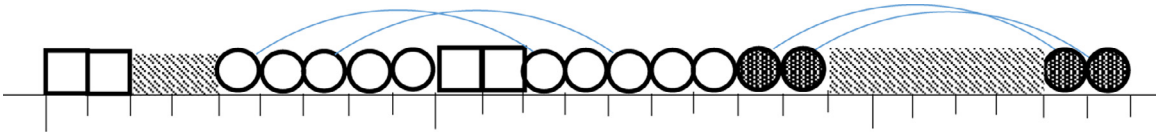

Fig. 4.7. The modified instance with $L'_1 = 8$, $n_1 = 2$, $n_{2,i} = 5$, $L_2 = 6$, $n_2 = 7$.

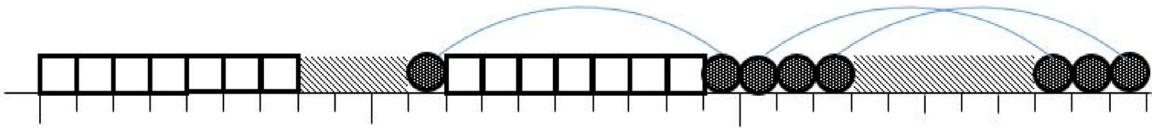

Fig. 4.8. An instance with $L_1 = 10$, $n_1 = 7$, $n_2 = 4$, $L_2 = 7$, and $z = 4$.

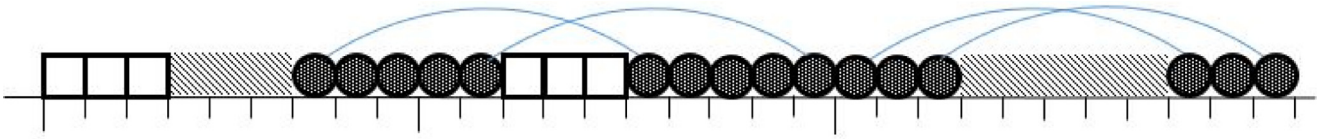

Fig. 4.9. The modified instance with $L_1 = 10$, $n'_1 = 3$, $n'_2 = 8$, and $L_2 = 7$.

Proof. Let $z > 0$, $n_{2,i}$ denote the number of nested and the number of interleaved short jobs, respectively. By Lemma 4.2, we can suppose that the *FFD* schedule of I does not contain complete blocks of long jobs, and by Lemma 4.3, there is no complete block of nested short jobs. So, the nested short jobs form an incomplete block, and their second tasks occupy z positions just before the second tasks of the long jobs (see Fig. 4.4).

Now, in I' let $L'_1 = L_1 - z$, and we consider the changes in the *FFD* schedule of our modified input. We remark that $L_2 = L'_1 - n_1$.

Claim 4.3. $LB(I') \leq LB(I)$

Proof. Since $LB1$ depends on only the number of jobs therefore $LB1(I) = LB1(I')$. In I' the sum of the gaps are smaller than in I. Therefore, $LB3(I') \leq LB3(I)$, and $LB4(I') \leq LB4(I)$. □

Case A. First, we suppose that there is at least one interleaved short job in I. Then

$$L_2 = n_1 + z + (n_{2,i} - 1). \tag{13}$$

If we schedule the items of I', then each earlier nested short job will be interleaved and the interleaved short jobs remain interleaved, i.e. $z' = 0$, and $n'_{2,i} = z + n_{2,i}$. It is easy to check that af-

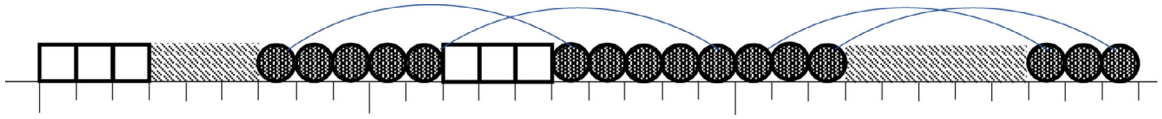

Fig. 4.10. Instance with $z = 2$ short jobs in long-consecutive block.

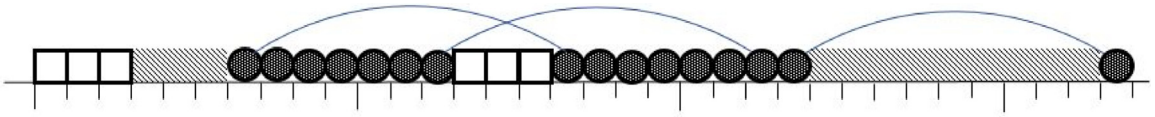

Fig. 4.11. Modified input with $z = 0$ short jobs in long-consecutive block.

ter having scheduled all long jobs and all interleaved jobs, the makespan is the same and the number of already scheduled short jobs is also the same, i.e. $n_1 + z + n_{2,i} = n_1 + z' + n'_{2,i}$. As a consequence of (13), there is no complete block in the modified schedule.

By finishing the schedule with the remained short jobs (that all will be long-consecutive jobs), the makespan of the *FFD* schedule does not change. So, applying the Claim 4.3 we get $C^{FFD}_{\max}(I')/LB(I') > C^{FFD}_{\max}(I)/LB(I)$.

Case B. Now assume that there is no interleaved short job while *FFD* schedules items of I. Let us denote the number of long-consecutive jobs by n_{lc}. It is clear that the *FFD* schedule of I' does not contain complete block from the long-consecutive short jobs. Now, after the scheduling of the long jobs, the last occupied timeslot in I' will be z unit earlier, and the number of nested jobs is $z' = 0$. The schedule will contain $n'_{2,i} = z$ new interleaved short jobs, therefore – before the *FFD* schedules the long-consecutive short jobs – the makespan of the two schedules is equal. Since the number of long-consecutive short jobs have not changed, $C^{FFD}_{\max}(I') = C^{FFD}_{\max}(I)$, and – using again Claim 4.3 we get the desired inequality. □

Lemma 4.7. *Let I be an instance for which the FFD schedule contains interleaved short jobs. Then there exists a dominant instance I' for which in the FFD schedule the first short job is an interleaved short job, and $L_2 = L_1 - n_1$.*

Proof. There are no nested jobs, therefore $L_2 \geq L_1 - n_1$. Then I' must contain at least one interleaved job. If $L_2 = L_1 - n_1$ then we are ready, $I' = I$. Let $L_2 = L_1 - n_1 + z$, where $z > 0$, integer.

Let us make a modified input I' where $n'_1 = n_1 - z$ and $n'_2 = n_2 + z$. Note that the number of jobs does not change. Now, having scheduled the long jobs and the interleaved short jobs by *FFD* the makespan is the same as earlier, and the number of already scheduled short jobs is increasing by z. So, finishing the schedule with the remained short jobs (that all will be long-consecutive jobs), the value of the *FFD* schedule remains the same.

Let us see how the lower bound changes. $LB1$ does not change, $LB3$ will be smaller, and $LB4$ decreases by z. Therefore, LB will not increase, thus for I' still holds that $C^{FFD}_{\max}(I')/LB(I') > C^{FFD}_{\max}(I)/LB(I)$. □

Now, we will investigate the structure of the long-consecutive short jobs.

Lemma 4.8. *Let I be an instance that its FFD schedule does not contain nested short jobs, contains interleaved and long-consecutive short jobs. Then there exists such a dominant instance I' which contains exactly one long-consecutive short job.*

Proof. Suppose there are at least $z + 1$ long-consecutive short jobs, where $z \leq L_2$. We derive the following instance: let $L'_1 = L_1 + z$ and $L'_2 = L_2 + z$. Note that the number of jobs does not change.

Now after scheduling the long jobs, the last occupied time slot will be z units later. Since $L'_2 > L_2$, the number of interleaved jobs increases by z, and just after scheduling the interleaved jobs, the corresponding makespan increased by $2z$. So, if it was previously t, now it is $t + 2z$. The number of the remained short jobs is decreased by z, but since L_2 is also increased by z, the increment comparing to $t + 2z$ is the same, as the increment was comparing to t in the previous input. It means that the makespan of the *FFD* schedule increases by $2z$.

Let us see how the lower bound changes. LB_1 does not change, LB_3 will be increased by z, and LB_4 also grows by z. This means that LB increases by at most z. Therefore, for I' it still holds that $C^{FFD}_{\max}(I')/LB(I') > C^{FFD}_{\max}(I)/LB(I)$, as the numerator increased by $2z$, while the denominator increased by at most z. □

4.1.3. The upper bound

In the previous subsections we have found some instances with performance ratio $\leq 3/2$, and also some instances have been analysed for which the *FFD* schedules have dominant examples with given structures. Therefore the following proposition is true.

Proposition 4.1. *If we want to find an instance I with $C^{FFD}_{\max}(I)/LB(I) > 3/2$ then we have to analyse those cases for which the* FFD *schedule (a) does not contain complete blocks (Lemma 4.2, Lemma 4.3), (b) does not contain nested short jobs (Lemma 4.4, Lemma 4.6), (c) contains interleaved short jobs (Lemma 4.4, Lemma 4.7), (d) the idle time of the short jobs is $L_2 = L_1 - n_1$ (Lemma 4.7), (e) contains exactly one long-consecutive short job (Lemma 4.5, 4.8).*

Let I be an example, which satisfies the conditions (a)-(e). From Proposition 4.1 the following simple assumptions follow.

- $n_1 < L_1 + 1$, results in a schedule with one – incomplete – block of long jobs, and after having scheduled the long jobs an idle time starts up with size $L_1 - n_1 + 1 > 0$.
- $L_2 + 2 > L_1 - n_1 + 1$, to avoid nested short jobs. We will investigate those special inputs where $L_2 = L_1 - n_1$, which results in the longest short jobs in the instance.
- $L_2 \geq n_1$, results in several interleaved jobs.
- $n_2 = L_2 - n_1 + 2$. Using this condition, *FFD* schedules as much interleaved jobs as possible and just one short job remains. This lonely short job will be a long-consecutive job.

As consequence of the assumptions above, if an instance I disposes of the characteristics above, the following facts are true.

Fact 4.1. From the conditions it follows that $L_2 = L_1 - n_1 \geq n_1$, and so

$$\frac{L_1}{n_1} \geq 2. \tag{14}$$

Fact 4.2. After scheduling the long jobs we have

$$C^{FFD}_{\max}(I^1) = L_1 + 2 + n_1 - 1 = L_1 + n_1 + 1. \tag{15}$$

Fact 4.3. After scheduling the interleaved jobs we have

$$C^{FFD}_{\max}(I) = (L_1 + n_1 + 1) + (L_2 - n_1 + 1) = L_1 + L_2 + 2. \quad (16)$$

Fact 4.4. After scheduling the last short job we have

$$C^{FFD}_{\max}(I) = (L_1 + L_2 + 2) + L_2 + 2 = L_1 + 2L_2 + 4. \quad (17)$$

We compare $C^{FFD}_{\max}(I)$ and $C^{*}_{\max}(I)$. Estimating the optimal solution, we use the lower bound $LB3$. To do that, let us express L_2 and n_2 as the variables of L_1 and n_1. Applying $L_2 = L_1 - n_1$ and $n_2 = L_2 - n_1 + 2 = L_1 - 2n_1 + 2$, we get

$$C^{FFD}_{\max}(I) = L_1 + 2L_2 + 4 = 3L_1 - 2n_1 + 4.$$

and

$$C^{*}_{\max}(I) \geq n_1 + (L_1 - 2n_1 + 2) + \frac{n_1 L_1 + (L_1 - 2n_1 + 2)(L_1 - n_1)}{n_1 + L_1 - 2n_1 + 2} + 1$$
$$= L_1 - n_1 + 3 + \frac{n_1 L_1 + (L_1 - 2n_1 + 2)(L_1 - n_1)}{L_1 - n_1 + 2}$$

Let us introduce the new variable $x = L_1/n_1$. Then we can express the previous values as $C^{FFD}_{\max}(I)/n_1 = 3x - 2 + 4/n_1$ and

$$\frac{C^{*}_{\max}(I)}{n_1} \geq x - 1 + \frac{3}{n_1} + \frac{x + (x - 2 + \frac{2}{n_1})(x - 1)}{x - 1 + \frac{2}{n_1}}.$$

For the sake of simpler notation we introduce the substitution $1/n_1 = a$. Then we get

$$\frac{C^{FFD}_{\max}(I)}{C^{*}_{\max}(I)} \leq \frac{3x - 2 + 4a}{x - 1 + 3a + \frac{x + (x-2+2a)(x-1)}{x-1+2a}}$$
$$= \frac{(3x - 2 + 4a)(x - 1 + 2a)}{(x - 1 + 3a)(x - 1 + 2a) + x + (x - 2 + 2a)(x - 1)}$$
$$= \frac{(3x - 2 + 4a)(x - 1 + 2a)}{6a^2 - 7a + 7ax + 2x^2 - 4x + 3} \quad (18)$$

and we are interested in the maximum of the right hand side. Because of the inequality (14), we can suppose that $2 \leq x$.

Lemma 4.9. *If for an instance I, $2 \leq x \leq 4$ then*

$$\frac{C^{FFD}_{\max}(I)}{C^{*}_{\max}(I)} \leq \frac{30}{19}$$

Proof. Applying the equality (18), our claim is equivalent with

$$3x^2 + (20a - 25)x + (28a^2 - 58a + 52) \geq 0 \quad (19)$$

Here, the discriminant is

$$D(a) = (20a - 25)^2 - 4 \cdot 3 \cdot (28a^2 - 58a + 52) = 64a^2 - 304a + 1.$$

Recall that $a = 1/n_1$ where n_1 is integer, thus $0 < a \leq 1$. It is easy to see that $D'(a) = 128a - 304 < 0$ in the whole $(0; 1]$ interval. Thus, $D(a)$ is decreasing. The unique solution of $D(a) = 0$ in the considered interval is $\frac{19}{8} - \frac{3}{4}\sqrt{10} \approx 0.0032918$ which means that $D(a) < 0$ for $\frac{19}{8} - \frac{3}{4}\sqrt{10} < a \leq 1$, or in equivalent form, $n_1 < 1/(\frac{19}{8} - \frac{3}{4}\sqrt{10}) \approx 303.79$.

Now let us suppose that $0 < a \leq \frac{19}{8} - \frac{3}{4}\sqrt{10}$ which means that $n_1 \geq 304$. Then, the equation (19) has two solutions, which are $x_{1,2} = \frac{25}{6} - \frac{10}{3}a \pm \frac{1}{6}\sqrt{64a^2 - 304a + 1}$. We state that both solutions are strictly bigger than 4. It suffices to see that the smaller root is bigger than 4, i.e.

$$\frac{1}{6} - \frac{10}{3}a - \frac{1}{6}\sqrt{64a^2 - 304a + 1} > 0,$$

By simple calculation, we get $24a(14a + 11) > 0$, which holds. This means that the function in the left hand side in (19) is positive, if $x \leq 4$. □

Lemma 4.10. *If for an instance I, $4 < x$, then*

$$\frac{C^{FFD}_{\max}(I)}{C^{*}_{\max}(I)} \leq \frac{(3x - 2)(x - 1)}{2x^2 - 4x + 3} \leq \frac{\sqrt{11} + 3}{4} \approx 1.579156.$$

Proof. Applying (18), we have to prove the following inequality.

$$\frac{(3x - 2 + 4a)(x - 1 + 2a)}{6a^2 - 7a + 7ax + 2x^2 - 4x + 3} \leq \frac{(3x - 2)(x - 1)}{2x^2 - 4x + 3}, \quad (20)$$

which is equivalent to the following inequality.

$$a(2x^2 + 2x - 12) + x^3 - 13x + 10 \geq 0.$$

In the left hand side $2x^2 + 2x - 12 = 2(x + 3)(x - 2) \geq 0$ because $x \geq 2$. Moreover, it is easy to see that for $x \geq 4$

$$x^3 - 13x + 10 > x^3 - 13x - 12 = (x + 3)(x - 4)(x + 1) \geq 0.$$

Now we are interested in the biggest possible value of $\frac{(3x-2)(x-1)}{2x^2-4x+3}$, considering that $x \geq 4$. To prove this, it is enough to see that

$$\frac{(3x - 2)(x - 1)}{2x^2 - 4x + 3} - \frac{\sqrt{11} + 3}{4} \leq 0. \quad (21)$$

The left hand side can be transformed as follows.

$$\frac{(3x - 2)(x - 1)}{2x^2 - 4x + 3} - \frac{\sqrt{11} + 3}{4} = \frac{3 - \sqrt{11}}{4} \frac{\left(x - \frac{1}{2}\sqrt{11} - \frac{5}{2}\right)^2}{x^2 - 2x + \frac{3}{2}}.$$

Since $3 - \sqrt{11} < 0$ and $x^2 - 2x + \frac{3}{2} > 0$ if $x > 4$, (21) is true. □

4.2. The lower bound

In this section, we give lower bounds for the performance ratio of *FFD*.

Lemma 4.11. *Let $I(n, k)$ be an instance which contains $n = n_1 + n_2 = 9k$ jobs, where $n_1 = 3k$ and $n_2 = 6k$ pieces of long and short jobs, respectively. Let us suppose that $L_1 = 12k - 2$ and $L_2 = 9k - 2$. Then*

$$\limsup_{k \to \infty} \frac{C^{FFD}_{\max}(I(n, k))}{C^{*}_{\max}(I(n, k))} = \frac{30}{19}.$$

Proof. After having scheduled long jobs there is a gap of $9k - 1$ among the two tasks of the items. Fig. 4.12 shows the status when every long job of $I(n, k)$ has been scheduled.

Since a short job needs $9k$ positions, *FFD* can not schedule any short job as nested job in the gap of long jobs. So, *FFD* will start to schedule interleaved short jobs.

FFD will schedule the first item in such a way that its second task occupies the first empty position after the last task of the long jobs. It occupies the positions $6k + 2$ and $15k$ positions for the first and the second tasks, respectively. (The tasks occupy the units $[6k + 1, 6k + 2]$ and $[15k - 1, 15k]$. It is clear that the position $15k$ was free. Similarly, the position $6k + 2$ is also free. So, the first interleaved short job can be scheduled in this position. Following this idea, *FFD* can schedule $(12k - 1) - (6k) = 6k - 1$ interleaved short jobs. The second task of the last interleaved short job occupies the position $21k - 2$.

The remaining short job needs $L_2 + 2$ units to be completed. Therefore, the makespan of the instance $I(n, k)$ produced by the algorithm *FFD* is $C^{FFD}_{\max}(I(n, k)) = 21k - 2 + L_2 + 2 = 30k - 2$. For the instance $I(n, k)$

$$\left\lceil \frac{1}{n}\left(\sum_{i=1}^{n} l_i - n(n - 1)\right)\right\rceil = k - 1 \geq 0,$$

therefore, we can apply *LB2*, and so $C^{*}(I(n, k)) \geq 19k - 1$. Therefore,

$$\limsup_{k \to \infty} \frac{C^{FFD}_{\max}(I(n, k))}{C^{*}_{\max}(I(n, k))} \leq \frac{30k - 2}{19k - 1} \quad (22)$$

To prove that the right hand side of the inequality (22) is tight, we consider the following feasible schedule of the instance $I(n, k)$. We schedule three times k pieces of long jobs and $2k$ pieces of

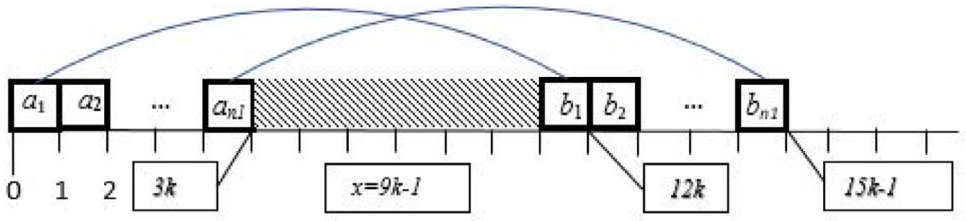

Fig. 4.12. Status when all long jobs are scheduled by *FFD*.

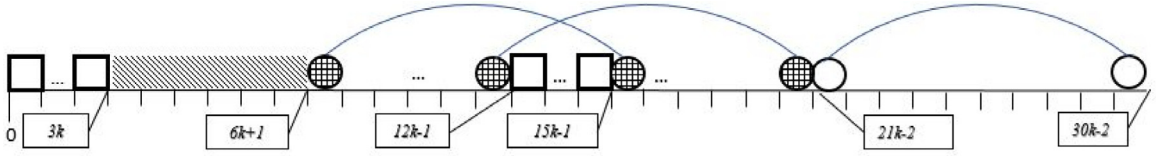

Fig. 4.13. Status when all the jobs of $I(n, k)$ are scheduled by *FFD*.

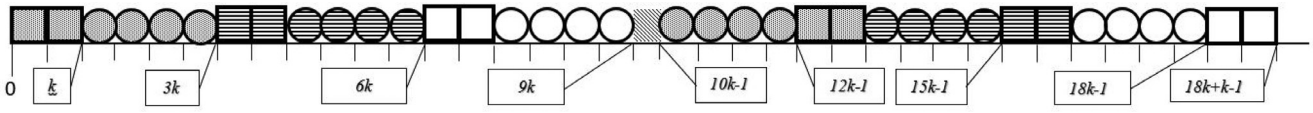

Fig. 4.14. An optimal schedule of the instance $I(n, k)$ if $k = 2$.

short jobs at the earliest possible time. We illustrate the schedule on the Fig. 4.14 for the case $k = 2$.

It is easy to check that all the positions are occupied except those ones which are between the first task of the last interleaved short job and the second task of the first nested short job. So the gap in this schedule is:

$$(12k - 2) - [(k - 1) + 2k + 3k + 3k + 2k] = k - 1.$$

So,

$$C^*(I(n, k)) \leq 2n + (k - 1) = 19k - 1. \tag{23}$$

Therefore

$$\limsup_{k\to\infty} \frac{C^{FFD}_{\max}(I(n, k))}{C^*_{\max}(I(n, k))} \geq \frac{30k - 2}{19k - 1} \tag{24}$$

Taking the inequalities (22) and (24), we get the desired result. □

Combining the results in the Lemma 4.9, Lemma 4.10, and Lemma 4.11 we get the following theorem.

Theorem 4.1. *For those problems where we only have jobs with two different idle times, the worst-case ratio of the FFD algorithm is*

$$1.57894\ldots = \frac{30}{19} \leq \rho_{FFD} \leq \frac{\sqrt{11} + 3}{4} = 1.579156\ldots,$$

and the lower bound is tight if $L_1/n_1 \leq 4$.

5. Conclusions

In this paper, we investigated a special case of CTP problem where the tasks have unit length and there are only two different gaps.

This special case is denoted by the three-field notation as

$$1|\textit{Coup-Task}, \textit{exact } l_i \in \{L_1, L_2\},\ a_i = b_i = 1|C_{\max}.$$

We considered the First Fit Decreasing (*FFD*) algorithm where the jobs are scheduled in greedy way according to their delay time: the larger the delay time, the sooner the schedule. We were looking for the absolute worst case ratio of *FFD*, and we proved that the worst-case ratio of *FFD* is in the interval $[1.57894\ldots, 1.57916\ldots)$.

Even though we considered a very special case, it is visible that the analysis is quite hard. We hope that this paper helps to understand the structure of the problem for those cases when at least three different delay times are present. Maybe the first step in this direction is to analyse the case with 3 different delay times. Our experiments show that the bound decreases. Finally, is it even possible to analyse the algorithm *FFD* for the general case of the UET problem?

Acknowledgment

The authors are grateful to the reviewers for their valuable comments and suggestions which strongly helped in improving the presentation of the paper.

References

Ageev, A. A., & Baburin, A. E. (2007). Approximation algorithms for UET scheduling problems with exact delays. *Operations Research Letters, 35*(4), 533–540.

Ageev, A. A., & Ivanov, M. (2016). Approximating coupled-task scheduling problems with equal exact delays. In Y. Kochetov (Ed.), *DOOR 2016. LNCS, vol. 9869, (pp 259–271)*. Springer, Cham.

Ageev, A. A., & Kononov, A. V. (2007). Approximation algorithms for scheduling problems with exact delays. In T. Erlebach, & C. Kaklamanis (Eds.), *WAOA 2006. LNCS, vol. 4368, (pp 1–14)*. Heidelberg: Springer.

Ahr, D., Békési, J., Galambos, G., Oswald, M., & Reinelt, G. (2004). An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research, 59*(2), 193–203.

Békési, J., Galambos, G., Oswald, M., & Reinelt, G. (2009). Improved analysis of an algorithm for the coupled task problem with UET jobs. *Operations Research Letters, 37*(2), 93–96.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Johnson, D. S. (1973). Near-Optimal bin Packing Algorithms. PhD thesis MIT, Cambridge, MA.

Khatami, M., Salehipour, A., & Cheng, T. C. E. (2020). Coupled task scheduling with exact delays: Literature review and models. *European Journal of Operational Research, 283*(1), 19–39.

Li, H., & Zhao, H. (2007). Scheduling coupled-tasks on a single machine. *IEEE symposium on computational intelligence in scheduling (pp 137–142)*. IEEE.

Orman, A. J., & Potts, C. N. (1997). On the complexity of coupled-task scheduling. *Discrete Applied Mathematics, 72*(1–2), 141–154.

Orman, A. J., Potts, C. N., Shahani, A., & Moore, A. (1996). Scheduling for a multifunction phased array radar system. *European Journal of Operational Research, 90*(1), 13-25.