Coupled task scheduling with time-dependent processing times

Mostafa Khatami[*]   Amir Salehipour[†]

Abstract

The single machine coupled task scheduling problem includes a set of jobs, each with two separated tasks and there is an exact delay between the tasks. We investigate the single machine coupled task scheduling problem with the objective of minimizing the makespan under identical processing time for the first task and identical delay period for all jobs, and the time-dependent processing time setting for the second task. Certain healthcare appointment scheduling problems can be modeled as the coupled task scheduling problem. Also, the incorporation of time-dependent processing time for the second task lets the human resource fatigue and the deteriorating health conditions be modeled. We provide optimal solution under certain conditions. In addition, we propose a dynamic program under the condition that the majority of jobs share the same time-dependent characteristic. We develop a heuristic for the general case and show that the heuristic performs well.

Keywords: coupled task scheduling; time-dependent processing time; simple linear processing time; dynamic program; heuristic; healthcare scheduling

1 Introduction

The single machine coupled task scheduling problem aims to schedule a set of jobs on a single machine (processor) such that a performance criterion (objective function) is optimized. In this study, we investigate the performance criterion of minimizing the makespan, i.e. the completion time of the last job in the sequence. Each job consists of two separated tasks with an exact delay (time interval) between them. The second (completion) task must be processed after the completion of the first (initial) task. A job $j \in J$, where J is the set of all jobs, is shown by a triple (a_j, L_j, b_j). Parameters a_j, L_j and b_j denote the processing time of the initial task, the amount of delay and the processing time of the completion task.

Shapiro (1980) modeled a pulsed radar system as a coupled task scheduling problem. A pulse of electromagnetic energy is used to track an object. The pulse is transmitted, and then, its reflection is received after a period of time, helping to measure the size and/or the shape of the object. The objective is to maximize the number of detected objects. Other applications of the coupled task problem include certain scheduling problems in chemistry manufacturing, where there is an exact technological delay between the completion time of the first task and the starting time of the second one (Ageev and Baburin, 2007), in robotic cells, in which a cell includes input and output stations, a machine and a robot, which transports material between stations and machine (Lehoux-Lebacque et al., 2015), and in healthcare appointment scheduling, for example, in a chemotherapy treatment once the medicine is prescribed the patient visits the health center at treatment days separated by a fixed number of rest days (Condotta and Shakhlevich, 2014).

The coupled task scheduling problem can be applied to certain healthcare problems with multi-stage characteristics. Consider scheduling the appointment of patients in a nuclear medicine clinic. The problem consists of strict multi-stage sequential procedures (Pérez et al., 2011; Pérez et al., 2013), where a single procedure requires multiple stages and each stage needs to be successfully completed within a strict time window. The cost of required resources and the short half-life of the radio-pharmaceuticals needed for the

[*]School of Mathematical and Physical Sciences, University of Technology Sydney, Australia. Email: mostafa.khatami@student.uts.edu.au

[†]Corresponding author. School of Mathematical and Physical Sciences, University of Technology Sydney, Australia. Email: amir.salehipour@uts.edu.au

procedure justifies minimizing a performance criterion related to the patients flow time. In this context, the tiredness of the staff, which typically occurs in practice and impacts the processing time and hence, the start time of the next job, can be modeled as a time-dependent event. For example, Pérez et al. (2013) highlights the importance of modeling human resource fatigue within the patient appointment scheduling in nuclear medicine clinics or in similar environments, proposing therefore studies on the coupled task scheduling with time-dependent processing times. The problem of scheduling patients in a pathology laboratory is another example. Certain blood tests, e.g., the fasting blood sugar test, require multiple tests and there is an exact time interval between a pair of tests. Although the next test cannot be administered until an exact delay is elapsed, the tests associated with other patients can be administered within such a delay period. In the simplest form, a pair of tests form a coupled task job. Here, the performance criterion is to minimize the waiting time of the patients (Marinagi et al., 2000; Azadeh et al., 2014). This problem can also be investigated with regard to the human resource fatigue. Similar characteristics can be found in the patient appointment scheduling in radiotherapy clinics (Legrain et al., 2015) and in hemodialysis treatment services (Liu et al., 2019).

We also note that Mosheiov (1994) and Gawiejnowicz (2008) studied the time-dependent processing times to model the time of performing the medical services as the time increases under deteriorating health conditions. It is true that in the aforementioned applications, particularly, in chemotherapy and radiotherapy services, the symptoms of the patient rapidly grows over time, i.e., the patient needs longer treatment if that treatment is started late.

Shapiro (1980) showed that the coupled task scheduling problem is equivalent to an NP-hard class of two-machine job-shop problem. They proposed two heuristic algorithms of "interleaving" and "nesting" for the problem. The interleaving heuristic aims to sequence jobs such that the completion tasks arrive for processing in the same order as the initial tasks (Figure 1a). In the nesting procedure, the completion tasks arrive in the reverse order of the initial tasks (Figure 1b). Scheduling a single job without interleaving or nesting is often referred to as "appending". The strong NP-hardness of minimizing the makespan for the coupled task problem was proved by Sherali and Smith (2005). Condotta and Shakhlevich (2012) showed that the problem remains strongly NP-hard even if the sequence for the initial tasks is given. Several special cases were also shown to be strongly NP-hard. For example, under the objective function of minimizing the makespan problems $(a_j = L_j = b_j)$, $(a_j, L_j = L, b_j = b)$ and $(a_j = p, L_j, b_j = p)$, where L, b, and p are positive integers, are strongly NP-hard (Orman and Potts, 1997). Even the complexity of a restricted case where all jobs have the same tasks and delay, i.e. $(a_j = a, L_j = L, b_j = b)$, which is commonly known as the "identical" case, is still open. Two notable studies for the identical case are due to Ahr et al. (2004), who proposed an $O(nr^{2L})$-time dynamic program, where r is a function of a, and to Baptiste (2010), who improved the complexity to $O(\log n)$ for fixed a, L and b.

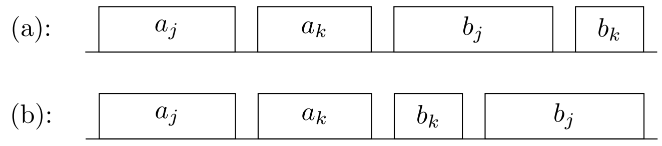

Figure 1: Interleaving jobs j and k (a) and nesting jobs j and k (b).

For the general case of the problem, Ageev and Kononov (2007) proposed a 3.5-approximation algorithm that operates by ordering the jobs in a non-increasing order of $(a_j + L_j)$. Li and Zhao (2007) presented lower bounds for the problem. Heuristic algorithms (Li and Zhao, 2007; Condotta and Shakhlevich, 2012) and exact methods (Békési et al., 2014) were also proposed. The recent heuristic of binary search by Khatami and Salehipour (2020) systematically tightens lower and upper bounds until a feasible schedule is obtained. Several studies investigated the problem with additional constraints, e.g. with the precedence constraints (Blazewicz et al., 2010), with the compatibility constraints (Simonin et al., 2011; Bessy and Giroudeau, 2019), and with the fixed-job-sequence constraints (Hwang and Lin, 2011). In the shop environment, Yu et al. (2004) and Leung et al. (2007) studied the makespan minimization in a two-machine flow-shop problem. Ageev (2018) recently studied the complexity of the open-shop problem with the coupled tasks. For a comprehensive review of the coupled task studies, applications, and models we refer the interested reader to Khatami et al. (2020).

Certain special cases of the coupled task problem have been shown to be polynomially solvable. For example, Orman and Potts (1997) proved that the case $(a_j = p, L_j = p, b_j)$ (identical initial tasks and delays for all jobs) is polynomially solvable for the objective function of minimizing the makespan. We investigate the same case, and also with the objective function of minimizing the makespan; however, with a time-dependent processing time characteristic for the completion tasks. Under this setting, the processing time of the completion task depends on its starting time. To the best of our knowledge, the present study is the first attempt towards studying the coupled task scheduling problem with time-dependent processing time characteristic.

Gupta and Gupta (1988) introduced the scheduling problems with time-dependent processing times. The time-dependent processing times have various applications, e.g. in steel production, in financial management and in resource allocation, where a delay in starting a job may decrease or increase its processing time (Kunnathur and Gupta, 1990; Cheng et al., 2004). Under this setting, job j has a normal processing time $\alpha_j \geq 0$ and a processing rate $\beta_j \geq 0$. The actual processing time of job j depends on its starting time s_j, and is typically shown as $p_j = \alpha_j \pm \beta_j s_j$. A variant of this model, which is called the "simple linear processing times", assumes $\alpha_j = 0$, and therefore, $p_j = \beta_j s_j$. We investigate the simple linear processing times model for the completion tasks. Therefore, with considering the three-field scheduling notation proposed by Graham et al. (1979), the present study investigates the problem $1|(a_j = p, L_j = p, b_j = \beta_j s_j)|C_{max}$. We assume a_j, L_j and b_j only take positive integers values. There are other cases that we do not study in this paper, either because their time-dependent processing times counterpart remains polynomially solvable or remains strongly NP-hard. For example, the case $(a_j = p, L_j = L, b_j = p)$, i.e., all jobs are identical was shown to be polynomially solvable (Orman and Potts, 1997). We do not study that case with the time-dependent processing times because the problem is trivial if all jobs share the same processing rate (a greedy approach leads to the optimal makespan). Also, the strongly NP-hard cases of $(a_j = L_j = b_j)$, $(a_j, L_j = L, b_j = b)$ and $(a_j = p, L_j, b_j = p)$ remain strongly NP-hard with the addition of the time-dependent processing times.

The remainder of this paper is organized as follows. In Section 2, we present a mathematical formulation for the problem. In Section 3, we discuss optimal properties for the problem. A dynamic program and a heuristic are also proposed. The results of numerical experiments are presented in Section 4. The paper ends with a few conclusions in Section 5.

2 Problem definition and formulation

Given a set of coupled task jobs $J = \{1, 2, \ldots, n\}$, each with two tasks and there is an exact delay period between two consecutive tasks, to be processed on a single machine, a job $j \in J$ is represented by $(a_j = p, L_j = p, b_j = \beta_j s_j)$, where p is a positive integer, and $\beta_j, s_j > 0, \forall j \in J$. Therefore, parameter $b_j, \forall j$ is a time dependent variable defined by a simple linear processing time. The goal is to develop a schedule for $(a_j = p, L_j = p, b_j = \beta_j s_j)$, so to minimize the makespan, i.e. C_{max}.

There are a number of mathematical programs available in the literature for the general coupled task problem. Khatami et al. (2020) discussed that the model proposed by Békési et al. (2014) is computationally among the top performing models. Hence, we extend that formulation for the problem of this study. The formulation utilizes linear ordering variables, and the sequence is therefore built by ordering the tasks. For this reason, we define a set of tasks $H = \{1, 2, \ldots, 2n\}$, where H_{2j-1} and H_{2j} represent the initial and completion tasks of job j. For any pair of tasks h, h', we define a binary variable $x_{h,h'}$, which takes a value of 1 if task h' starts after task h in the sequence, and 0 otherwise. The problem P1 below shows the formulation for $(a_j = p, L_j = p, b_j = \beta_j s_j)$.

Problem P1

$$z = \min C_{max} \tag{1}$$

subject to

$$C_{max} \geq s_{2j} + \beta_j s_{2j}, \quad 1 \leq j \leq n, \tag{2}$$

$$x_{2j-1,2j} = 1, \quad 1 \leq j \leq n, \tag{3}$$

$$x_{h,h'} + x_{h',h} = 1, \quad 1 \leq h < h' \leq 2n, \tag{4}$$

$$x_{h,h'} + x_{h',h''} + x_{h'',h} \leq 2, \quad 1 \leq h < h' < h'' \leq 2n, \tag{5}$$

$$s_{2j} = s_{2j-1} + 2p, \quad 1 \leq j \leq n, \tag{6}$$

$$s_{2j} \leq UB - \beta_j s_{2j}, \quad 1 \leq j \leq n, \tag{7}$$

$$s_h \geq s_{2j-1} + p - UB(1 - x_{2j-1,h}), \quad 1 \leq j \leq n, \quad 1 \leq h \leq 2n, \quad h \notin \{2j-1, 2j\}, \tag{8}$$

$$s_h \geq s_{2j} + \beta_j s_{2j} - UB(1 - x_{2j,h}), \quad 1 \leq j \leq n, \quad 1 \leq h \leq 2n, \quad h \notin \{2j-1, 2j\}, \tag{9}$$

$$s_h \geq 0, \quad 1 \leq h \leq 2n, \tag{10}$$

$$x_{h,h'} \in \{0, 1\}, \quad 1 \leq h, h' \leq 2n, \quad h \neq h'. \tag{11}$$

The objective function (Equation (1)) minimizes the makespan. The constraints (2) ensure that the makespan is larger than the completion time of any job. Constraints (3) ensure that the completion task of each job should be scheduled after its initial task. Constraints (4) and (5) set the relative order of any pair of tasks and any triple distinct tasks, respectively. The link between the starting time of the tasks (of the same job) is established by constraints (6), while an upper bound (UB) is considered for the starting time of the completion tasks in constraints (7). Constraints (8) and (9) relate the starting time of tasks, and also relate the starting time variables to the linear ordering variables. Constraints (10) and (11) ensure that the decision variables are non-negative and binary. We note that the total number of variables and constraints of the model is equal to $4n^2 + 1$ and $\frac{8}{3}n^3 + 2n^2 + \frac{1}{3}n$, respectively.

3 Minimizing the makespan

We show that problem P1 can be easily solved for these two cases: (1) $\beta_j > 0.5, \forall j \in J$, and (2) a two-job instance. In addition, under the condition that jobs are grouped into a few classes we propose a dynamic program for problem P1 that delivers the optimal schedule in polynomial time. In many applications it is indeed safe to group the jobs. For the general case, we propose an efficient heuristic algorithm.

3.1 Optimal schedule

Orman and Potts (1997) showed that for problem $(a_j = p, L_j = p, b_j)$ under general processing times, the nesting of jobs is not possible. For a pair of jobs j and k, if $b_j \leq p$, it is possible to interleave jobs j and k, where j is the first job and k is the second job of the pair. The contribution of this setting to the makespan is equal to $3p + b_k$. This is illustrated in Figure 2a. On the other hand, any job j with $b_j > p$ contributes $2p + b_j$ to the makespan (see Figure 2b). Therefore, the optimal schedule is derived when as many jobs as possible are interleaved. We will investigate whether this is the case in problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$.

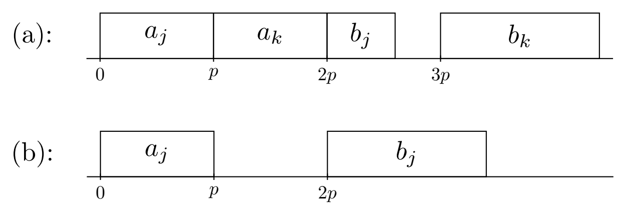

Figure 2: Contribution of an interleaving pair of jobs (a), and a single job (b) to the makespan.

In the classical single machine setting, the optimal schedule for both simple linear and linear time-dependent processing times exists. For example, under the simple linear condition Mosheiov (1994) showed that all schedules lead to the same makespan, which is equal to $s_1 \times \prod_j (1 + \beta_j), s_1 > 0$, where s_1 is the start time of the schedule. Under the linear processing times, Gupta and Gupta (1988) proved that the optimal makespan is obtained when jobs are sequenced in a non-decreasing order of α_j/β_j.

The results of Gupta and Gupta (1988) may be extended for problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$. We note that the combination of the initial task and the delay period of job j can be considered as the normal processing time of job j, i.e. $\alpha_j = p + p = 2p$. If no interleaving is possible, problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$ reduces to the single machine scheduling with linear time-dependent processing times, for which sequencing jobs in a non-decreasing order of α_j/β_j leads to the optimal makespan. Therefore, it is suffice to investigate if interleaving is possible.

It is safe to assume that the first job starts at time zero because all jobs are available at time zero. Then, the processing time of its completion task will be $b_j = \beta_j \times 2p$. Two cases are possible: (1) $\beta_j > 0.5, \forall j \in J$, and (2) $\beta_j \leq 0.5, \exists j \in J$. The following theorem leads to the optimal schedule if $\beta_j > 0.5, \forall j$.

Theorem 1. *The optimal solution for problem P1 is obtained when jobs are sorted in a non-increasing order of β_j, if and only if $\beta_j > 0.5, \forall j$.*

Proof. Without loss of generality let the first job start at time zero. Therefore, its completion task starts at time $2p$ and $b_j = \beta_j \times 2p$. It is clear that $b_j > p$, since $\beta_j > 0.5, \forall j$. Recall that there is no possibility for jobs interleaving if $b_j > p$ (see Figure 2b). Hence, the optimal sequence is obtained by ordering jobs in a non-decreasing order of α_j/β_j, or equivalently in a non-increasing order of β_j since $\alpha_j = 2p > 0, \forall j \in J$. □

The proof of Theorem 1 shows that even though the actual processing time of jobs depends on the start time of the completion tasks, this does not impact the optimal sequence because $\alpha_j = 2p, \forall j \in J$. The result of Theorem 1 may also be utilized to locate jobs that cannot be the first of an interleaving pair. This leads to the following lemma.

Lemma 1. *Under arbitrary values of β the jobs in set $\bar{J} \subset J$, where $\bar{J} = \{j | \beta_j > 0.5\}$ appear in the optimal schedule in a non-increasing order of $\beta_j, j \in \bar{J}$.*

Proof. Let $\beta_j > \beta_k > 0.5$ for jobs $j, k \in \bar{J}$. Assume that job k precedes job j in the optimal schedule. It is easy to see that swapping jobs j and k decreases the makespan, which implies that job k cannot precede job j in the optimal schedule. We note that the jobs in $\bar{J}$ cannot be the first of an interleaving pair, and swapping jobs j, k does not therefore change the order of other jobs. □

We now investigate the case of $\beta_j \leq 0.5, \exists j \in J$. There might be some possibility for interleaving of jobs. The following scenario shows the impact of interleaving two jobs on the makespan. Let $l = (p, p, b_l = \beta_l s_l)$ and $k = (p, p, b_k = \beta_k s_k)$ be a two-job instance of problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$. Also, let $\beta_l \leq 0.5$ and $\beta_k > 0.5$. Assume that the schedule starts at time zero and the first completion task therefore starts at time $2p$. Because $\beta_l \leq 0.5$, $\beta_l(2p) \leq p$, implying that the jobs can be interleaved if the schedule starts with l. On the contrary, because $\beta_k > 0.5$, and therefore $\beta_k(2p) \nleq p$, the interleaving of jobs is not possible if the schedule starts with k. The Gantt chart of Figure 3 illustrates these two cases. The makespan for those cases can be derived as follows.

$$(l,k): C_{(1)} = p + p + p + 3p\beta_k = 3p + 3p\beta_k, \tag{12}$$

$$(k,l): C_{(2)} = p + p + 2p\beta_k + p + p + (4p + 2p\beta_k)\beta_l = 4p + 2p\beta_k + (4p + 2p\beta_k)\beta_l. \tag{13}$$

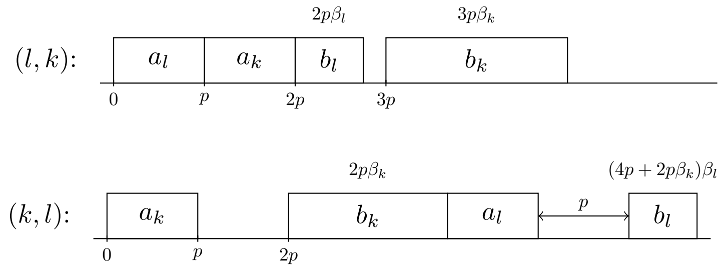

Figure 3: Two possible schedules for a two-job instance: (l,k), where interleaving occurs, and (k,l), where interleaving is not possible.

Obviously, we are interested in finding the values of β_l and β_k such that $C_{(1)} \le C_{(2)}$:

$$\begin{aligned} 3p + 3p\beta_k \le 4p + 2p\beta_k + (4p + 2p\beta_k)\beta_l &\implies \\ p\beta_k \le p + 4p\beta_l + 2p\beta_l\beta_k &\implies \\ \beta_k \le 1 + 4\beta_l + 2\beta_l\beta_k &\implies \\ \beta_k - 2\beta_l\beta_k \le 1 + 4\beta_l &\implies \\ \beta_k(1 - 2\beta_l) \le 1 + 4\beta_l&. \end{aligned} \tag{14}$$

It should be noted that if $\beta_l = 0.5$, Inequality (14) always holds, i.e. interleaving is beneficial. Following this, we propose Lemma 2.

Lemma 2. *If there exists a job l with $\beta_l = 0.5$, and the remaining jobs with $\beta_j > 0.5, \forall j \in J \setminus \{l\}$, the optimal schedule is obtained by interleaving job l with the job with the largest value of $\beta_j, j \in J \setminus \{l\}$, and sequencing the remaining jobs in a non-increasing order of their β values.*

Proof. Interleaving job l with a job $k, \beta_k > 0.5$ leads to a smaller makespan. This is shown in Inequality (14). It is clear that the largest improvement in the makespan is obtained when interleaving job l with the job with the largest value of β. The optimal sequence for the remaining jobs can be determined by Theorem 1. □

Lemma 2 further shows that it is only enough to investigate the potential of interleaving when $0 < \beta_l < 0.5$. Given a pair of jobs l, k, Inequality (15) calculates a threshold for $\beta_k > 0.5$ such that an interleaving improves the makespan:

$$\beta_k \le \frac{1 + 4\beta_l}{1 - 2\beta_l}. \tag{15}$$

This leads to the following theorem.

Theorem 2. *In a two-job (l,k) instance of problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$, an interleaving reduces the makespan if $0.5 < \beta_k \le \frac{1+4\beta_l}{1-2\beta_l}, 0 < \beta_l < 0.5$.*

Proof. As discussed above. □

We note that Theorem 2 does not necessarily hold when $n \ge 3$. A counter example is shown in Figure 4. The optimal schedule for a three-job instance with $\beta_1 = 0.1$, $\beta_2 = 1$, $\beta_3 = 1.5$ and $p = 1$ does not follow Theorem 2, because the theorem implies that we may schedule an interleaving pair of jobs 1 and 3 at the beginning of the schedule since $\beta_3 < \frac{1+4\beta_1}{1-2\beta_1}$ (prioritizing job 3 to job 2 since $\beta_3 > \beta_2$ due to applying Lemma 1), followed by job 2. However, the optimal sequence is $(3, 1, 2)$.

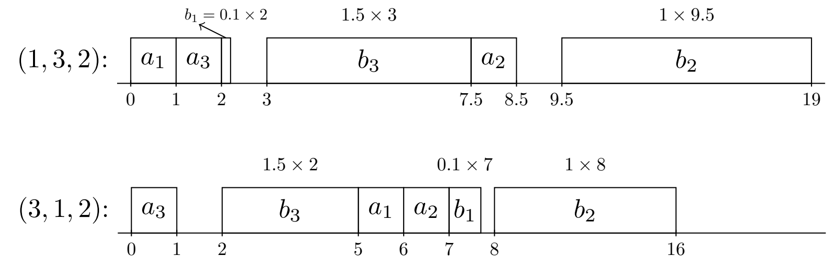

Figure 4: Counter example for generalizing the result of Theorem 2.

3.2 Groups of identical jobs

Although the computational complexity of problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$ under arbitrary values of β remains open, we now investigate a polynomially solvable case, in which jobs are partitioned into a set of $G = \{1, \ldots, m\}, |G| = m$ groups. An important characteristic of group $g \in G$ is that all of its jobs share the same processing rate denoted by β_g.

The simplest case includes only one group of jobs, i.e. $m = 1$, implying that all jobs are identical and in the form of $(a_j = p, L_j = p, b_j = \beta s_j)$. The problem can easily be solved because the sequence is immaterial. We further show this in Section 3.4. When $m > 1$, however, the number of all possible permutations of jobs grows exponentially. As an example, consider $m = 2$. For the simplicity, let n be an even number and let each group have an equal number of jobs. Therefore, $\frac{n}{2}$ jobs have a processing rate of β_1 and the remaining $\frac{n}{2}$ jobs have a processing rate of β_2. It is clear that the number of all possible permutations of jobs is equal to $\frac{n!}{\frac{n}{2}!\frac{n}{2}!}$. Next, we present a dynamic program for problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$, and show that DP runs in polynomial time when m is relatively small.

3.2.1 The dynamic programming algorithm

Let $i = 1, \ldots, n$ denote the current stage of the algorithm, where the total number of stages is equal to the number of jobs. At stage i the set of i first jobs is scheduled. Let π denote the set of jobs-group at stage i and j denote the last job scheduled in stage i. We denote by $z^i_{\pi,j} = (c^i_{\pi,j}, t^i_{\pi,j})$ the state of the system at stage i, where $c^i_{\pi,j}$ presents the completion time of i first jobs and $t^i_{\pi,j}$ represents two operations of "interleaving" (*int*) or "appending" (*app*) for the next job in the sequence. The recursive formula for $z^i_{\pi,j}, 1 \le i \le n-1$ is shown in Equation (16).

$$z^i_{\pi,j} = (c^i_{\pi,j}, t^i_{\pi,j}) = \begin{cases} (c^{i-1}_{\pi\setminus\{j\}} + b_j, app) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = int, \\ (c^{i-1}_{\pi\setminus\{j\}} + 2p + b_j, app) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app \wedge b_j > p, \\ (c^{i-1}_{\pi\setminus\{j\}} + 3p, int), (c^{i-1}_{\pi\setminus\{j\}} + 2p + b_j, app) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app \wedge b_j \le p. \end{cases} \tag{16}$$

where

$$b_j = \begin{cases} \beta_j (s^{i-1}_{\pi\setminus\{j\}}) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = int, \\ \beta_j (s^{i-1}_{\pi\setminus\{j\}} + 2p) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app. \end{cases} \tag{17}$$

Given that $c^{i-1}_{\pi\setminus\{j\}}$ represents the completion time of the $i-1$ first jobs and $s^{i-1}_{\pi\setminus\{j\}}$ and $t^{i-1}_{\pi\setminus\{j\}}$ denote the starting time and the operations "*int*" or "*app*" for the last job in the sequence of $i-1$ first jobs, then at each stage $i > 1$, we show the state of the system at the previous stage by $z^{i-1}_{\pi\setminus\{j\}}$.

The initial state is $(c^0_{\varnothing}, t^0_{\varnothing}) = \begin{cases} (0, int) \\ (0, app) \end{cases}$, and the final state is

$$z^n_{\pi,j} = c^n_{\pi,j} = \begin{cases} c^{n-1}_{\pi\setminus\{j\}} + b_j & \text{if } t^{n-1}_{\pi\setminus\{j\}} = int, \\ c^{n-1}_{\pi\setminus\{j\}} + 2p + b_j & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app. \end{cases} \tag{18}$$

The four possible cases for $z^i_{\pi,j}$ in Equation (16) are as follow. If $t^{i-1}_{\pi\setminus\{j\}} = int$, job j is interleaved with the last job in the sequence and the next job should be in the form of appending (case 1). If $t^{i-1}_{\pi\setminus\{j\}} = app$, job j will be appended, but the possibilities for the next job depend on the value of b_j. If $b_j > p$, the next job is also in the form of appending since it cannot be interleaved with job j (case 2). However, if $b_j \le p$, the next job can be interleaved with job j. We point that both interleaving and appending (cases 3 and 4, respectively) must be considered for the next job. Consider a three-job instance, where $\beta_1 = 0.25, \beta_2 = 0.20, \beta_3 = 0.17$ and $p = 1$. Figure 5 shows that although job 2 can be interleaved with job 1, instead, its appending leads to the optimal schedule (Figure 5a).

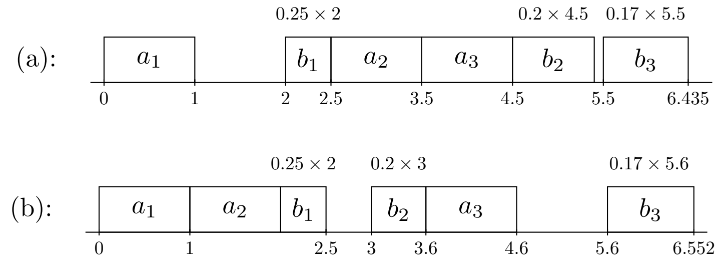

Figure 5: A three-job example showing that appending job 2 (a) leads to a smaller makespan than interleaving jobs 1 and 2 (b).

We note that at most three cases, out of four, need to be considered in any stage. Also, at each stage i, for any π, from the cases with similar $t^i_{\pi,j}$ the one with smaller $c^i_{\pi,j}$ is stored for the next stage. Therefore, in each stage at most two options of appending and interleaving may be possible. Next, we show that the proposed dynamic program can be solved in polynomial time when m is relatively small.

3.2.2 Complexity of the proposed dynamic program

We assume that jobs are partitioned into m groups. Let first consider the case that each group contains an equal number of jobs, which is $\frac{n}{m}$.

In stage i there is a number of candidates for π. Each candidate includes exactly i jobs. The number of candidates depends on both i and m because we only distinguish jobs by their group(s). In stage $i, 1 \le i \le \frac{n}{m}$, the number of candidates, which we denote by η, is equal to the number of solutions of Equation (19):

$$\sum_{m'=1}^{m} x_{m'} = i, \quad 0 \le x_{m'} \le \frac{n}{m}, \forall m' \in \{1, \dots, m\}. \tag{19}$$

Since both i and $x_{m'}$ are bounded from above by $\frac{n}{m}$, η is equal to $\binom{i+m-1}{i}$. In stage i, $\frac{n}{m} < i \le n$, η is still derived by using Equation (19), however, out of the total number of $\binom{i+m-1}{i}$ some are invalid. More precisely, because $i > \frac{n}{m}$ the solutions including $x_{m'} > \frac{n}{m}, \exists m' \in \{1, \dots, m\}$ are not considered, leading to a smaller value of η. Therefore, η in any stage i is not greater than $\binom{i+m-1}{i}$. Now consider the case where the groups do not contain an equal number of jobs. We can still derive η by using Equation (19), however, again some of the solution are invalid, and hence, η in stage i is never greater than $\binom{i+m-1}{i}$.

Additionally, in stage i there are at most m candidate jobs to occupy the last position because m groups of jobs exist. Also, at most three cases of appending or interleaving need to be considered. Hence, the time complexity of stage i is in the order of $O(m\binom{n+m-1}{n})$, implying that the time complexity of the proposed dynamic program is $O(nm\binom{n+m-1}{n})$, which is shown by Theorem 3 to be polynomial for small values of m.

Theorem 3. *The proposed dynamic program solves problem P1 with n jobs and m groups of identical jobs in $O(mn^m)$.*

Proof. The proof is by induction:

If $m = 2$, then $2n\binom{n+2-1}{n} = 2n(n+1) \approx O(n^2)$,
If $m = 3$, then $3n\binom{n+3-1}{n} = 3n(n+2)(n+1)/2 \approx O(n^3)$,
If $m = 4$, then $4n\binom{n+4-1}{n} = 4n(n+3)(n+2)(n+1)/6 \approx O(n^4)$,
$\ldots$.

In general, if m groups of identical jobs exist, the time complexity is $O(mn^m)$. We note that when $m = n$, the complexity of the dynamic program is $O(n^{n+1})$. □

Next, we explain a numerical example to illustrate the operation of the dynamic program.

3.2.3 A numerical example

Consider a four-job problem, where $G = \{1, 2\}$, $\beta_1 = 0.1, \beta_2 = 0.2$ and $p = 1$. Each group consists of an equal number of jobs.

Stage $i = 1$ includes only one job, and the two options for π include $\{k\}$ and $\{l\}$, where k represents jobs that belong to group 1 and l denotes jobs of group 2. It is clear that there is only one candidate for job j (the job in the last position in the sequence). Table 1 shows the calculations. Because there is one option for *app* and one for *int*, both will be stored by the algorithm for the next stage. Those are shown by an asterisk in Table 1.

Table 2 shows the calculations for stage $i = 2$. Three candidates for π include $\{k, k\}$, $\{k, l\}$ and $\{l, l\}$. In the second set both k and l may fill the last position. Also, because there are more than one option for *app* under $\{k, k\}$ and $\{l, l\}$, and more than one option for both *app* and *int* under $\{k, l\}$, the ones with the smaller value of makespan will be kept by the algorithm for the next stage.

Due to the example's assumption that there are an equal number of jobs in each group, in stage $i = 3$ we have only two candidates for π, which are $\{k, k, l\}$ and $\{k, l, l\}$. Table 3 shows the calculations. The algorithm proceeds to stage $i = 4$. There is a single candidate for π and $\pi = \{k, k, l, l\}$ since all jobs are considered. Table 4 shows that the optimal value of makespan is equal to 7.26. By starting from stage $i = 4$ and going backward we observe that the optimal solution is formed by first scheduling jobs of group 2, followed by jobs of group 1. Figure 6 depicts the optimal schedule.

Table 1: The dynamic programming calculations for stage $i = 1$ of the numerical example.

π	$\{k\}$	$\{l\}$
j	k	l
$z^0_\varnothing$	$\begin{cases}(0, int)\\(0, app)\end{cases}$	$\begin{cases}(0, int)\\(0, app)\end{cases}$
b_j	$\begin{cases}0.1(0) = 0\\0.1(0+2) = 0.2\end{cases}$	$\begin{cases}0.2(0) = 0\\0.2(0+2) = 0.4\end{cases}$
$z^1_{\pi,j}$	$\begin{cases}(0+3, int) = (3, int)^*\\(0+2+0.2, app) = (2.2, app)^*\end{cases}$	$\begin{cases}(0+3, int) = \mathbf{(3,\boldsymbol{int})}^*\\(0+2+0.4, app) = (2.4, app)^*\end{cases}$

Table 2: The dynamic programming calculations for stage $i = 2$ of the numerical example.

π	$\{k,k\}$	$\{k,l\}$		$\{l,l\}$
j	k	k	l	l
$z^1_{\pi\setminus\{j\}}$	$\begin{cases}(3, int)\\(2.2, app)\end{cases}$	$\begin{cases}(3, int)\\(2.4, app)\end{cases}$	$\begin{cases}(3, int)\\(2.2, app)\end{cases}$	$\begin{cases}(3, int)\\(2.4, app)\end{cases}$
b_j	$\begin{cases}0.1(3) = 0.3\\0.1(2.2+2) = 0.42\end{cases}$	$\begin{cases}0.1(3) = 0.3\\0.1(2.4+2) = 0.44\end{cases}$	$\begin{cases}0.2(3) = 0.6\\0.2(2.2+2) = 0.84\end{cases}$	$\begin{cases}0.2(3) = 0.6\\0.2(2.4+2) = 0.88\end{cases}$
$z^2_{\pi,j}$	$\begin{cases}(3+0.3, app) = (3.3, app)^*\\(2.2+3, int) = (5.2, int)^*\\(2.2+2+0.42, app) = (4.62, app)\end{cases}$	$\begin{cases}(3+0.3, app) = (3.3, app)^*\\(2.4+3, int) = (5.4, int)\\(2.4+2+0.44, app) = (4.84, app)\end{cases}$	$\begin{cases}(3+0.6, app) = (3.6, app)\\(2.2+3, int) = (5.2, int)^*\\(2.2+2+0.84, app) = (5.04, app)\end{cases}$	$\begin{cases}(3+0.6, app) = \mathbf{(3.6,\boldsymbol{app})}^*\\(2.4+3, int) = (5.4, int)^*\\(2.4+2+0.88, app) = (5.28, app)\end{cases}$

Table 3: The dynamic programming calculations for stage $i = 3$ of the numerical example.

π	$\{k,k,l\}$		$\{k,l,l\}$	
j	k	l	k	l
$z^2_{\pi\setminus\{j\}}$	$\begin{cases}(5.2, int)\\(3.3, app)\end{cases}$	$\begin{cases}(5.2, int)\\(3.3, app)\end{cases}$	$\begin{cases}(5.4, int)\\(3.6, app)\end{cases}$	$\begin{cases}(5.2, int)\\(3.3, app)\end{cases}$
b_j	$\begin{cases}0.1(5.2) = 0.52\\0.1(3.3+2) = 0.53\end{cases}$	$\begin{cases}0.2(5.2) = 1.04\\0.2(3.3+2) = 1.06\end{cases}$	$\begin{cases}0.1(5.4) = 0.54\\0.1(3.6+2) = 0.56\end{cases}$	$\begin{cases}0.2(5.2) = 1.04\\0.2(3.3+2) = 1.06\end{cases}$
$z^3_{\pi,j}$	$\begin{cases}(5.2+0.52, app) = (5.72, app)^*\\(3.3+3, int) = (6.3, int)^*\\(3.3+2+0.53, app) = (5.83, app)\end{cases}$	$\begin{cases}(5.2+1.04, app) = (6.24, app)\\(3.3+2+1.06, app) = (6.36, app)\end{cases}$	$\begin{cases}(5.4+0.54, app) = (5.94, app)^*\\(3.6+3, int) = \mathbf{(6.6, \boldsymbol{int})}^*\\(3.6+2+0.56, app) = (6.16, app)\end{cases}$	$\begin{cases}(5.2+1.04, app) = (6.24, app)\\(3.3+2+1.06, app) = (6.36, app)\end{cases}$

Table 4: The dynamic programming calculations for stage $i = 4$ of the numerical example

π	$\{k,k,l,l\}$	
j	k	l
$z^3_{\pi\setminus\{j\}}$	$\begin{cases}(6.6, int)\\(5.94, app)\end{cases}$	$\begin{cases}(6.3, int)\\(5.72, app)\end{cases}$
b_j	$\begin{cases}0.1(6.6) = 0.66\\0.1(5.94+2) = 0.794\end{cases}$	$\begin{cases}0.2(6.3) = 1.26\\0.2(5.72+2) = 1.544\end{cases}$
$z^4_{\pi,j}$	$\begin{cases}6.6+0.66 = \mathbf{7.26}\\5.94+2+0.794 = 8.734\end{cases}$	$\begin{cases}6.3+1.26 = 7.56\\5.72+2+1.544 = 9.264\end{cases}$

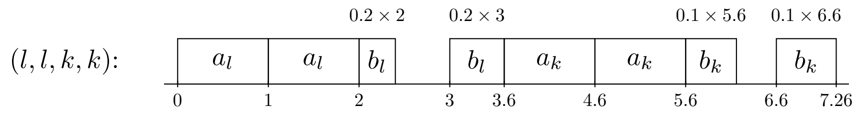

Figure 6: Optimal schedule of a four-job problem produced by the dynamic program.

3.3 The heuristic algorithm

In Section 3.1, we showed that in problem $(a_j = p, L_j = p, b_j = \beta_j s_j)$ the first priority must be given to jobs with greater values of β_j (implied by Theorem 1). An interleaving of jobs, however, may potentially decrease the makespan if Theorem 2 holds. Therefore, when constructing a schedule the only two available options at any point include (1) appending a single job, or (2) interleaving a pair of jobs. We utilize those principles and develop a heuristic algorithm for problem P1. The proposed heuristic first constructs a schedule (see Algorithm 1), and then iteratively improves the schedule (see Algorithm 2).

Let $T = J$ be the set of unscheduled jobs and $S = ()$ be the sequence of performing jobs. Each iteration of the constructive heuristic consists of identifying a single job to be appended, or a pair of jobs to be interleaved. Let assume that the jobs can start at time zero. Therefore, the start time of the completion task in the first iteration is $s_1 = 2p$. At every iteration $i \geq 1$, a threshold on β is calculated: $\beta_{thr} = \frac{p}{s_i}$ (the threshold is used to identify jobs with $b_j \leq p$). The subset of jobs with $\beta_j \leq \beta_{thr}$ are identified as the jobs that can be the first of a potential interleave. From those, the job with the largest value of β is selected. Let l denote this job. The other job, say k, is then selected such that it has the largest value of β among all jobs.

Next, it is checked whether job k satisfies the bound $\beta_k \leq \frac{1+4\beta_l}{1-2\beta_l}$. If so, the interleaving pair of jobs l and k is scheduled, where job l is the first job of the interleaving pair. Otherwise, job k is appended to S. At the end of each iteration, the makespan, i.e. the start time of the next completion task, and S and T are updated. The procedure continues until all jobs are scheduled, or no interleaving is possible, i.e. $b_j \nleq p, j \in T$. In this case, the remaining jobs are appended to S in a non-increasing order of β_j.

Algorithm 1: The construction procedure of the heuristic algorithm.

```
1  Input: $S = (), T = J, p, \beta_j, \forall j \in J, s_1 = 2p$.
2  Output: A sequence $S$ with makespan $C_S$.
3  for $i = 1$ to $n$ do
4      $\beta_{thr} = \frac{p}{s_i}$;
5      if $\exists j \in T, \beta_j \leq \beta_{thr}$ then
6          $l \leftarrow \arg\max_{j \in T}(\beta_j | \beta_j \leq \beta_{thr})$;
7          $k \leftarrow \arg\max_{j \in T}(\beta_j)$;
8          if $\beta_k \leq \frac{1+4\beta_l}{1-2\beta_l}$ then
9              Interleave jobs $l$ and $k$ adjacently;
10             $s_{i+1} = s_i + (\beta_k)(s_i + p) + 3p$;
11             $S \leftarrow S \cup \{l, k\}$;
12             $T \leftarrow T \setminus \{l, k\}$;
13         else
14             Append job $k$ adjacently;
15             $s_{i+1} = s_i + (\beta_k)s_i + 2p$;
16             $S \leftarrow S \cup \{k\}$;
17             $T \leftarrow T \setminus \{k\}$;
18         end
19     else
20         Break;
21     end
22 end
23 Adjacently append the remaining jobs in $T$ to $S$, in a non-increasing order of $\beta_j$;
24 return $S$;
```

By utilizing the delay periods, Algorithm 1 constructs as many interleaving pairs as possible, while it gives higher priority to the jobs with larger value of β. The total number of iterations performed by the algorithm is at most equal to the number of jobs. Because finding jobs l and k in each iteration requires $O(n)$ time, the algorithm therefore has a time complexity of $O(n^2)$.

We now present an example to clarify the operation of Algorithm 1. Consider four jobs with β values of $\{0.1, 0.15, 0.18, 3.0\}$ and $p = 1$. We initialize $T = \{1, 2, 3, 4\}$ and $S = ()$. Assuming that we start at time zero, then $s_1 = 2$. Table 5 shows that the algorithm appends job 4 in the first iteration. In the second iteration, jobs 1 and 3 are interleaved, where job 1 is the first job of the interleaving pair. Then, because the condition in line 5 of Algorithm 1 is not satisfied, the loop is terminated and the remaining job, i.e. job 2 is appended. The Gantt chart depicted in Figure 7 shows the schedule delivered by Algorithm 1, which is indeed the optimal schedule.

Table 5: The operation of Algorithm 1 for a four-job instance.

Step i	β_{thr}	l	k	S	T	s_{i+1}
1	$\frac{1}{2}$	3	4	(4)	$\{1, 2, 3\}$	10
2	$\frac{1}{10}$	1	3	$(4, 1, 3)$	$\{2\}$	14.98
3	$\frac{1}{14.98}$	-	-	-	-	-

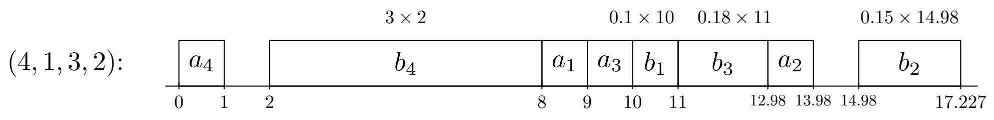

Figure 7: The sequence and schedule for a four-job instance delivered by Algorithm 1.

The schedule obtained by Algorithm 1 may further be improved. To do so, we iteratively apply swap moves. This is presented in Algorithm 2. We implement the "first improvement" criterion, i.e. once an improving solution is obtained it is accepted and the schedule is updated. It is clear that the run time of Algorithm 2 is $O(n^2)$. Therefore, the run time of the proposed heuristic is $O(n^2)$.

Algorithm 2: The improvement procedure of the heuristic algorithm.

```
1  Input: $\beta_j, \forall j \in J$, $S_0$, $C_{S_0}$, $j = 1$.
2  Output: A sequence $S$ with makespan $C_S$.
3  $S = S_0$;
4  $C_S = C_{S_0}$;
5  while $j \leq n - 1$ do
6      $Improve = 0$;
7      for $k = j + 1 : n$ do
8          $S' \leftarrow$ swap$(j, k)$;
9          $C_{S'} \leftarrow$ makespan$(S')$;
10         if $C_{S'} < C_S$ then
11             $S = S'$;
12             $C_S = C_{S'}$;
13             $Improve = 1$;
14         end
15     end
16     if $Improve = 0$ then
17         $j = j + 1$;
18     end
19 end
20 return $S$;
```

3.4 Lower bound

We derive a lower bound for problem P1 by letting $\beta_j = \min_{j \in J} \beta_j, \forall j \in J$, i.e. all jobs have an identical processing rate. Theorem 4 shows this.

Theorem 4. *Optimizing problem P1 under the setting $\beta_{min} = \min_{j \in J} \beta_j$ leads to a makespan, which is never greater than the makespan under the arbitrary values for β.*

Proof. Assume that the makespan under the setting $\beta_{min} = \min_{j \in J} \beta_j$ is greater than the makespan under the arbitrary values for β. Then, there exits an optimal makespan where $\beta_j > \beta_{min}, j \in J$. Because the initial task and the delay period take identical values for all jobs, the makespan under $\beta_j > \beta_{min}, j \in J$ is never less than the one under β_{min}, implying that the initial assumption is contradicted. □

Next, we show that obtaining this lower bound is trivial.

Lemma 3. *Under the setting $\beta_j = \beta_{min}, \forall j \in J$ the makespan for problem P1 is minimized if the sequence includes a number of adjacent interleaving pairs followed by appending the remaining jobs once no interleaving is possible (sequence 1), or if the sequence includes appending a single job at the beginning, followed by a number of adjacent interleaving pairs and then appending the remaining jobs (sequence 2).*

Proof. We note that the schedule with the minimum makespan consist of as many interleaving pairs (of jobs) as possible. Intuitively, this implies that a number of adjacent interleaving pairs followed by appending the remaining jobs once no interleaving is possible must lead to the minimum makespan. We show that in some cases a better makespan (with smaller value) is obtained if we first append a single job, and then add a set of interleaving pairs followed by a set of appending jobs. We note that interleaving is possible as long as the completion task of the first job (of an interleaving pair) starts no later than $\frac{p}{\beta_{min}}$. Let illustrate sequences 1 and 2 by a three-job example, where $p = 1$ and $\beta_{min} = 0.1$. Sequence 1 consists of one interleaving pair, followed by appending the third job. This results in $C_{max} = 5.83$ (see Figure 8a). In sequence 2, the third job is scheduled before the interleaving pair. This results in $C_{max} = 5.72$, i.e. the minimum makespan (see Figure 8b).

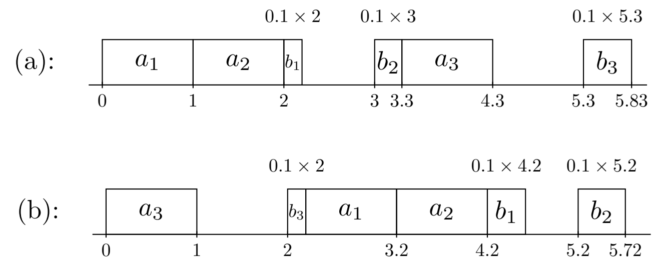

Figure 8: A three-job instance to illustrate calculation of the lower bound for problem P1.

There is no possibility to append two (or more) jobs at the beginning of the sequence because an interleaving pair of those jobs would complete earlier than appending them adjacently. □

4 Computational results

We evaluate the performance of the proposed heuristic on a set of 120 randomly generated instances. The instances include 5, 10, 20, 50, 75 and 100 jobs (n). We set the parameter β in a way to allow some interleaving in the schedule. Since large values of β result in less possibility for interleaving, and hence, easier instances, we therefore consider two settings. For the first setting, we randomly select β from the continuous uniform distribution such that $\beta_j \in (0, 0.1), \forall j \in J$, and for the second setting $\beta_j \in (0, 0.2), \forall j \in J$. We generated 10 instances for each combination of n and β. This results in 120 instances in total. We set $p = 1$ for all instances.

We also solve the instances by optimizing problem P1 with the solver Gurobi version 8.0.0 (Gurobi Optimization, 2018). We implement problem P1 and the heuristic algorithm in the programming language Python version 2.7. We perform all computational experiments on a PC with Intel® Core^{TM} i5-7500 CPU clocked at 3.40GHz with 8GB of memory under Linux Ubuntu 18.04 operating system. We set a time limit of 3600 seconds for the solver Gurobi. We utilize one processor (thread) for the heuristic algorithm, however, we run the Gurobi by using one processor and four processors (denoted as Gurobi^1 and Gurobi^4). For the remaining parameters of the solver Gurobi we used the default values.

Table 6 reports the outcomes of the heuristic algorithm, denoted as "Heur$_{cons}$" and Gurobi. We use two criteria of "Feasible" and "Optimal", which denote the number of feasible and optimal solutions, respectively, obtained by the heuristic and Gurobi in order to evaluate the performance of the methods. According to the results, Gurobi^1 and Gurobi^4 generate feasible solution for only 71 instances, out of 120 (i.e. for almost 59%). Within 3600 seconds of running, Gurobi reports feasible solution for only one instance with 75 jobs and $\beta_j \in (0, 0.1)$; it also does not report feasible solution for the instances with 50 jobs and $\beta_j \in (0, 0.2)$. For the instances with 100 jobs, Gurobi runs out of memory. The performance of Gurobi^4 is slightly better than that of Gurobi^1 since it obtains three additional optimal solutions. The proposed heuristic, however, delivers feasible solution for all instances. Interestingly, the heuristic produces the same best solutions for 18 of those instances, i.e. for 45%.

Table 6: Number of feasible and optimal solutions delivered by Heur$_{cons}$ and Gurobi.

n	Setting for β	Feasible			Optimal		
		Heur$_{cons}$	Gurobi^1	Gurobi^4	Heur$_{cons}$	Gurobi^1	Gurobi^4
5	(0, 0.1)	10	10	10	0	10	10
	(0, 0.2)	10	10	10	2	10	10
10	(0, 0.1)	10	10	10	9	7	10
	(0, 0.2)	10	10	10	7	10	10
20	(0, 0.1)	10	10	10	0	0	0
	(0, 0.2)	10	10	10	0	0	0
50	(0, 0.1)	10	10	10	0	0	0
	(0, 0.2)	10	0	0	0	0	0
75	(0, 0.1)	10	1	1	0	0	0
	(0, 0.2)	10	0	0	0	0	0
100	(0, 0.1)	10	0	0	0	0	0
	(0, 0.2)	10	0	0	0	0	0
Total		120	71	71	18	37	40

Table 7 reports two criteria of "Gap (%)" and "Time (sec)" (computation time in seconds), which are averaged over 10 instances per setting (either heuristic or Gurobi). The gap is calculated as $\frac{z-z^*}{z^*} \times 100$, where z is the objective function value, i.e. the makespan delivered by the method, and z^* is the best objective function value between the heuristic and Gurobi. The gap measures proximity of a solution obtained by the method from the best available one. Consistent with earlier findings, for small instances with 5 and 10 jobs the solver Gurobi outperforms the proposed heuristic. For larger instances, however, the heuristic delivers improved solutions. Particularly, we note that both versions of Gurobi have a gap of 75.42% and 77.16% for instances with 50 and 75 jobs, not to mention that because Gurobi is not able to report any feasible solution for four groups of instances, the value of gap cannot be calculated for those instances ("-" in Table 7 shows this).

Table 8 summarizes the outcomes of Heur$_{cons}$ and Gurobi^1 and Gurobi^4. The highlighted values denote the superiority of the method with respect to the criterion. As the table shows, the proposed heuristic performs very well, and obtains high quality solutions: its average gap is 0.30%, while its worst gap is nearly 1.22%. In addition, it is very efficient since it solves even the problems with 100 jobs within three seconds. The average time of both Gurobi^1 and Gurobi^4 is almost 40 minutes, and significantly increases with the number of jobs.

To further evaluate the performance of the proposed heuristic, i.e., Heur$_{cons}$, we compare the values of its gap to the lower bound and those of the solver Gurobi. We report the outcomes in Table 9, where the values of gap are averaged over 10 instances per setting. The gap is calculated as $\frac{z-lb}{lb} \times 100$, where

Table 7: Gap from the best obtained solution, and the computation time for Heur$_{cons}$ and Gurobi.

n	Setting for β	Gap (in %)			Time		
		Heur$_{cons}$	Gurobi^1	Gurobi^4	Heur$_{cons}$	Gurobi^1	Gurobi^4
5	(0, 0.1)	1.08	0.00	0.00	< 0.01	0.20	0.18
	(0, 0.2)	1.13	0.00	0.00	< 0.01	0.18	0.16
10	(0, 0.1)	0.01	0.00	0.00	< 0.01	3167.36	1526.39
	(0, 0.2)	1.22	0.00	0.00	< 0.01	443.31	199.38
20	(0, 0.1)	0.00	6.52	4.86	0.02	3600.00	3600.03
	(0, 0.2)	0.14	4.45	3.81	0.02	3600.01	3600.03
50	(0, 0.1)	0.00	75.42	75.42	0.25	3600.01	3600.07
	(0, 0.2)	0.00	-	-	0.31	3600.02	3600.02
75	(0, 0.1)	0.00	77.16	77.16	0.92	3600.15	3600.26
	(0, 0.2)	0.00	-	-	1.17	3600.28	3600.39
100	(0, 0.1)	0.00	-	-	2.36	-	-
	(0, 0.2)	0.00	-	-	2.79	-	-

Table 8: Overall results for Heur$_{cons}$ and Gurobi.

Method	Feasible	Optimal	Gap (%)		Time (sec)	
			Ave	Max	Ave	Max
Heur$_{cons}$	**120**	18	**0.30**	**1.22**	**0.65**	**2.79**
Gurobi^1	71	37	13.25	77.16	2521.18	3600.28
Gurobi^4	71	**40**	12.93	77.16	2332.69	3600.39

z is the objective function value, i.e., the makespan delivered by the method, and lb is the lower bound obtained via procedure explained in Section 3.4. We report the results only for instances with $n = 5, 10$ because proven optimal solutions are available only for these instances. The results indicate that Heur$_{cons}$ performs very closely to Gurobi because its average values of gap to the lower bound is very close to those of Gurobi.

Table 9: Gap to the lower bound for Heur$_{cons}$ and Gurobi.

n	Setting for β	Heur$_{cons}$	Gurobi^1	Gurobi^4
5	(0, 0.1)	2.55	1.50	1.50
	(0, 0.2)	6.73	5.67	5.67
10	(0, 0.1)	3.72	3.71	3.71
	(0, 0.2)	23.34	22.58	22.58
Average		9.08	8.36	8.36

Because Gurobi cannot deliver feasible solutions for large instances, we further assess the performance of the proposed heuristic, i.e., Heur$_{cons}$ by solving the instances with two new settings and comparing the outcomes of Heur$_{cons}$ and those of the two settings. For this purpose, we generate initial solutions via sorting the jobs in non-increasing and non-decreasing orders of their β values that results in two new variants for the heuristic, denoted as "Heur$_{LPT}$", "Heur$_{SPT}$", respectively. We summarize the results in Table 10, where the metric best denotes the number of best solutions obtained by each setting. The results show that Heur$_{cons}$ obtains significantly better solutions than those two variants of Heur$_{LPT}$ and Heur$_{SPT}$. Indeed, Heur$_{cons}$ obtains the best solution in 108 instances. The average gap of Heur$_{cons}$ over all instances is almost 0.44%, that is much lower than the average gap of the two settings of Heur$_{LPT}$ and Heur$_{SPT}$. Those results further indicate the quality of solutions produced by the proposed heuristic.

5 Conclusion

We investigated the single machine coupled task scheduling problem where the processing time of initial tasks and the delay periods have identical values, and the processing time of completion tasks follow a time-dependent characteristic. We showed that the optimal schedule can be obtained under certain conditions. Also, we proposed a dynamic program that can solve the problem in polynomial time if jobs can be grouped with respect to their processing rates, and the number of groups is not large. For general

Table 10: Assessing the performance of Heur$_{cons}$, Heur$_{LPT}$ and Heur$_{SPT}$.

n	Setting for β	Heur$_{cons}$		Heur$_{LPT}$		Heur$_{SPT}$	
		Gap (in %)	Best	Gap (in %)	Best	Gap (in %)	Best
5	(0, 0.1)	1.08	10	1.08	10	1.08	10
	(0, 0.2)	2.01	10	2.01	10	2.01	10
10	(0, 0.1)	0.01	9	0.01	9	0.04	9
	(0, 0.2)	1.22	8	3.10	5	2.48	5
20	(0, 0.1)	0.52	6	1.11	7	1.91	2
	(0, 0.2)	0.40	9	3.13	2	9.80	0
50	(0, 0.1)	0.02	8	2.23	2	10.95	0
	(0, 0.2)	0.00	10	5.71	0	6.88	0
75	(0, 0.1)	0.00	10	2.86	0	10.23	0
	(0, 0.2)	0.00	9	4.94	1	10.43	0
100	(0, 0.1)	0.07	9	2.87	1	7.09	0
	(0, 0.2)	0.00	10	4.61	0	12.61	0
Average / total		**0.44**	**108**	2.81	47	6.29	36

case, the computational complexity of the problem still remains open, and we therefore developed certain theoretical results and utilized those in a very efficient heuristic algorithm. Particularly, in large instances where the solver Gurobi is unable to generate even feasible solutions, the proposed heuristic is shown to perform very well and obtains quality solutions very quickly.

A direction for future research includes studying the problem where the delay duration is time-dependent. Also, it is interesting to investigate the identical case, i.e. $(a_j = a, L_j = L, b_j = b)$ with time-dependent processing times.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable suggestions and comments. Mostafa Khatami is the recipient of the UTS International Research Scholarship (IRS) and the UTS President's Scholarship (UTSP). Amir Salehipour is the recipient of an Australian Research Council Discovery Early Career Researcher Award (project number DE170100234) funded by the Australian Government.

References

Ageev, A. A. (2018). "Inapproximately lower bounds for open shop problems with exact delays". *Approximation and Online Algorithms.* Springer International Publishing AG, 45–55.

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

Ageev, A. A. and Kononov, A. V. (2007). "Approximation algorithms for scheduling problems with exact delays". *Approximation and Online Algorithms.* Springer Berlin Heidelberg.

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

Azadeh, A., Farahani, M. H., Torabzadeh, S, and Baghersad, M. (2014). "Scheduling prioritized patients in emergency department laboratories". *Computer Methods and Programs in Biomedicine* 117(2), 61–70.

Baptiste, P. (2010). "A note on scheduling identical coupled tasks in logarithmic time". *Discrete Applied Mathematics* 158(5), 583 –587.

Békési, J., Galambos, G., Jung, M. N., Oswald, M., and Reinelt, G. (2014). "A branch-and-bound algorithm for the coupled task problem". *Mathematical Methods of Operations Research* 80(1), 47–81.

Bessy, S. and Giroudeau, R. (2019). "Parameterized complexity of a coupled-task scheduling problem". *Journal of Scheduling* 22(3), 305–313.

Blazewicz, J., Ecker, K., Kis, T., Potts, C. N., Tanas, M., and Whitehead, J. (2010). “Scheduling of coupled tasks with unit processing times”. *Journal of Scheduling* 13(5), 453–461.

Cheng, T. C. E., Ding, Q., and Lin, B. M. T. (2004). “A concise survey of scheduling with time-dependent processing times”. *European Journal of Operational Research* 152, 1 –13.

Condotta, A. and Shakhlevich, N. (2012). “Scheduling coupled-operation jobs with exact time-lags”. *Discrete Applied Mathematics* 160(16), 2370 –2388.

Condotta, A. and Shakhlevich, N. (2014). “Scheduling patient appointments via multilevel template: A case study in chemotherapy”. *Operations Research for Health Care* 3(3), 129 –144.

Gawiejnowicz, S. (2008). *Time-dependent scheduling.* Springer Science & Business Media.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. R. (1979). “Optimization and approximation in deterministic sequencing and scheduling: A survey”. *Annals of Discrete Mathematics* 5, 287 –326.

Gupta, J. N. D. and Gupta, S. K. (1988). “Single facility scheduling with nonlinear processing times”. *Computers & Industrial Engineering* 14(4), 387 –393.

Gurobi Optimization, L. (2018). *Gurobi Optimizer Reference Manual.*

Hwang, F. J. and Lin, B. M. T. (2011). “Coupled-task scheduling on a single machine subject to a fixed-job-sequence”. *Computers & Industrial Engineering* 60(4), 690 –698.

Khatami, M. and Salehipour, A. (2020). “A binary search algorithm for the general coupled task scheduling problem”. *4OR*, 1–19.

Khatami, M., Salehipour, A., and Cheng, T. C. E. (2020). “Coupled task scheduling with exact delays: Literature review and models”. *European Journal of Operational Research* 282(1), 19 –39.

Kunnathur, A. S. and Gupta, S. K. (1990). “Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem”. *European Journal of Operational Research* 47(1), 56 –64.

Legrain, A., Fortin, M.-A., Lahrichi, N., Rousseau, L.-M., and Widmer, M. (2015). “Stochastic optimization of the scheduling of a radiotherapy center”. *Journal of Physics: Conference Series.* Vol. 616. 1. IOP Publishing, 012008.

Lehoux-Lebacque, V., Brauner, N., and Finke, G. (2015). “Identical coupled task scheduling: polynomial complexity of the cyclic case”. *Journal of Scheduling* 18(6), 631–644.

Leung, J. Y.-T., Li, H., and Zhao, H. (2007). “Scheduling two-machine flow shops with exact delays”. *International Journal of Foundations of Computer Science* 18(02), 341–359.

Li, H. and Zhao, H. (2007). “Scheduling Coupled-Tasks on a Single Machine”. *IEEE Symposium on Computational Intelligence in Scheduling*, 137–142.

Liu, Z., Lu, J., Liu, Z., Liao, G., Zhang, H. H., and Dong, J. (2019). “Patient scheduling in hemodialysis service”. *Journal of Combinatorial Optimization* 37(1), 337–362.

Marinagi, C. C., Spyropoulos, C. D., Papatheodorou, C., and Kokkotos, S. (2000). “Continual planning and scheduling for managing patient tests in hospital laboratories”. *Artificial Intelligence in Medicine* 20(2), 139–154.

Mosheiov, G. (1994). “Scheduling jobs under simple linear deterioration”. *Computers & Operations Research* 21(6), 653 –659.

Orman, A. and Potts, C. (1997). “On the complexity of coupled-task scheduling”. *Discrete Applied Mathematics* 72(1), 141 –154.

Pérez, E., Ntaimo, L., Wilhelm, W. E., Bailey, C., and McCormack, P. (2011). “Patient and resource scheduling of multi-step medical procedures in nuclear medicine”. *IIE Transactions on Healthcare Systems Engineering* 1(3), 168–184.

Pérez, E., Ntaimo, L., Malavé, C. O., Bailey, C., and McCormack, P. (2013). “Stochastic online appointment scheduling of multi-step sequential procedures in nuclear medicine”. *Health Care Management Science* 16(4), 281–299.

Shapiro, R. D. (1980). “Scheduling coupled tasks”. *Naval Research Logistics Quarterly* 27(3), 489–498.

Sherali, H. D. and Smith, J. C. (2005). “Interleaving two-phased jobs on a single machine”. *Discrete Optimization* 2(4), 348 –361.

Simonin, G., Darties, B., Giroudeau, R., and König, J.-C. (2011). “Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor”. *Journal of Scheduling* 14(5), 501–509.

Yu, W., Hoogeveen, H., and Lenstra, J. K. (2004). “Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard”. *Journal of Scheduling* 7(5), 333–348.