

# Makespan Minimization of Multi-Slot Just-In-Time Scheduling on Single and Parallel Machines

Dariusz Dereniowski

*Department of Algorithms and System Modeling  
Gdańsk University of Technology  
Gdańsk, Poland  
email: [deren@eti.pg.gda.pl](mailto:deren@eti.pg.gda.pl)*

Wiesław Kubiak

*Faculty of Business Administration  
Memorial University of Newfoundland  
St. John's, Canada  
email: [wkubiak@mun.ca](mailto:wkubiak@mun.ca)*

## Abstract

The paper addresses the problem of multi-slot just-in-time scheduling. Unlike the existing literature on this subject, it studies a more general criteria – the minimization of schedule makespan rather than the minimization of the number of slots used by schedule. It gives an  $O(n \log^2 n)$ -time optimization algorithm for the single machine problem. For arbitrary number of  $m > 1$  identical parallel machines it presents an  $O(n \log n)$ -time optimization algorithm for the case when the processing time of each job does not exceed its due date time. For a general case on  $m > 1$  machines it proposes a polynomial time constant factor approximation algorithm.

**Keywords:** just-in-time scheduling, makespan minimization, multi-slot

## 1 Literature review and the aim of this paper

Just-in-time (JIT) scheduling problems have received a growing attention recently. The motivation to study these problems comes from both supply chains and computing where it has become much clearer understood that completing the execution of jobs exactly, or close to, their predefined due dates avoids costly disadvantages of their completing too late or too early. These include the necessity for storage or buffering, decreased efficiency of the production process, loss of customers, and costs of inventory and delay in supply chains, as well as the loss of computation accuracy and perishability of computational results in computer systems, to name just few.

Most of research in just-in-time scheduling traditionally assumes a single due date for each job and thus it has focused on reducing the costs of being early or being tardy with respect to the due date, see Józefowska [10], and Kubiak [11] for the comprehensive reviews of this area. This paper instead is interested in just-in-time multi-slot scheduling problem, where the time is divided into time slots of equal length  $L$ , and the execution of each job  $J_j$  must end exactly at its due date  $d_j \leq L$  in one of the slots, i.e. at  $iL + d_j$  for some integer  $i \geq 0$ .

Hiraishi, Levner and Vlach [9] studied a weighted version of the JIT scheduling problem with a nonnegative weight and a set-up time for each job. The objective is to find a schedule on  $m$  identical machines for which the total weight of the jobs completed just-in-time is maximized. This *single* slot problem turns out to be solvable in polynomial time. The subsequent research focused on just-in-time scheduling to minimize the number of time slots used by the schedule.

The *multi-slot* just-in-time scheduling problem was first studied by Hiraishi [8]. Chiba and Hiraishi [5] prove that for any polynomial time computable function  $\alpha(n)$  there is no polynomial time algorithm computing a just-in-time schedule using fewer slots than  $\alpha(n)$  times the optimum number of slots [5]. The problem instances created in their proof have  $m = 1$ ,  $L = 1$  and  $p_j = d_j = 1$ , thus only the set-up times are responsible

for the problem intractability. However, Chiba and Hiraishi show that there exists  $(h + 1)$ -approximation algorithm to the single machine problem, where  $hL$  is an upper bound for each set-up time.

Sung, Čepek and Hiraishi [15] show that a just-in-time single machine schedule minimizing the number of slots can be computed in  $O(n \log n)$ -time for the problem without set-up times. They also show that for  $m > 1$  parallel machines it is an NP-hard problem to find a schedule minimizing the number of slots on the machine with maximum number of slots even if the processing time of each job does not exceed the length of the slot  $L$ . For this case they show that there exists an approximation algorithm producing schedules, which are at most one slot longer than an optimal schedule. If the lengths of the jobs are arbitrary, then they show 2-approximation algorithm for the problem.

The only known polynomial-time case for  $m > 1$  machines is given by Čepek and Sung [18] for the case when the processing time of each job does not exceed its due date. Then, a schedule minimizing the number of slots on the machine with maximum number of slots can be created in  $O(n^2)$ -time. Čepek and Sung [18] also prove that the problem of minimizing the number of slots is binary NP-hard for  $m = 2$  machines and is unary (strongly) NP-hard if the number of machines is a part of the input to the problem.

This paper considers a more general objective for the multi-slot just-in-time scheduling – namely, the minimization of schedule makespan rather than the minimization of number of slots used by schedule. Clearly, all problems that are NP-hard for the minimum number of slot objective remain so for the makespan. However, the algorithms minimizing the number of slots on a single machine or minimizing the number of slots on the machine with maximum number of slots in parallel machine environment do not necessarily minimize makespan defined as the latest completion time.

This paper is organized as follows. Section 2 gives a formal statement of the problem and introduces the notation used throughout the paper. Section 4 gives an  $O(n \log^2 n)$ -time algorithm for finding a minimum makespan single machine schedule. For  $m > 1$  parallel machines we give an  $O(n \log n)$ -time optimization algorithm for instances with the processing time of each job not exceeding its due date in Section 5. Finally, in Section 6 we deal with the general parallel machine case and derive an approximation algorithm for it.

## 2 Basic definitions

In this paper we consider the problem of multi-slot parallel just-in-time scheduling on  $m \geq 1$  identical machines. The input parameters for the problem are an integer  $L > 0$  being the length of a time slot and the set of jobs denoted by  $\mathcal{J} = \{J_1, \dots, J_n\}$ . Each job  $J_j \in \mathcal{J}$  has two integer parameters: processing time  $p_j \geq 0$  and due date  $0 \leq d_j \leq L$ . The schedule is divided into slots of size  $L$  and the execution of each job should end at its due date in one of the slots, i.e. a job  $J_j \in \mathcal{J}$  must end at a time  $C_j = iL + d_j$  for some integer  $i \geq 0$ . Each machine can work on at most one job at a time. No preemptions are allowed, thus the execution of each job  $J_j$  starts at  $s_j$  satisfying  $s_j + p_j - (\text{slots}(J_j) - 1) \cdot L = d_j$ , where  $\text{slots}(J_j)$  is the number of slots a job  $J_j \in \mathcal{J}$  is intersecting in any schedule. We have  $\lfloor p_j/L \rfloor \leq \text{slots}(J_j) \leq \lceil p_j/L \rceil + 1$  for each  $J_j \in \mathcal{J}$ . Observe that this start time  $s_j$  is relative to the beginning of the slot where the job starts, thus we also use the notation  $S_j$  to denote the start time relative to the beginning of the schedule. Clearly,  $S_j = iL + s_j$  for some integer  $i \geq 0$ .

The set of machines is denoted by  $\mathcal{M} = \{M_1, \dots, M_m\}$ . If  $S$  is a schedule for  $m$  machines then the symbol  $S|_M$  is used to denote the schedule  $S$  restricted to a machine  $M \in \mathcal{M}$ . Thus,  $S|_M$  is simply a single machine schedule. Given a schedule  $S$ , let  $\text{len}(S)$  be the *length* of  $S$ , defined as the point of time where the execution of the last job ends, and let  $\text{slots}(S)$  be the maximum number of slots used by a machine, i.e.

$$\text{slots}(S) = \max\{\lceil \text{len}(S|_M)/L \rceil : M \in \mathcal{M}\}.$$

We consider two objectives in this paper the minimization of  $\text{len}(S)$  and the minimization of  $\text{slots}(S)$ . However, the primary focus will be on the minimization of  $\text{len}(S)$ .

We now introduce a notation for different types of schedules. Namely,  $S(m, X, Y)$  is a schedule for the set of jobs  $X$  on  $m$  identical machines, while  $Y$  refers to the type of the optimization criteria/type of the schedule. We consider the following:

- $Y = \text{LPT}$ : a schedule computed according to the LPT rule, see [7],
- $Y = \text{JIT}$ : an optimal schedule for the multi-slot just-in-time scheduling problem,
- $Y = C_{\max}$ : an optimal solution to the problem  $P||C_{\max}$ ,
- $Y = A$ : a schedule computed by an algorithm  $A$ .

Some other types of schedules will also be introduced later in the paper.

The only criterion for multi-slot just-in-time scheduling considered in several papers [5, 9, 16, 15, 18] thus far is the minimization of  $\text{slots}(S)$ . This paper considers mainly the minimization of  $\text{len}(S)$ . It will be clear from the context whether we optimize the length or the number of slots in the schedule, so we will use the same symbol JIT in both cases.

The following simple numerical example shows that a schedule minimizing the number of slots may have the makespan nearly twice longer than a schedule minimizing the makespan. For  $n = 3$  let  $p_1 = p_2 = p_3 = 1$  and  $d_1 = d_2 = 1, d_3 = L, L > 2$ . It is easy to check that each feasible schedule requires at least 2 slots. If we have  $C_1 = 1, C_2 = L + 1, C_3 = 2L$  in a schedule  $S$ , then  $S$  uses two slots and  $\text{len}(S) = 2L$ . However, if we have  $C_1 = 1, C_2 = L + 1, C_3 = L$  in a schedule  $S'$ , then also  $\text{slots}(S') = 2$ , yet  $\text{len}(S') = L + 1$ .

### 3 Motivating applications

Our motivation to study the multi-slot just-in-time scheduling problem arises not only from the existing literature reviewed in Section 3 but also from its generalization of a number of well-known scheduling problems. To begin with the problem of scheduling independent jobs on parallel identical machines,  $P||C_{\max}$ , is a special case of the multi-slot just-in-time scheduling; we use here the three filed scheduling notation described in detail in Błażewicz et al. [3]. This follows from an observation that one may without loss of generality assume that the processing time of each job being the input to the  $P||C_{\max}$  problem is an integer. Then, in an optimal schedule  $S$  the completion time of each job is also an integer. Thus, if one fixes the length of the slot  $L$  to be 1, then  $S$  becomes a multi-slot just-in-time schedule for this set of jobs.

Another application of the multi-slot just-in-time scheduling is an optical ring network, where the ring is divided into nodes numbered  $0, \dots, s - 1$ . The nodes can receive and transmit packets. Let there be  $n$  unit length packets to be sent  $\mathcal{P} = \{P_1, \dots, P_n\}$ . Each packet  $P_i \in \mathcal{P}$  has its source node  $s(P_i)$  and a destination node  $d(P_i)$ . The transmission occurs only in one direction in the ring, i.e. a node  $i$  can send packets only to node  $(i + 1) \bmod s$ . The time is divided into steps  $1, \dots, l$  and at each step  $x$  a node  $j$  may contain a packet  $P_i \in \mathcal{P}$  or it may be empty. For a more detailed description of this model of the optical ring network see Barth et al. [1]. Given  $\mathcal{P}$ , a *schedule* assigns to each packet  $P_i$  a step  $b(P_i)$  where the transmission of  $P_i$  begins.

We illustrate the above problem in the following example. Let the ring contain 5 nodes, as shown in Figure 1(a). The packets  $\mathcal{P} = \{P_1, \dots, P_9\}$  are defined as follows:  $s(P_2) = s(P_8) = 0, s(P_7) = 1, s(P_1) =$

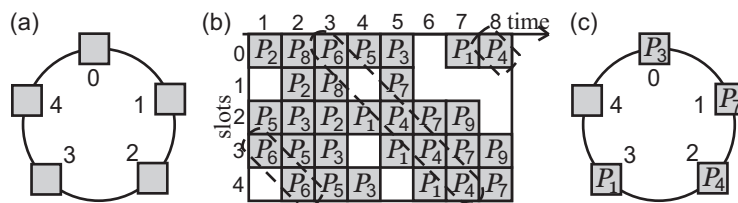


Figure 1: (a) A ring network with 5 slots; (b) a schedule for  $\mathcal{P}$ ; (c) the state of the network in the 5th step.

$s(P_3) = s(P_4) = s(P_5) = s(P_9) = 2, s(P_6) = 3$ , and  $d(P_7) = 0, d(P_1) = d(P_3) = d(P_4) = d(P_5) = d(P_6) = 1, d(P_8) = 2, d(P_2) = 3, d(P_9) = 4$ .

Figure 1(b) depicts a schedule, where  $b(P_2) = b(P_5) = b(P_6) = 1$ ,  $b(P_3) = b(P_8) = 2$ ,  $b(P_1) = 4$ ,  $b(P_4) = b(P_7) = 5$ ,  $b(P_9) = 7$ . The state of the network can be easily checked in each step (see Figure 1(c) for the state in the 5th step).

We define the multi-slot just-in-time scheduling problem for this routing problem as follows. For each packet  $P_i \in \mathcal{P}$  create a job  $J_i$  with  $d_i = d(P_i)$  and

$$p_i = \begin{cases} d(P_i) - s(P_i) & \text{if } d(P_i) > s(P_i), \\ s - s(P_i) + d(P_i) & \text{if } d(P_i) < s(P_i). \end{cases}$$

Set  $L = s$  and  $m = s$ . Note that  $s_i = s(P_i)$ .

Figure 2 gives a multi-slot just-in-time schedule corresponding to the schedule for  $\mathcal{P}$  given in Figure 1(b). In particular, the diagonals selected in Figure 1(b) correspond to the schedule on  $M_2$  in Figure 2.

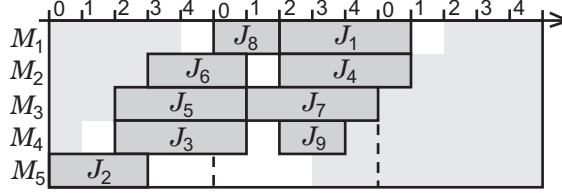


Figure 2: A schedule for  $\mathcal{J}$  corresponding to the schedule for  $\mathcal{P}$  from Fig. 1

## 4 Optimal scheduling on a single machine

This section gives an  $O(n \log^2 n)$ -time algorithm for computing a minimum makespan multi-slot just-in-time schedule for a single machine, that is  $S(1, \mathcal{J}, \text{JIT})$  in our notation. The outline of this section is as follows. Section 4.1 reduces the original problem to a problem of minimizing the total idle time on a single machine. Section 4.2 reduces the latter to the traveling salesperson problem with the Gilmore-Gomory distance matrix, and shows how the solution to the TSP results into  $S(1, \mathcal{J}, \text{JIT})$ .

### 4.1 Problem transformation

We begin with a simple observation that motivates this section. Given a single machine schedule  $S$ , let  $\text{idle}(S)$  be the total idle time in the interval  $[0, \text{len}(S)]$ . We have

$$\text{len}(S) = \text{idle}(S) + \sum_{J_j \in \mathcal{J}} p_j. \quad (1)$$

Thus, instead of minimizing the makespan of the schedule, we can minimize the total idle time because the sum  $\sum_{J_j \in \mathcal{J}} p_j$  is a constant independent of the schedule. We prove that the minimization of  $\text{idle}(S)$  can be reduced to the minimization of the traveling salesperson (TS) tour in a graph with the Gilmore-Gomory distances. The reduction requires three preliminary steps that we now describe in detail.

The first step gets rid of long jobs in a problem instance. A job is *long* if its processing time is at least  $L$ . Thus, for a job  $J_j \in \mathcal{J}$  we define  $J'_j$  to be a job with processing time  $p'_j = p_j - \lfloor \frac{p_j}{L} \rfloor \cdot L < L$  and due date  $d'_j = d_j$ . The set of all the jobs  $J'_j$  is denoted by  $\mathcal{J}'$ . The following lemma shows the correctness of the reduction first step.

**Lemma 1** *For each  $\mathcal{J}$  we have*

$$\text{len}(S(1, \mathcal{J}, \text{JIT})) = \text{len}(S(1, \mathcal{J}', \text{JIT})) + \sum_{j=1}^n \left\lfloor \frac{p_j}{L} \right\rfloor \cdot L.$$

**Proof:** Given a schedule  $S(1, \mathcal{J}', \text{JIT})$ , we obtain a schedule  $S$  for  $\mathcal{J}$  as follows. The permutation of the jobs in  $S$  remains as in  $S(1, \mathcal{J}', \text{JIT})$ , but for each  $j = 1, \dots, n$  the job  $J'_j$  gets replaced by  $J_j$ . By definition of  $\mathcal{J}'$ , we have that replacing  $J'_j$  by  $J_j$  makes the schedule  $\lfloor \frac{p_j}{L} \rfloor$  slots longer. The resulting schedule  $S$  is feasible, because each job in it still ends at its due date ( $J_j$  and all the jobs scheduled after  $J_j$  simply end exactly  $\lfloor p_j/L \rfloor$  slots later). This proves that  $\text{len}(S(1, \mathcal{J}, \text{JIT})) \leq \text{len}(S) = \text{len}(S(1, \mathcal{J}', \text{JIT})) + \sum_{j=1}^n \lfloor \frac{p_j}{L} \rfloor \cdot L$ .

Now assume that  $S(1, \mathcal{J}, \text{JIT})$  is given. Consider a job  $J_j$  with  $p_j > p'_j$ . Remove from  $S(1, \mathcal{J}, \text{JIT})$  an interval of length  $\lfloor \frac{p_j}{L} \rfloor \cdot L$  contained in the interval  $[S_j, C_j]$  during which the execution of  $J_j$  occurs. The resulting schedule remains feasible, because each job scheduled before  $J_j$  ends at the same time, while each job  $J_l$  such that  $S_l \geq S_j$  ends exactly  $\lfloor \frac{p_j}{L} \rfloor$  slots earlier than in  $S(1, \mathcal{J}, \text{JIT})$ , and thus at its due date. This proves that  $\text{len}(S(1, \mathcal{J}, \text{JIT})) \geq \text{len}(S(1, \mathcal{J}', \text{JIT})) + \sum_{j=1}^n \lfloor \frac{p_j}{L} \rfloor \cdot L$  and completes the proof of the lemma.  $\square$

By Lemma 1 we may assume w.l.o.g. that for each  $J \in \mathcal{J}$  it holds  $p_j < L$ . Let us define

$$\mathcal{J}_i = \{J \in \mathcal{J} : \text{slots}(J) = i\}, \quad i \in \mathbb{N}. \quad (2)$$

Then, by the assumption we have  $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2$ , which takes us to the second step of the reduction.

The second step shows how to construct an equivalent instance that in addition to  $p_j < L$  ensures  $0 < s_j < L$  and  $0 < d_j < L$  for each job  $J_j \in \mathcal{J}$ . To this end we modify the problem instance as follows. Let  $L' = L + 2$ . For each  $J_j \in \mathcal{J}$  let  $d'_j = d_j + 1$ . Furthermore, if  $J_j \in \mathcal{J}_2$  then  $p'_j = p_j + 2$ , and  $p'_j = p_j$  otherwise. Denote the new set of jobs by  $\mathcal{J}' = \mathcal{J}'_1 \cup \mathcal{J}'_2$ , where  $\mathcal{J}'_i = \{J'_j : J_j \in \mathcal{J}_i\}$ .

**Lemma 2** *We have the following equalities:*

$$\text{slots}(S(1, \mathcal{J}, \text{JIT})) = \text{slots}(S(1, \mathcal{J}', \text{JIT})), \quad (3)$$

$$\text{len}(S(1, \mathcal{J}, \text{JIT})) = \text{len}(S(1, \mathcal{J}', \text{JIT})) - 2\text{slots}(S(1, \mathcal{J}', \text{JIT})) + 1. \quad (4)$$

**Proof:** Consider the following one-to-one relation between the feasible schedules of  $\mathcal{J}$  and the feasible schedules of  $\mathcal{J}'$ :  $J_j \in \mathcal{J}$  is done in the interval  $[S_j = iL + s_j, C_j = \ell L + d_j]$  if and only if  $J'_j \in \mathcal{J}'$  is done in the interval  $[S'_j = iL' + s'_j, C'_j = \ell L' + d'_j]$ , where

$$\ell = \begin{cases} i & \text{if } J_j \in \mathcal{J}_1, \\ i + 1 & \text{if } J_j \in \mathcal{J}_2. \end{cases}$$

Observe that the reduction second step ensures that  $s'_j = s_j + 1$  and  $d'_j = d_j + 1$ . Therefore, (3) is obvious. Furthermore, if  $J_j$  is the last job in the schedule  $S(1, \mathcal{J}, \text{JIT})$  then

$$\text{len}(S(1, \mathcal{J}, \text{JIT})) = (\text{slots}(S(1, \mathcal{J}, \text{JIT})) - 1)L + d_j$$

and

$$\begin{aligned} \text{len}(S(1, \mathcal{J}', \text{JIT})) &= (\text{slots}(S(1, \mathcal{J}, \text{JIT})) - 1)L' + d'_j \\ &= (\text{slots}(S(1, \mathcal{J}, \text{JIT})) - 1)L + 2(\text{slots}(S(1, \mathcal{J}, \text{JIT})) - 1) + d_j + 1 \\ &= \text{len}(S(1, \mathcal{J}, \text{JIT})) + 2\text{slots}(S(1, \mathcal{J}, \text{JIT})) - 1, \end{aligned}$$

which proves (4).  $\square$

By Lemmas 1 and 2 we may assume w.l.o.g. that for each  $J \in \mathcal{J}$  we have  $0 < s_j < L$ ,  $0 < d_j < L$  and  $p_j < L$ .

The third step of the reduction depends on the number of idle intervals  $(iL, iL+1)$ ,  $i \in \{0, \dots, \text{slots}(S(1, \mathcal{J}, \text{JIT})) - 1\}$  in  $S(1, \mathcal{J}, \text{JIT})$ . This number equals

$$|\mathcal{J}_d| = \left\lceil \frac{r}{L} \right\rceil - |\mathcal{J}_2|, \quad (5)$$

where  $r = \text{len}(S(1, \mathcal{J}, \text{JIT}))$ , and it is the same for all optimal schedules. We show later how to calculate it, however, for the time being we assume that this number is given. Under this assumption, for each idle interval in  $S(1, \mathcal{J}, \text{JIT})$ , we introduce a dummy job  $J$  with processing time 1 and due date 1. Denote the set of all such dummy jobs by  $\mathcal{J}_d$ . By this definition, we have

$$\text{len}(S(1, \mathcal{J}, \text{JIT})) = \text{len}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})). \quad (6)$$

and

$$\text{len}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})) = \text{idle}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})) + \sum_{J_j \in \mathcal{J} \cup \mathcal{J}_d} p_j \quad (7)$$

$$= \text{idle}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})) + |\mathcal{J}_d| + \sum_{J_j \in \mathcal{J}} p_j. \quad (8)$$

Thus, by (1) and (6)-(8) we have

$$\text{idle}(S(1, \mathcal{J}, \text{JIT})) = \text{idle}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})) + |\mathcal{J}_d|. \quad (9)$$

We summarize these reductions with the following theorem.

**Theorem 1** *If  $\mathcal{J}$  is a problem instance satisfying  $p_j < L$  and  $0 < s_j, d_j < L$  for each  $J_j \in \mathcal{J}$  and  $\mathcal{J}_d$  is a set of  $\lceil \frac{r}{L} \rceil - |\mathcal{J}_2|$  dummy jobs with processing time 1 and due time 1, then*

$$\text{idle}(S(1, \mathcal{J}, \text{JIT})) = \text{idle}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})) + \lceil \frac{r}{L} \rceil - |\mathcal{J}_2|. \quad (10)$$

Moreover, for each pair of consecutive jobs  $i$  followed by  $j$  in an optimal multi-slot just-in-time schedule  $S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})$  we have

$$0 \leq S_j - C_i < L, \quad (11)$$

and thus, we have the following upper bound on the minimum idle time:

$$\text{idle}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})) < (|\mathcal{J}| + |\mathcal{J}_d|)L. \quad (12)$$

□

We close this section by illustrating the reduction in Figure 3, where  $\mathcal{J} = \{J_1, J_2, J_3, J_4\}$ ,  $L = 4$ ,  $p_1 = 3, d_1 = 3$ ,  $p_2 = 14, d_2 = 1$ ,  $p_3 = 7, d_3 = 4$ ,  $p_4 = 6, d_4 = 3$ . Figure 3(a) depicts a schedule for the corresponding problem with  $L' = L + 2$  and short jobs. In this schedule  $|\mathcal{J}_d| = 2$  – the jobs in  $\mathcal{J}_d$  are represented by the shaded rectangles. Figure 3(b) is the corresponding schedule with removed dummy jobs and with the initial value of  $L$ . When each job is replaced with the original one, then we get the schedule in Figure 3(c).

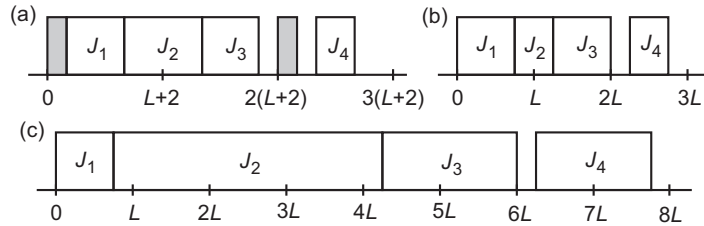


Figure 3: Problem reductions: (a) the schedule for jobs satisfying  $p_j < L$ ,  $0 < s_j < L$ ,  $0 < d_j < L$ , dummy jobs and  $L' = L + 2$ ; (b) the corresponding schedule without dummy jobs and with the initial value of  $L$ ; (c) the schedule for the initial instance

## 4.2 Transformation to TSP

We are now ready to reduce the problem of minimizing the total idle time to the traveling salesman problem (TSP) with a Gilmore-Gomory distance matrix [6] where the distances between the cities  $i$  and  $j$  are of the form

$$c_{i,j} = \begin{cases} \int_{a_i}^{b_j} f(x)dx & \text{if } a_i \leq b_j, \\ \int_{b_j}^{a_i} g(x)dx & \text{if } a_i > b_j, \end{cases}$$

with the parameters  $a_i, b_i$  being fixed for each city  $i$  and  $f, g$  being any integrable functions satisfying the inequality  $f(x) + g(x) \geq 0$  for all  $x$  [6].

A job  $J_j$  corresponds to the city  $j$ . We set  $a_j = d_j$  for  $J_j \in \mathcal{J} \cup \mathcal{J}_d$  and

$$b_j = \begin{cases} s_j & \text{if } J_j \in \mathcal{J}, \\ L & \text{if } J_j \in \mathcal{J}_d. \end{cases}$$

The  $f$  is a constant function equal to 1, while the  $g$  is a constant function equal to  $(|\mathcal{J}| + |\mathcal{J}_d| + 1)L$ . Thus, the distance matrix is as follows

$$c_{i,j} = \begin{cases} b_j - d_i & \text{if } d_i \leq s_j, \\ (|\mathcal{J}| + |\mathcal{J}_d| + 1)L(d_i - s_j) & \text{if } d_i > s_j. \end{cases} \quad (13)$$

Given the value of  $r$ , we now more formally describe the procedure which finds a multi-slot just-in-time schedule of length at most  $r$  if one exists. This procedure, which we call **GG** takes  $\mathcal{J}$ , satisfying the conditions  $p_j < L$  and  $0 < s_j, d_j < L$  for each  $J_j \in \mathcal{J}$  of Theorem 1, and an integer  $r$  as an input. The **GG** proceeds as follows:

1. let  $\mathcal{J}_d$  contain  $\lceil \frac{r}{L} \rceil - |\mathcal{J}_2|$  dummy jobs;
2. let  $J_0$  be a job with  $p_0 = L - (r \bmod L)$ ,  $d_0 = 0$ ;
3. use the Gilmore-Gomory algorithm to find an optimal TS tour for  $\mathcal{J} \cup \mathcal{J}_d \cup \{J_0\}$ ;
4. if the length of the TS tour is at least  $(|\mathcal{J}| + |\mathcal{J}_d| + 1)L$  then exit – no schedule of length at most  $r$  exists;
5. otherwise convert the TS tour to the just-in-time schedule  $S'$  for  $\mathcal{J}$  by ordering the jobs in  $\mathcal{J} \cup \mathcal{J}_d \cup \{J_0\}$  according to the order in which the cities are traversed in the TS tour. The last city to visit corresponds to  $J_0$ ;
6. given  $S'$ , compute the schedule  $S$  for the original instance: the ordering of jobs remains unchanged and jobs in  $\mathcal{J}_d \cup \{J_0\}$  are removed;
7. return  $S$  and exit;

We have the following lemma.

**Lemma 3** *Given  $\mathcal{J}$  and  $r$ ,  $\text{len}(S(1, \mathcal{J}, \text{JIT})) \leq r$  if and only if the procedure **GG** produces a schedule  $S'$  in Step 5. The schedule is not longer than  $r$ , that is  $\text{len}(S') \leq r$ .*

**Proof:** Assume that the algorithm **GG** exits in Step 4 and  $\text{len}(S(1, \mathcal{J}, \text{JIT})) \leq r$ . Then, (12) in Theorem 1 is in contradiction with the exit condition in Step 4. The latter stipulates that the shortest TS tour for  $\mathcal{J} \cup \mathcal{J}_d \cup \{J_0\}$  is at least  $(|\mathcal{J}| + |\mathcal{J}_d| + 1)L$  implying  $\text{idle}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT})) \geq (|\mathcal{J}| + |\mathcal{J}_d|)L$ . This contradiction proves that if the algorithm **GG** exits in Step 4, then  $\text{len}(S(1, \mathcal{J}, \text{JIT})) > r$ .

Now, assume that **GG** returns a schedule in Step 7. By Theorem 1 we obtain that the problems of minimizing  $\text{idle}(S(1, \mathcal{J}, \text{JIT}))$  and  $\text{idle}(S(1, \mathcal{J} \cup \mathcal{J}_d, \text{JIT}))$  are equivalent because  $\lceil \frac{r}{L} \rceil - |\mathcal{J}_2|$  is independent of the permutation of jobs. The algorithm of Gilmore and Gomory [6] finds an optimal TS tour for the cities

corresponding to  $\mathcal{J} \cup \mathcal{J}_d \cup \{J_0\}$ . This tour is shorter than  $(|\mathcal{J}| + |\mathcal{J}_d| + 1)L$  by (12) in Theorem 1. For a schedule  $S'$  produced in Step 5 we have  $\text{len}(S') \leq \text{idle}(S') + |\mathcal{J}_d| + \sum_{i=0}^n p_i$ . By (1), (5), Theorem 1 and the fact that  $J_0$  is the last job in the schedule  $S'$  we have that  $\text{len}(S) \leq r$ .  $\square$

We now describe how to find the right value of  $r$ . Let us denote by  $s = \lceil (\sum_{i=1}^n p_i)/L \rceil$  the smallest possible number of slots required in an optimal schedule.

1. modify the instance as described in Section 4.1, so that it fulfills the conditions  $p_j < L$  and  $0 < s_j, d_j < L$  for each  $J_j \in \mathcal{J}$ ;
2. using a binary search for  $r \in \{sL, (s+1)L, \dots, 2nL\}$ , call  $S' = \text{GG}(\mathcal{J}, r)$ ;
3. assume that  $r = xL$  is the smallest value for which a schedule has been found in Step 2. Furthermore, let the jobs be sorted in ascending order of their due dates, i.e.  $d_i \leq d_j$  for  $1 \leq i \leq j \leq n$ . Using a binary search for  $r \in \{(x-1)L + d_1, (x-1)L + d_2, \dots, (x-1)L + d_n\}$ , call  $S' = \text{GG}(\mathcal{J}, r)$ ;
4. return the schedule  $S$ , corresponding to the smallest  $r$  found in Step 3;

First, observe that a schedule shorter than  $sL$  cannot exist. On the other hand,  $2n$  is an upper bound for the number of required slots since  $p_j < L$  for each job  $J_j \in \mathcal{J}$ . Thus, in Step 2 the algorithm finds a schedule  $S'$  requiring the minimum number of  $x$  slots. This implies that  $\text{slots}(S') = \text{slots}(S(1, \mathcal{J}, \text{JIT}))$ . However, there is no guarantee that this is the shortest schedule. Clearly,  $\text{len}(S(1, \mathcal{J}, \text{JIT})) = (x-1)L + d_j$  for some  $j \in \{1, \dots, n\}$ , because each schedule has to end at one of the due dates. So, in Step 3 the procedure finds the optimal value of  $r$ . By Theorem 1 we know that  $S'$  is of minimum length for  $\mathcal{J}$ .

The original paper of Gilmore and Gomory [6] gave an  $O(n^2)$ -time algorithm solving the TSP to optimality in Step 2 of GG. This complexity has been subsequently improved to  $O(n \log n)$ , see [13, 17]. Finally, due to the condition  $p_j < L$  for each  $j = 1, \dots, n$  we obtain the  $O(\log n)$ -time complexity of the binary search in Steps 2 and 3 of the search for the optimal  $r$ . Consequently, we have

**Theorem 2** *An optimal single machine multi-slot just-in-time schedule for an arbitrary set of jobs  $\mathcal{J}$  can be computed in  $O(n \log^2 n)$ -time.*  $\square$

## 5 Scheduling jobs with $p_j \leq d_j$ on $m > 1$ machines

The problem of multi-slot just-in-time scheduling with jobs  $J_j$  satisfying  $p_j \leq d_j$  has been considered in [18], where an  $O(n^2)$ -time optimization algorithm has been given. In this section we improve this result in two ways. First, we consider a more general optimization criteria than the one in [18], i.e. the makespan instead of the number of slots used in a schedule. Second, we improve the time complexity to  $O(n \log n)$ .

We use as a tool a procedure for optimal coloring of (open) intervals  $\mathcal{I} = \{I_1, \dots, I_n\}$ . A function  $f: \mathcal{I} \rightarrow \{1, \dots, p\}$  is a  $p$ -coloring of  $\mathcal{I}$  if for each two intersecting intervals  $I_i, I_j$  it holds  $f(I_i) \neq f(I_j)$ . A  $p$ -coloring is *optimal* if there exists no  $p'$ -coloring for each  $p' < p$ . For each interval  $I_i$  define  $l(I_i)$  and  $r(I_i)$  to be respectively its *left* and *right* endpoint, that is  $I_i = (l(I_i), r(I_i))$ . Let us introduce two orderings on intervals  $\mathcal{I}$ :  $I_i \leq_r I_j$  if  $r(I_i) \leq r(I_j)$ , and  $I_i \leq_l I_j$  if  $r(I_i) \leq l(I_j)$ . Assume in the following that the intervals are sorted according to increasing values of their right endpoints.

**Definition 1** A coloring  $f$  is a *best fit coloring* if the intervals have been colored sequentially according to the order  $\leq_r$  and if for an interval  $I_i$  it holds  $\{I_j : I_j \leq_l I_i\} \neq \emptyset$  then  $I_i$  gets color assigned to an interval  $I_j$ ,  $I_j \leq_l I_i$ , satisfying  $r(I_j) = \max\{r(I_{j'}) : I_{j'} \leq_l I_i\}$ .

**Theorem 3** ([4]) *Given  $n$  sorted intervals and an integer  $p$ . There exists an  $O(n+p)$ -time algorithm for finding a best-fit  $p$ -coloring for  $\mathcal{I}$ .*  $\square$

**Theorem 4** ([4]) *Each best fit coloring of  $\mathcal{I}$  is optimal.*  $\square$

We now prove that a best fit coloring has a special property which will be crucial in our reduction of the just-in-time scheduling problem to the problem of coloring the intervals. In the following we will use the following notation. Given a  $p$ -coloring  $f$  of  $\mathcal{I}$ ,  $f_i = \max\{r(I_j) : f(I_j) = i\}$ . If  $i < j$  are two colors used by  $f$  then we may assume that  $f_i \geq f_j$ , because otherwise we may exchange the colors in such a way that each interval colored with  $i$  gets color  $j$  while each interval previously colored by  $j$  gets color  $i$ . So, we assume that  $f_i \geq f_{i+1}$  for each  $i = 1, \dots, p-1$ . The property is given in the following lemma.

**Lemma 4** *If  $f$  is a best fit  $p$ -coloring of  $\mathcal{I}$  and  $g$  is any  $p'$ -coloring of  $\mathcal{I}$  then for each  $i = 1, \dots, p$  we have  $f_i \leq g_i$ .*

**Proof:** Assume that the intervals  $\mathcal{I}$  are sorted according to the ascending order of their right endpoints and let  $\mathcal{I}_i = \{I_1, \dots, I_i\}$  for  $i = 1, \dots, n$ . By Theorem 4 we have that  $p' \geq p$ .

The proof is by induction on  $n$ . The lemma clearly holds for  $\mathcal{I}_1$ . Assuming that it holds for any  $n-1$  interval instance  $\mathcal{I}_{n-1}$  we show that it also holds for  $\mathcal{I}_n$ , an instance with  $n$  intervals. Let  $f'$  and  $g'$  be the colorings  $f$  and  $g$ , respectively, restricted to  $\mathcal{I}_{n-1}$ . Since  $I_n$  is the interval colored last by the best fit algorithm, then  $f'$  is a best fit coloring for  $\mathcal{I}_{n-1}$ . This in particular implies the optimality of  $f'$  for  $\mathcal{I}_{n-1}$ . We may without loss of generality assume that the colors are sorted in such a way that  $f'_i \geq f'_{i+1}$  and  $g'_i \geq g'_{i+1}$  for each  $i = 1, \dots, p-1$ . If  $f'$  uses  $p-1$  colors then  $I_n$  intersects  $p-1$  intervals with right endpoints  $f'_1, \dots, f'_{p-1}$ . So, the lemma holds for  $n$  in this case. We assume that  $f'$  uses  $p$  colors in the remaining part of the proof.

By the induction hypothesis  $f'_i \leq g'_i$  for each  $i = 1, \dots, p$ . Denote  $f(I_n) = c_f$  and  $g(I_n) = c_g$  to be the colors assigned to  $I_n$  by  $f$  and  $g$  respectively. If  $c_g = c_f$  then the lemma clearly holds. Thus, since  $f$  is a best fit coloring, then it remains to consider the case  $c_f < c_g$ . Suppose  $c_g \leq p$ . Since  $f_{c_f} \geq f'_i$  and  $g_{c_g} \geq g'_i$  for each  $i = 1, \dots, p$  we have

$$f_{c_f} \geq f'_1 \geq \dots \geq f'_{c_f-1} \geq f'_{c_f+1} \geq \dots \geq f'_{c_g} \geq f'_{c_g+1} \geq \dots \geq f'_p, \quad (14)$$

$$g_{c_g} \geq g'_1 \geq \dots \geq g'_{c_f-1} \geq g'_{c_f} \geq \dots \geq g'_{c_g-1} \geq g'_{c_g+1} \geq \dots \geq g'_p. \quad (15)$$

By the induction hypothesis we have

$$f_i = f'_i \leq g'_i = g_i, \quad \text{for } i \in \{1, \dots, c_f-1\} \cup \{c_g+1, \dots, p\},$$

and

$$f_{i+1} = f'_{i+1} \leq g'_{i+1} \leq g_i, \quad i = c_f, \dots, c_g-1.$$

Moreover,

$$f_{c_f} = g_{c_g}.$$

Thus, the lemma holds for  $n$ . Finally, note that if  $g$  introduces a new color  $c_g > p$  for  $I_n$ , then the sequence in (15) becomes

$$g_{c_g} \geq g'_1 \geq \dots \geq g'_{c_f-1} \geq g'_{c_f} \geq \dots \geq g'_{c_g} \geq g'_{c_g-1} \geq g'_{c_g+1} \geq \dots \geq g'_p. \quad (16)$$

thus the lemma holds for  $n$  again. This ends the induction and proves the lemma.  $\square$

Given a set of jobs  $\mathcal{J}$  such that  $p_j \leq d_j$  for each  $J_j \in \mathcal{J}$ , for each job  $J_j$  we create the corresponding interval  $I_j = (l(I_j), r(I_j)) = (s_j, d_j)$ . The following lemma shows the connection between the multi-slot just-in-time schedules and the colorings of intervals.

**Lemma 5** *For a multi-slot just-in-time schedule  $S$  on  $m$  machines, there exists a  $p'$ -coloring  $g$  of  $\mathcal{I}$ , where  $p'$  is the total number of slots occupied by jobs in  $S$ , such that  $g_i \geq g_{i+1}$  for  $i = 1, \dots, p'-1$ , and  $\text{len}(S) \geq (\lceil p'/m \rceil - 1)L + g_{(\lceil p'/m \rceil - 1)m+1}$ .*

**Proof:** Let now  $S$  be a multi-slot just-in-time schedule. Define  $g$  in such a way that  $g(I_j) = c$  if the job  $J_j$  is scheduled in the  $i$ th slot on machine  $M_t$  in  $S$ , where  $i = \lceil c/m \rceil$  and  $t = ((c-1) \bmod m) + 1$ . Since only the jobs scheduled in the same slot get the same color and no job intersects two or more slots, the  $g$  is a coloring. The number of colors  $p'$  used by  $g$  equals the total number of slots occupied by jobs in  $S$ . Without loss of generality  $g_i \geq g_{i+1}$  for  $i = 1, \dots, p' - 1$ . Thus, it can be readily checked that  $\text{len}(S) \geq (\lceil p'/m \rceil - 1)L + g_{(\lceil p'/m \rceil - 1)m + 1}$ . This proves the lemma.  $\square$

We are now ready to describe our Best-Fit Scheduling algorithm (B-fS for short):

1. let  $\mathcal{I} = \{I_1, \dots, I_n\}$  be the set of intervals corresponding to  $\mathcal{J}$ ;
2. sort the intervals in  $\mathcal{I}$  according to the ascending order of their right endpoints;
3. call the procedure from Theorem 3 to find a best-fit  $p$ -coloring  $f$  of  $\mathcal{I}$ ;
4. sort the colors so that  $f_i \geq f_{i+1}$  for each  $i = 1, \dots, p - 1$ ;
5. schedule the jobs corresponding to intervals with color  $i$ ,  $i = 1, \dots, p$ , in  $j$ th slot on machine  $M_l$ , where  $j = \lceil i/m \rceil$  and  $l = ((i-1) \bmod m) + 1$ ; return schedule  $S$  created in this step;

This process is illustrated in Figure 4. Assume that  $J_1, \dots, J_9$  are given. Their processing times, due dates and the corresponding intervals  $\mathcal{I}$  are shown in Figure 4(a). The integers at the right endpoints of the intervals are the colors assigned by a best fit algorithm. For this instance 5 colors are sufficient, and are already sorted to satisfy  $f_1 \geq f_2 \geq \dots \geq f_5$ . Figure 4(b) depicts the corresponding schedule. The first slot

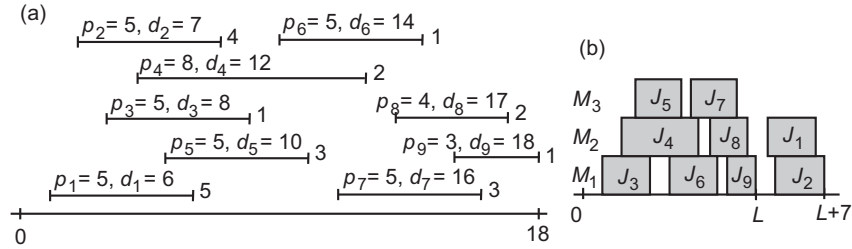


Figure 4: (a) The set of intervals  $\mathcal{I}$  corresponding to  $\mathcal{J}$  and their optimal coloring  $f$ , (b) the minimum length schedule corresponding to  $f$

on each  $M_1, M_2$  and  $M_3$  is occupied by jobs corresponding to intervals labeled by 1, 2 and 3, respectively. The jobs corresponding to intervals labeled by 4 and 5 are scheduled, respectively, in the second slot on  $M_1$  and  $M_2$ . The makespan of the final schedule is  $L + f_4 = L + r(I_2) = L + 7 = 25$ . In the following we prove how the makespan of a schedule depends on an initial coloring used to create the schedule.

**Lemma 6** For a  $p$ -coloring  $f$  of  $\mathcal{I}$ , with  $f_i \geq f_{i+1}$  for  $i = 1, \dots, p - 1$ , there exists a schedule  $S$  of length  $\text{len}(S) = (l-1)L + f_{(l-1)m+1}$  on  $m$  machines, where  $l = \lceil p/m \rceil$ .

**Proof:** Consider a schedule  $S$  created in Step 5 of the B-fS algorithm from a  $p$ -coloring  $f$ . The  $S$  is feasible, because each slot contains jobs corresponding to non-intersecting intervals of the same color under  $f$ , and each job starts and completes in the same time interval. Moreover, since  $f$  is  $p$ -coloring, then  $l = \text{slots}(S) = \lceil p/m \rceil$ . By the construction of  $S$ , the  $j$ th slot on machines  $M_1, \dots, M_m$  contains jobs corresponding to intervals colored by  $(j-1)m+1, \dots, jm$  respectively (if  $lm > p$ , then the  $l$ th slot on each machine  $M_{(p \bmod m)+1}, \dots, M_m$  remains empty). The last job in a slot with the jobs corresponding to intervals of color  $i$  ends at its due date equal  $f_i$ . Since  $f_i \geq f_{i+1}$  and  $\text{len}(S) = \max\{\text{len}(S|_{M_i}) : i = 1, \dots, m\}$  we obtain that  $\text{len}(S) = \text{len}(S|_{M_1}) = (l-1)L + f_i$ , where  $i = (l-1)m + 1$ .  $\square$

**Lemma 7** *The B-fS algorithm finds a minimum makespan multi-slot just-in-time schedule for  $\mathcal{J}$ . The running time of B-fS is  $O(n \log n)$ .*

**Proof:** Consider a multi-slot just-in-time schedule  $S$  for  $\mathcal{J}$ . Let  $g$  be the coloring for  $\mathcal{I}$  built for  $S$  as in the proof of Lemma 5. Let  $f$  be a best-fit coloring created in Step 3 of B-fS algorithm. By Lemmas 4, 5 and 6 we have that  $\text{len}(S_f) \leq \text{len}(S)$  for the schedules  $S_f$  and  $S$  corresponding to  $f$  and  $g$ , respectively. Thus  $S_f$  is optimal.

Finally, Steps 1 and 5 have linear running time. By Theorem 3, Step 3 also runs in  $O(n)$ -time. Steps 2 and 4 are the most time consuming steps since they require sorting of the  $n$  intervals and the colors, respectively. So, the overall running time of Bf-S algorithm is  $O(n \log n)$ .  $\square$

An important feature of the optimal solutions obtained by the Bf-S algorithm is their lexicographical optimality defined as follows. A single machine schedule is *lexicographically minimal* if its makespan is minimum. A schedule  $S$  on  $m > 1$  machines with  $\text{len}(S|_{M_i}) \geq \text{len}(S|_{M_{i+1}})$ ,  $i = 1, \dots, m-1$ , is called *lexicographically minimal* if its makespan is minimum, and recursively if we remove from  $S$  the jobs scheduled on  $M_1$ , then the remaining schedule remains lexicographically minimal. The schedules produced by the Bf-S procedure are lexicographically minimal, which implies the following theorem, which follows directly from Lemmas 4 and 5.

**Theorem 5** *Let  $S$  be a schedule returned by algorithm Bf-S and  $S'$  be any schedule for  $\mathcal{J}$ . Moreover, let  $\pi$  be such a permutation of machines that*

$$\text{len}(S')|_{M_{\pi(i)}} \geq \text{len}(S')|_{M_{\pi(i+1)}}$$

for  $i = 1, \dots, m-1$ . Then,

$$\text{len}(S|_{M_i}) \leq \text{len}(S'|_{M_{\pi(i)}})$$

for each  $i = 1, \dots, m$ .  $\square$

More on the lexicographically minimal (or maximal) solutions for other combinatorial optimization problems can be found in Megiddo [14], and Kumar and Kleinberg [12].

Another implication of the approach based on coloring of interval is the existence of a polynomial time algorithm for computing a schedule using at most two slots on each machine or deciding that any schedule must use more than two slots on some machines.

We now sketch this algorithm. Since we are interested in finding schedule  $S$  with  $\text{slots}(S) \leq 2$ , then any job with  $p_j > d_j$  must start in the first slot on a machine and finish in the second one on this machine. Without loss of generality we assume that these jobs are  $J_1, \dots, J_x$ , where  $0 \leq x \leq m$ . Clearly, if  $x = 0$ , then the Bf-S can be used to test whether  $\text{slots}(S) \leq 2$ . Thus, we may assume positive  $x$  from now on. Let us split any job  $J_j$  with  $p_j > d_j$  into two jobs  $J'_j$  and  $J''_j$  satisfying  $p'_j = p_j - d_j$ ,  $d'_j = L$  and  $p''_j = d_j$ ,  $d_j = d_j$  respectively. The job  $J'_j$  is the part of  $J_j$  executed in the first slot, while  $J''_j$  is the part of  $J_j$  executed in the second slot on some machine. We can then obtain a  $p$ -coloring,  $p \leq 2m$ , for the original set of jobs from the coloring  $c$  for the new set of jobs as long as  $m < c(J''_j) = c(J'_j) + m \leq 2m$ . To get the desired coloring, we pre-color the jobs  $J'_1, \dots, J'_x$  with colors  $1, \dots, x$  respectively, and the jobs  $J''_1, \dots, J''_x$  with colors  $m+1, \dots, m+x$  respectively, and use a polynomial time algorithm for coloring the vertices of an interval graph, where some of the vertices have been pre-colored and each pre-color has been used at most once, given by Biró, Hujter, and Tuza [2] to check if there is a coloring with at most  $2m$  colors.

## 6 An approximation algorithm for parallel machines

We now turn to a general multi-slot just-in-time scheduling on  $m$  parallel identical machines. As pointed out in Section 3, this problem is a generalization of the problem  $P||C_{\max}$  and thus it is NP-hard in the strong sense, and it remains NP-hard even for a fixed  $m$ . Therefore, we focus on approximation algorithms for the general multi-slot just-in-time scheduling on  $m$  parallel identical machines in this section. First, we observe

that a 3-approximate solution can be readily obtained from an optimal solution for a single machine problem shown to be computable in time  $O(n \log^2 n)$  in Section 4. We simply note that

$$\lceil \text{slots}(S(1, \mathcal{J}, \text{JIT}))/m \rceil \leq \text{slots}(S(m, \mathcal{J}, \text{JIT})). \quad (17)$$

Thus, we can find a solution  $S'$  to the  $m$  machine problem by slicing a schedule  $S(1, \mathcal{J}, \text{JIT})$  in such a way that all the jobs starting in the slots

$$i \cdot \lceil \text{slots}(S(1, \mathcal{J}, \text{JIT}))/m \rceil + 1, \dots, (i+1) \cdot \lceil \text{slots}(S(1, \mathcal{J}, \text{JIT}))/m \rceil$$

are scheduled on machine  $M_{i+1}$  in  $S'$ ,  $i = 0, \dots, m-1$ . Then, by (17)

$$\begin{aligned} \text{len}(S'|_{M_i}) &\leq \lceil \text{slots}(S(1, \mathcal{J}, \text{JIT}))/m \rceil \cdot L + \max\{p_j : J_j \in \mathcal{J}\} \\ &\leq \text{slots}(S(m, \mathcal{J}, \text{JIT})) \cdot L + \text{len}(S(m, \mathcal{J}, \text{JIT})) \leq 2\text{len}(S(m, \mathcal{J}, \text{JIT})) + L \end{aligned}$$

for each  $i = 1, \dots, m$ . Finally, we observe that if  $\text{len}(S(m, \mathcal{J}, \text{JIT})) \leq L$ , then an optimal solution can be found in time  $O(n \log n)$  as shown in Section 5. Thus, we can limit ourselves to the case when the length of  $S(m, \mathcal{J}, \text{JIT}) > L$ . This, however, leads to the required approximation ratio of 3. Consequently, a 3-approximate solution can be obtained in time  $O(n \log^2 n)$ , and we assume in the remainder of this section that this 3-approximate algorithm can always be run independently of other approximation algorithms and a shorter solution can be then selected as a result. Thus, the worst case error does not exceed 3 for any approximation algorithm discussed in this section.

We now introduce an approximation algorithm based on the Longest Processing Time (LPT) algorithm [7], a standard approximation algorithm for  $P||C_{\max}$ . The idea behind this algorithm is in order to find a short multi-slot just-in-time schedule we need to assign jobs to machines so as to partition the workload measured by the total job processing time in a sufficiently even way between the machines. A significantly uneven partition may result in a rather long schedule even if the minimization of the total idle time is subsequently done optimally on each machine. Having this in mind we deal with the multi-slot just-in-time problem by first partitioning the workload of long jobs between the machines by using the LPT, and then by adding short jobs to the resulting schedule. We begin by defining long and short jobs first.

**Definition 2** Let an integer  $k > 0$  be given. A job  $J_j$ ,  $j \in \{1, \dots, n\}$ , is  $k$ -long if  $\lfloor p_j/L \rfloor \geq k$ . Otherwise the job is called  $k$ -short. Divide the set  $\mathcal{J}$  as follows

$$\mathcal{J}_s = \{J \in \mathcal{J} : J \text{ is } k\text{-short}\}, \quad \mathcal{J}_l = \mathcal{J} \setminus \mathcal{J}_s.$$

The integer  $k$  is fixed in the remainder of the section, thus for brevity we use the terms long and short job, respectively.

**Definition 3** A schedule is called *normal* if on each machine the jobs are scheduled in such a way that each long job is completed before the beginning of each short job. We use the symbol JITN in the notation introduced in Section 2 to denote a normal schedule of minimum makespan.

The following theorem shows that the shortest normal just-in-time schedule may be at most  $(1 + 2/k)$  times longer than the shortest just-in-time schedule.

**Theorem 6** For each set of jobs  $\mathcal{J}$  and each number of machines  $m$  we have

$$\text{len}(S(m, \mathcal{J}, \text{JITN})) \leq \left(1 + \frac{2}{k}\right) \text{len}(S(m, \mathcal{J}, \text{JIT})).$$

**Proof:** Let  $S = S(m, \mathcal{J}, \text{JIT})$ . By a *block* of short (respectively long) jobs in  $S$  on machine  $M_i \in \mathcal{M}$  we mean a maximal subsequence of short (long, resp.) jobs scheduled consecutively on  $M_i$ . We perform a sequence of exchanges in  $S$ , so that the final schedule  $S'$  is normal. Let  $S^0 = S$ . Given  $S^i$  we create  $S^{i+1}$  as follows. Find a machine  $M_l \in \mathcal{M}$  and a block of short jobs  $B$ , such that there is a block of long jobs following  $B$ . Move  $B$  to the end of the schedule on machine  $M_l$ . If there is no such block  $B$  then  $S' = S^i$  is the final schedule. Clearly  $S'$  is normal. Figure 5 illustrates this process. Here  $J_2, J_4$  are short, while  $J_1, J_3$  are long jobs. The single job  $J_2$  forms a block of short jobs in the schedule in Figure 5(a), moved to the end of the schedule as shown in Figure 5(b).

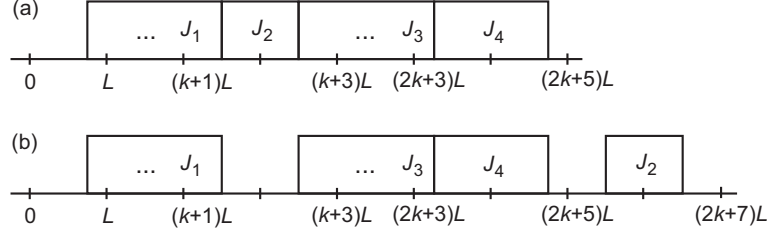


Figure 5: Obtaining normal schedule: (a) a schedule with two blocks of short jobs and two blocks of long jobs, (b) the corresponding normal schedule created by moving the first block of short jobs to the end of the schedule

Now let us check how an operation of moving a block  $B$  of short jobs affects the makespan of the schedule. If  $B$ , moved to the end, intersected at most two slots, then the schedule on  $M_l$  becomes at most two slots longer. If it intersects  $s > 2$  slots, then removing  $B$  from  $M_l$  decreases the  $\text{len}(S^i|_{M_l})$  by at least  $s - 2$  slots – the number of the inner slots of  $B$ . Then, inserting  $B$  at the end of the schedule on  $M_l$  increases the schedule by at most  $s$ . So, the schedule on  $M_l$  becomes at most two slots longer for each operation of moving a block. For each machine  $M_i$ , the number of such operations on  $M_i$  does not exceed the number of blocks of long jobs on  $M_i$ . This number however is bounded by the number of long jobs on  $M_i$ , which is at most  $\text{len}(S(m, \mathcal{J}, \text{JIT})|_{M_i})/k \leq \text{len}(S(m, \mathcal{J}, \text{JIT}))/k$ . This gives the required ratio of  $(1 + 2/k)$ .  $\square$

We are now ready to give details of our approximation algorithm referred to as  $A$ . For each job  $J_j \in \mathcal{J}_l$  define  $\tilde{p}_j = L \lfloor \frac{p_j}{L} \rfloor$ . The job with processing time  $\tilde{p}_j$  and due date  $\tilde{d}_j = L$  is denoted by  $\tilde{J}_j$ . Note that the problem of scheduling jobs  $\tilde{J}_j \in \tilde{\mathcal{J}}_l$  corresponding to long jobs  $J_j \in \mathcal{J}_l$  is equivalent to the problem  $P||C_{\max}$ . Though we use the Longest Processing Time (LPT) algorithm for the latter, we note that any other algorithm for  $P||C_{\max}$  can be used instead of LPT. If that is the case then, the approximation ratio in the discussion that follows should be changed accordingly. The algorithm  $A$  is as follows.

1. Use the LPT algorithm to find a schedule  $S(m, \tilde{\mathcal{J}}_l, \text{LPT})$ .
2. Compute  $S(m, \mathcal{J}_l, \text{LPT})$  from  $S(m, \tilde{\mathcal{J}}_l, \text{LPT})$  by simply replacing each job  $\tilde{J}_j \in \tilde{\mathcal{J}}_l$  by the corresponding job  $J_j$ . The ordering of the jobs does not change on any machines.
3. Find an optimal schedule  $S(1, \mathcal{J}_s, \text{JIT})$ .
4. Compute a schedule  $S(m, \mathcal{J}, \text{JITN})$  from the schedules  $S(m, \mathcal{J}_l, \text{LPT})$  and  $S(1, \mathcal{J}_s, \text{JIT})$  as follows:

4a. let

$$s := \text{slots}(S(1, \mathcal{J}_s, \text{JIT})) + \sum_{M \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}_l, \text{LPT})|_M), \quad (18)$$

be the total number of slots used in both schedules.

4b. let machines  $M_1, \dots, M_m$  be ordered in descending order of

$$\lceil s/m \rceil - \text{slots}(S(m, \mathcal{J}_l, \text{LPT})|_{M_i}). \quad (19)$$

For each  $1, \dots, m$  remove the sequence, if any, of jobs from the schedule  $S(1, \mathcal{J}_s, \text{JIT})$  containing all jobs intersecting the interval:

$$\left(0, L \cdot \lceil s/m \rceil - L \cdot \text{slots}(S(m, \mathcal{J}_l, \text{LPT})|_M)\right) \quad (20)$$

and schedule these jobs on  $M_i$  after the block of long jobs on  $M_i$ . Make the remainder of the schedule  $S(1, \mathcal{J}_s, \text{JIT})$  to start at 0.

In Sections 6.1 and 6.2 we derive the following approximation guarantee for  $A$ .

**Theorem 7** *For any set of jobs  $\mathcal{J}$  it holds*

$$\begin{aligned} \text{len}(S(m, \mathcal{J}, A)) \leq & \max \left\{ \frac{4}{3} \left(1 + \frac{2}{k}\right) \text{len}(S(m, \mathcal{J}, \text{JIT})), \right. \\ & \left. \left(1 + \frac{2}{k}\right)^2 \text{len}(S(m, \mathcal{J}, \text{JIT})) + \left(k + 2 + \left(1 + \frac{2}{k}\right)^2\right)L \right\}. \end{aligned}$$

The ratio of the algorithm  $A$  depends on the properties of the final schedule, namely if the final schedule has the same makespan as the intermediate schedule for long jobs computed in Step 2 of  $A$ , i.e.  $\text{len}(S(m, \mathcal{J}, A)) = \text{len}(S(m, \mathcal{J}_l, \text{LPT}))$ , then  $\text{len}(S(m, \mathcal{J}, A)) \leq \frac{4}{3} \left(1 + \frac{2}{k}\right) \text{len}(S(m, \mathcal{J}, \text{JIT}))$ . However, if  $\text{len}(S(m, \mathcal{J}_l, \text{LPT})) < \text{len}(S(m, \mathcal{J}, A))$ , then we obtain that

$$\text{len}(S(m, \mathcal{J}, A)) \leq \left(1 + \frac{2}{k}\right)^2 \text{len}(S(m, \mathcal{J}, \text{JIT})) + \left(k + 2 + \left(1 + \frac{2}{k}\right)^2\right)L.$$

Both cases are described in Sections 6.1 and 6.2, respectively. Moreover, observe that if there is no job in  $\mathcal{J}$  with  $p_j \in [aL, bL]$ ,  $1 \leq a < b$  then each value of  $k \in [a, b)$  results in the same schedule computed by the algorithm  $A$ . This fact can be used to substitute the value of  $k$  giving the best bound in Theorem 7.

The approximation ratio can be bounded by a fixed number in the interval  $[4/3, 3]$  in the worst case. The lower bound on this ratio is a result of choosing the LPT algorithm in Step 1 of the algorithm  $A$ , and the upper bound follows from the argument given at the beginning of this Section. It is worth observing that any general fixed approximation ratio for  $A$  could be quite inaccurate for any particular instance with concrete values of  $k$  and  $L$ . Therefore, the parametric, in terms of  $k$  and  $L$ , form of the bound given in Theorem 7 appears more useful than less accurate fixed bound.

### 6.1 Case 1: $\text{len}(S(m, \mathcal{J}, A)) = \text{len}(S(m, \mathcal{J}_l, \text{LPT}))$

We start with an example. Let  $L > 4$ ,  $\mathcal{J} = \{J_1, \dots, J_7\}$ ,  $p_1 = 10L, d_1 = L, p_2 = 9L - 1, d_2 = \frac{1}{2}L, p_3 = 9L - 1, d_3 = \frac{1}{4}L$ . The first three jobs are long, for any  $1 \leq k \leq 8$ , and the remaining jobs are short and their single machine optimal schedule is shown in Figure 6(b). An LPT schedule for  $\tilde{\mathcal{J}}_l = \{\tilde{J}_1, \tilde{J}_2, \tilde{J}_3\}$  is shown in Figure 6(a). The final schedule, where each job  $\tilde{J}_j$  has been replaced by its corresponding job  $J_j$ ,  $j = 1, 2, 3$ , and all the short jobs have been placed on  $M_1$  is shown in Figure 6(c).

We now return to the case itself. We obtain  $S(m, \mathcal{J}_l, \text{LPT})$  from  $S(m, \tilde{\mathcal{J}}_l, \text{LPT})$  in Step 2 of  $A$ . We have  $\text{slots}(J_j) = \text{slots}(\tilde{J}_j) + l$  for  $l \geq 0$ . By the definition of  $\tilde{\mathcal{J}}_l$  we have  $l \in \{0, 1, 2\}$ . Thus, replacing  $\tilde{J}_j$  by  $J_j$  does introduce at most two extra slots to the schedule. Since  $J_j$  is long, by definition we have that  $\tilde{p}_j \geq kL$ . This implies that the number of long jobs on a machine  $M_i$ ,  $i \in \{1, \dots, m\}$  is at most  $\text{len}(S(m, \mathcal{J}_l, \text{LPT})|_{M_i})/k$ , thus

$$\text{len}(S(m, \mathcal{J}_l, \text{LPT})) \leq \left(1 + \frac{2}{k}\right) \text{len}(S(m, \tilde{\mathcal{J}}_l, \text{LPT})). \quad (21)$$

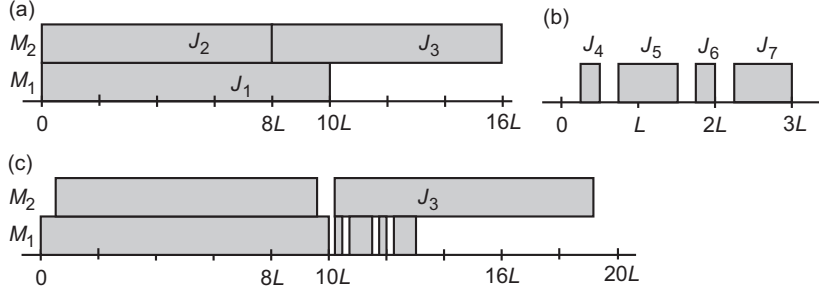


Figure 6: (a)  $S(2, \tilde{\mathcal{J}}, \text{LPT})$ , (b)  $S(1, \mathcal{J}_s, \text{JIT})$ , (c)  $S(2, \mathcal{J}, A)$

The Graham LPT algorithm [7] ensures

$$\text{len}(S(m, \tilde{\mathcal{J}}, \text{LPT})) \leq \frac{4}{3} \text{len}(S(m, \tilde{\mathcal{J}}, C_{\max})). \quad (22)$$

Clearly,

$$\text{len}(S(m, \tilde{\mathcal{J}}, C_{\max})) = \text{len}(S(m, \tilde{\mathcal{J}}, \text{JIT})) \quad (23)$$

and

$$\text{len}(S(m, \tilde{\mathcal{J}}, \text{JIT})) \leq \text{len}(S(m, \mathcal{J}, \text{JIT})). \quad (24)$$

For the case

$$\text{len}(S(m, \mathcal{J}, A)) = \text{len}(S(m, \mathcal{J}_l, \text{LPT})), \quad (25)$$

the equations (21)–(25) give

$$\text{len}(S(m, \mathcal{J}, A)) \leq \frac{4}{3} \left(1 + \frac{2}{k}\right) \text{len}(S(m, \mathcal{J}, \text{JIT})). \quad (26)$$

Observe that it was not necessary to use Theorem 6 in this case.

## 6.2 Case 2: $\text{len}(S(m, \mathcal{J}_l, \text{LPT})) < \text{len}(S(m, \mathcal{J}, A))$

This case is illustrated in Figure 7. Again, let  $L > 4$ , and  $\mathcal{J} = \{J_1, \dots, J_7\}$ . We have three long jobs with parameters:  $p_1 = 8L, p_2 = 3L, p_3 = 10L, d_1 = d_2 = d_3 = L$ . The first three jobs are long, for any  $1 \leq k \leq 3$ , and the remaining jobs are short. We remove from  $S(1, \mathcal{J}_s, \text{JIT})$ , shown in Figure 7(b), a sequence of jobs intersecting the interval  $(0, 2L)$  and we append it to the LPT schedule on  $M_1$  shown in Figure 7(a). The remainder of  $S(1, \mathcal{J}_s, \text{JIT})$ , i.e. the job  $J_7$ , goes on machine  $M_2$ . The resulting schedule is given in Figure 7(c).

We now return to the case itself. For the purposes of this subsection we extend the notation from Section 2. Let  $S(m, X, Y)_{M_i, l}$ , and  $S(m, X, Y)_{M_i, s}$  denote the block of long and short jobs respectively on machine  $M_i$  in the schedule  $S(m, X, Y)$ . This notation is only used for normal schedules. Thus, for a normal schedule  $S(m, X, Y)$ ,  $S(m, X, Y)|_{M_i}$  is a concatenation of  $S(m, X, Y)_{M_i, l}$  and  $S(m, X, Y)_{M_i, s}$  for each machine  $M_i \in \mathcal{M}$ .

Cutting from the remaining single machine schedule a sequence of jobs intersecting the interval (20) in Step 4b and appending it to the schedule of long jobs on machine  $M$  lengthen the schedule on  $M$  beyond  $\lceil s/m \rceil$  slots if a job  $J_j \in \mathcal{J}_s$  starts in the last slot intersecting the interval (20) and does not end in this slot. The execution of  $J_j$  ends at  $\lceil s/m \rceil + p_j - 1$  in the worst case. Since  $J_j$  is short we have  $\lfloor p_j/L \rfloor \leq k - 1$ . So,  $p_j \leq kL$  and consequently  $J_j$  lengthens the schedule on  $M$  by at most  $kL$  units (i.e.  $k$  slots) beyond  $\lceil s/m \rceil$ . Hence, by (18) we have

$$\text{slots}(S(m, \mathcal{J}, A)|_{M_i}) \leq \lceil s/m \rceil + k, \quad (27)$$

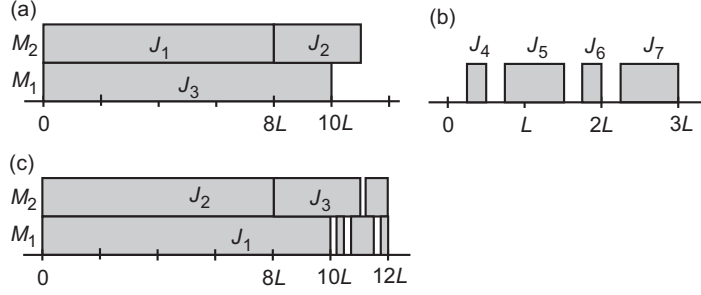


Figure 7: (a)  $S(2, \tilde{\mathcal{J}}, \text{LPT})$ , (b)  $S(1, \mathcal{J}_s, \text{JIT})$ , (c)  $S(2, \mathcal{J}, A)$

for a machine  $M_i \in \mathcal{M}$ . Since  $s \leq \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)|_{M_i})$  we obtain that

$$\text{slots}(S(m, \mathcal{J}, A)) \leq k + \left\lceil \frac{1}{m} \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)|_{M_i}) \right\rceil. \quad (28)$$

Clearly, for each  $M_i \in \mathcal{M}$  we have

$$\text{slots}(S(m, \mathcal{J}, A)|_{M_i}) \leq \text{slots}(S(m, \mathcal{J}, A)_{M_i, l}) + \text{slots}(S(m, \mathcal{J}, A)_{M_i, s}). \quad (29)$$

Thus,

$$\sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)|_{M_i}) \leq \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)_{M_i, l}) + \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)_{M_i, s}). \quad (30)$$

Each operation of removing several slots from a schedule  $S(1, \mathcal{J}_s, \text{JIT})$  and placing them on one of the machines may result in creating at most one additional slot. This gives a bound

$$\sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)_{M_i, s}) \leq \text{slots}(S(1, \mathcal{J}_s, \text{JIT})) + m. \quad (31)$$

Since the blocks of short jobs in the schedule  $S(m, \mathcal{J}, A)$  come from an optimal just-in-time schedule for a single machine, we have that

$$\text{slots}(S(1, \mathcal{J}_s, \text{JIT})) \leq \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, s}). \quad (32)$$

Inequalities (31) and (32) imply

$$\sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)_{M_i, s}) \leq \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, s}) + m. \quad (33)$$

We now want to bound the sum for long jobs in Equation (30). We have

$$\sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \tilde{\mathcal{J}}, \text{LPT})_{M_i, l}) \leq \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, l}), \quad (34)$$

since the schedule  $S(m, \tilde{\mathcal{J}}, \text{LPT})$  has jobs shorter than the corresponding jobs in  $S(m, \mathcal{J}, \text{JITN})$ , and there is no idle time between jobs in  $S(m, \tilde{\mathcal{J}}, \text{LPT})$ . Hence, replacing each job  $\tilde{J}_j$  by its corresponding possibly

longer job  $J_l$  in the schedule  $S(m, \tilde{\mathcal{J}}_l, \text{LPT})$  adds at most two additional slots for each long job which implies

$$\sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)_{M_i, l}) \leq (1 + \frac{2}{k}) \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \tilde{\mathcal{J}}_l, \text{LPT})_{M_i, l}). \quad (35)$$

Equations (30) and (33)-(35) give the following:

$$\begin{aligned} \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, A)|_{M_i}) &\leq (1 + \frac{2}{k}) \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, l}) \\ &\quad + \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, s}) + m. \end{aligned} \quad (36)$$

By the definition of a normal schedule we have

$$\text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, l}) + \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, s}) = \text{slots}(S(m, \mathcal{J}, \text{JITN})|_{M_i}) \quad (37)$$

for  $M_i \in \mathcal{M}$  and, as a consequence,

$$\begin{aligned} \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, l}) + \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})_{M_i, s}) \\ = \sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})|_{M_i}) \end{aligned} \quad (38)$$

Since  $\sum_{M_i \in \mathcal{M}} \text{slots}(S(m, \mathcal{J}, \text{JITN})|_{M_i}) \leq m \cdot \text{slots}(S(m, \mathcal{J}, \text{JITN}))$ , by (28), (36) and (38) we obtain the approximation ratio of the form

$$\text{slots}(S(m, \mathcal{J}, A)) \leq (1 + \frac{2}{k}) \text{slots}(S(m, \mathcal{J}, \text{JITN})) + k + 2. \quad (39)$$

By Theorem 6,

$$\text{slots}(S(m, \mathcal{J}, A)) \leq (1 + \frac{2}{k})^2 \text{slots}(S(m, \mathcal{J}, \text{JIT})) + k + 2. \quad (40)$$

Clearly, for each schedule  $S$  we have  $\text{len}(S) \leq L \text{slots}(S)$  and  $L(\text{slots}(S) - 1) \leq \text{len}(S)$ . Using Equation (40) we may bound the length of  $S(m, \mathcal{J}, A)$

$$\text{len}(S(m, \mathcal{J}, A)) \leq (1 + \frac{2}{k})^2 \text{len}(S(m, \mathcal{J}, \text{JIT})) + (k + 2 + (1 + \frac{2}{k})^2)L. \quad (41)$$

This ends the proof of the case and Theorem 7.

## 7 Conclusions and open problems

We present polynomial time algorithms for two special cases of the multi-slot just-in-time scheduling problem: the single machine case and the case for  $m > 1$  parallel identical machines and  $p_j \leq d_j$  for each job  $J_j$ ,  $j = 1, \dots, n$ . Our goal is to minimize the makespan of the schedule instead of the number of slots used. Another improvement presented in this paper is the time complexity for the case of  $m > 1$  machines and jobs  $J_j$  with  $p_j \leq d_j$ . To solve this case we have reduced the just-in-time multi-slot scheduling problem to finding optimal colorings of intervals.

For the general problem on  $m > 1$  machines we have described an approximation algorithm and proved its approximation ratio. A challenging open problem is the complexity of the multi-slot just-in-time scheduling problem for fixed number of  $s > 2$  slots. Such a result would have obvious applications in deriving an inapproximability bound for the problem.

Another open intriguing problem is the precise complexity of the multi-slot just-in-time scheduling problem for a fixed number of  $m > 1$  machines. Though the problem is NP-hard it remains open whether there is a pseudopolynomial time algorithm for it. Such an algorithm would imply that the problem is polynomially solvable if each job length  $p_j$  is bounded by a polynomial in  $n$ , and could result in better approximation algorithms for fixed  $m$ .

### Acknowledgment

This research has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant OPG0105675. Moreover, Dariusz Dereniowski has been supported by the Foundation for Polish Science (FNP) and by MNiSW grant N516 029 31/2941.

## References

- [1] D. Barth, J. Cohen, L. Gastal, T. Mautor, and S. Rousseau. Fixed size and variable size packet models in an optical ring network: Complexity and simulations. In *ISCIS*, pages 238–246, 2004.
- [2] M. Biró, M. Hujter, and Zs. Tuza. Precoloring extension. i: Interval graphs. *Discrete Math.*, 100(1-3):267–279, 1992.
- [3] J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling computer and manufacturing processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.
- [4] M. C. Carlisle and E.L. Lloyd. On the  $k$ -coloring of intervals. *Discrete Appl. Math.*, 59:225–235, 1995.
- [5] E. Chiba and K. Hiraishi. A heuristic algorithm for one-machine just-in-time scheduling problem with periodic time slots. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E88-A(5):1192–1199, 2005.
- [6] P. C. Gilmore and R. E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679, 1964.
- [7] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.
- [8] K. Hiraishi. Just-in-time scheduling of parallel identical machines with multiple time slots. In *Proceedings of the 4th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty*, Jindřichuv Hradec, 2001.
- [9] K. Hiraishi, E. Levner, and M. Vlach. Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs. *Comput. Oper. Res.*, 29(7):841–848, 2002.
- [10] J. Józefowska. *Just-in-Time Scheduling*. Springer, New York, 2007.
- [11] W. Kubiak. *Proportional Optimization and Fairness*. Springer, New York, 2009.
- [12] A. Kumar and J. Kleinberg. Fairness measures for resource allocation. *SIAM J. Comput.*, 36(3):657–680, 2006.
- [13] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Travelling Salesman Problem*. Wiley, New York, 1985.
- [14] N. Megiddo. Optimal flows in networks with multiple sources and sinks. *Math. Programming*, 7:97–107, 1974.
- [15] S-Ch. Sung, O. Čeppek, and K. Hiraishi. Algorithm for multislot just-in-time scheduling. In *ISME '07: International Symposium on Management Engineering*, 2007.

- [16] S-Ch. Sung, O. Čepeċ, and K. Hiraishi. Algorithm for multislot just-in-time scheduling on identical parallel machines. In *ICICIC '07: Proceedings of the Second International Conference on Innovative Computing, Information and Control*, page 120, Washington, DC, USA, 2007. IEEE Computer Society.
- [17] G. L. Vairaktarakis. Simple algorithms for Gilmore–Gomory’s traveling salesman and related problems. *J. of Scheduling*, 6(6):499–520, 2003.
- [18] O. Čepeċ and S-Ch. Sung. Just-in-time scheduling with periodic time slots. *Scientiae Mathematicae Japonicae Online*, 10:431–437, 2004.