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. We provide optimal solution under certain conditions, and propose a heuristic for the general case. The numerical study shows that the heuristic performs very well.

Keywords: single machine coupled task scheduling; time-dependent processing time; simple linear processing time; heuristic

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. We assume a_j, L_j, and b_j only take positive integers values.

The coupled task scheduling problem was introduced by Shapiro (1980) in order to formulate a pulsed radar system, where 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 (Brauner et al., 2009), 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). Extensions of the problem including additional constraints such as precedence constraints (Blazewicz et al., 2010), compatibility constraints (Simonin et al., 2011) and fixed-job-sequence constraints (Hwang and Lin, 2011) have also been investigated.

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

*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

the problem. The interleaving heuristic aims at sequencing 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).

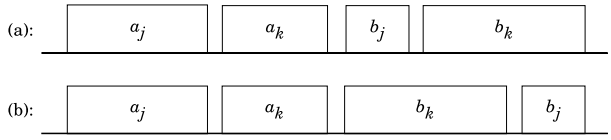

Figure 1: Interleaving jobs j and k (a), and nesting jobs j and k (b).

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 $\alpha_j \geqslant 0$ and a processing rate $\beta_j \geqslant 0$. 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 $1|(a_j = p, L_j = p, b_j = \beta_j s_j)|C_{max}$.

The remaining 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 heuristic is 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. (2019) discussed that the model proposed by Békési et al. (2014) is 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 this formulation.

Problem P1

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

subject to

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

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

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

$$x_{h,h'} + x_{h',h''} + x_{h'',h} \leqslant 2, \quad \text{for any triple distinct tasks:} \quad h, h', h'' \in H \tag{5}$$

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

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

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

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

$$s_h \geqslant 0, \quad 1 \leqslant h \leqslant 2n \tag{10}$$

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

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 is considered for the starting time of the completion tasks in constraints (7) (UB denotes the upper bound). Constraints (8) and (9) relate the starting time of tasks, as well as relating the starting time variables to the linear ordering variables. Constraints (10) and (11) ensure that the decision variables are non-negative and binary.

3 Minimizing the makespan

We show that the optimal schedule for problem P1 can easily be constructed for these two cases: (1) $\beta_j > 0.5, \forall j \in J$, and (2) a two-job instance of P1. 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. In addition, the delay period, i.e. $L_j = p$ is not large enough to allow interleaving of more than two jobs. For a pair of jobs j and k, if $b_j \leqslant p$, it is possible to interleave jobs j and k, where j is the first 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 P1, i.e. $(a_j = p, L_j = p, b_j = \beta_j s_j)$.

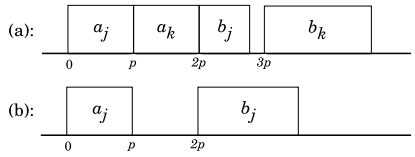

Figure 2: Contribution of an interleaving pair of jobs (a), and a single job (b) to the makespan.

Optimal schedule under the single machine setting 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 starting time of the schedule, and 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 P1. We note that the combination of the initial task and the delay period of a job (say j) can be considered as the normal processing time of the job, i.e. $\alpha_j = p + p = 2p$. If no interleaving is possible, problem P1 may reduce 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 whether interleaving is possible.

Assume that the first job starts at time zero (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) $\beta_j \leqslant 0.5, \exists j \in J$. The following theorem leads to the optimal schedule if $\beta_j > 0.5, \forall j$.

Theorem 1. *The minimum (optimal) makespan for problem P1 is obtained when jobs are ordered 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. Then its completion task starts at time $2p$, and hence, $b_j = \beta_j \times 2p$. 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). Therefore, the optimal sequence is obtained by ordering jobs in a non-decreasing order of α_j/β_j (equivalently in a non-increasing order of β_j since $\alpha_j = 2p > 0, \forall j$). □

The proof of Theorem 1 shows that although the actual processing time of jobs depends on the starting time of the completion tasks, this does not impact the optimal order because $\alpha_j = 2p, \forall j$. 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 β_j, i.e. $\beta_j \leqslant 0.5, \exists j \in J$, the jobs in the subset $\bar{J} \subset J$, where $\beta_j > 0.5, \forall j \in \bar{J}$, appear in the optimal schedule in a non-increasing order of $\beta_j, j \in \bar{J}$.*

Proof. Let jobs $j, k \in \bar{J}$ be two jobs with $\beta_j > \beta_k > 0.5$. Assume job k precedes job j in the optimal schedule. It is easy to see that swapping jobs j and k decreases the makespan, which is a contradiction to the optimal order. 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. □

For the case of $\beta_j \leqslant 0.5, \exists j \in J$, there might be some possibility for interleaving of jobs. The following scenario shows the impact of interleaving two jobs on the makespan. Let $J_l = (p, p, b_l = \beta_l s_l)$ and $J_k = (p, p, b_k = \beta_k s_k)$ be a two-job instance of problem P1, where $\beta_l \leqslant 0.5, \beta_k > 0.5$ (i.e. $\beta_l < \beta_k$). Let the schedule start at time zero, and hence, the first completion task starts at time $2p$. Because $\beta_l \leqslant 0.5$, then $\beta_l(2p) \leqslant p$, implying that the jobs can be interleaved if the schedule starts with J_l. On the contrary, because $\beta_k > 0.5$, and therefore $\beta_k(2p) \nleqslant p$, the interleaving of jobs is not possible if the schedule starts with J_k. The Gantt chart of Figure 3 illustrates those two cases. The makespan for those schedules can be derived as follows.

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

$$(J_k, J_l) : C_{(2)} = p + p + 2p\beta_k + p + p + (4p + 2p\beta_k)\beta_l = 4p + 2p\beta_k + (4p + 2p\beta_k)\beta_l \tag{13}$$

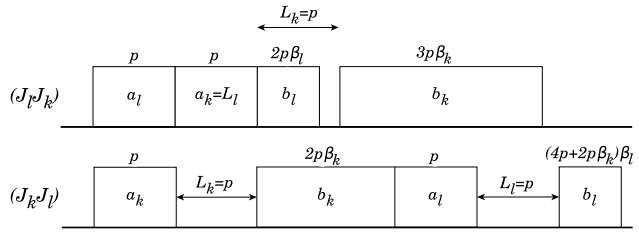

Figure 3: Two possible schedules for a two-job instance of problem P1: (J_l, J_k), where interleaving occurs, and (J_k, J_l), where interleaving is not possible.

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

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

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

Proof. Interleaving job l with a job k, where $\beta_k > 0.5$ 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 job k with the largest β_k. 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 $0 < \beta_l < 0.5$. Inequality (15) calculates a threshold for $\beta_k > 0.5$ such that an interleaving improves the makespan:

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

This leads to the following theorem.

Theorem 2. *In a two-job (l,k) instance of problem P1, an interleaving reduces the makespan if $\beta_k \leqslant \frac{1+4\beta_l}{1-2\beta_l}, 0 < \beta_l < 0.5, \beta_k > 0.5$.*

Proof. As discussed above. □

It should be noted that Theorem 2 does not necessarily hold when $n \geqslant 3$. A counter example is shown in Figure 4. The optimal schedule for a three-job problem 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 J_1 and J_3 at the beginning of the schedule since $\beta_3 < \frac{1+4\beta_1}{1-2\beta_1}$ (prioritizing J_3 to J_2 since $\beta_3 > \beta_2$ due to applying Lemma 1), followed by job J_2. Surprisingly, the optimal schedule is (J_3, J_1, J_2).

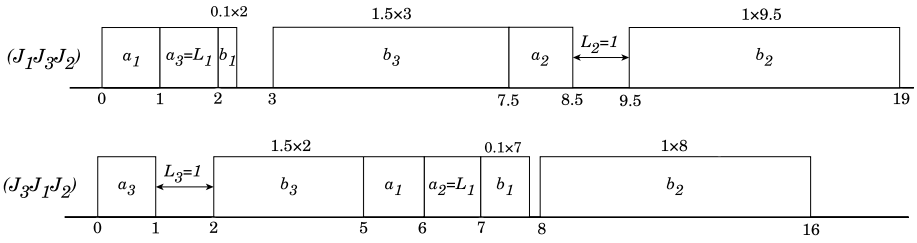

Figure 4: Counter example for generalizing the result of Theorem 2.

To conclude, in problem P1, i.e. with simple linear completion tasks, 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. Next, we utilize those principles and develop a heuristic algorithm for problem P1.

3.2 The heuristic algorithm

The proposed heuristic for problem P1 first constructs a schedule (Algorithm 1), and then iteratively improves the so obtained schedule (Algorithm 2).

Let $T = J$ be the set of unscheduled jobs and $S = ()$ be the sequence of performing scheduled jobs. Each iteration of the constructive heuristic (Algorithm 1) consists of identifying a single job to be appended, or a pair of jobs to be interleaved. Without loss of generality, assume that the jobs can start at time zero. Therefore, the starting time of the completion task in the first iteration is $s_1 = 2p$. At each iteration $i \geqslant 1$, a threshold on the time-dependent parameter is calculated: $\beta_{thr} = \frac{p}{s_i}$ (the threshold is used to identify jobs with $b_j \leqslant p$). The subset of jobs with $\beta_j \leqslant \beta_{thr}$ are identified as the jobs that can be the first of a potential interleave. From those, the job with the largest β_j is selected. Let l denote this job. Job 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 $\beta_k \leqslant \frac{1+4\beta_l}{1-2\beta_l}$. 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 partial makespan, i.e. the starting 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. $b_j \nleqslant p, j \in T$. In this case, the remaining jobs are appended to S in a non-increasing order of β_j.

By utilizing the delay periods, Algorithm 1 constructs as many interleaving pairs as possible, while it gives more 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 $O(n^2)$.

Algorithm 1: The construction procedure of the heuristic algorithm.

Input: $S = (), T = J, s_1 = 2p, p, \beta_j, \forall j \in J$.
Output: A schedule S with makespan C_S.

for $i = 1 : n$ **do**
  $\beta_{thr} = \frac{p}{s_i}$;
  if $\exists j \in T, \beta_j \leqslant \beta_{thr}$ **then**
    $l \leftarrow \arg\max_{j \in T}(\beta_j | \beta_j \leqslant \beta_{thr})$;
    $k \leftarrow \arg\max_{j \in T}(\beta_j)$;
    if $\beta_k \leqslant \frac{1+4\beta_l}{1-2\beta_l}$ **then**
      Interleave jobs l and k adjacently;
      $s_{i+1} = s_i + (\beta_k)(s_i + p) + 3p$;
      $S \leftarrow S \cup \{l, k\}$;
      $T \leftarrow T \setminus \{l, k\}$;
    else
      Append job k adjacently;
      $s_{i+1} = s_i + (\beta_k)s_i + 2p$;
      $S \leftarrow S \cup \{k\}$;
      $T \leftarrow T \setminus \{k\}$;
    end
  else
    Break;
  end
end
Adjacently append the remaining jobs in T to S, in a non-increasing order of β_j;
return S;

Schedule S 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.

Input: $\beta_j, \forall j \in J$, S_0, C_{S_0}, $j = 1$.
Output: A schedule S with makespan C_S.

$S = S_0$;
$C_S = C_{S_0}$;
while $j \leqslant n - 1$ **do**
  $Improve = 0$;
  for $k = j + 1 : n$ **do**
    $S' \leftarrow \text{swap}(j, k)$;
    $C_{S'} \leftarrow \text{makespan}(S')$;
    if $C_{S'} < C_S$ **then**
      $S = S'$;
      $C_S = C_{S'}$;
      $Improve = 1$;
    end
  end
  if $Improve = 0$ **then**
    $j = j + 1$;
  end
end
return S;

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). The time-dependent parameter, i.e. β was set 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 let $\beta_j \in (0, 0.1), \forall j \in J$, and for the second setting $\beta_j \in (0, 0.2), j \in J$. 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 the mathematical model and the heuristic in the programming language Python version 2.7. We perform all computational experiments on a PC with Intel® Core™ i5-7500 CPU clocked at 3.40GHz with 8GB of memory under Linux Ubuntu 18.04 operating system. We set the time limit of 3600 seconds for the solver Gurobi. We utilize one processor (thread) for the heuristic, 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 1 reports two criteria of "Feasible", i.e. number of feasible solutions obtained by the method, and "Optimal", i.e. number of optimal solutions delivered by the method. Both 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; 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 more optimal solutions. The proposed heuristic, however, delivers feasible solution for all instances. Interestingly, the heuristic produces the same best solution for 18 of those instances, i.e. for 45%.

Table 1: Number of feasible and optimal solutions for the heuristic and Gurobi.

n	Setting for β	Feasible			Optimal		
		Heuristic	Gurobi^1	Gurobi^4	Heuristic	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 2 reports two criteria of "Gap (%)" and "Time (sec)" (computation time in seconds), which are averaged over 10 instances per size of the problem. 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. Indeed, the gap measures how close the solution obtained by the method is to the best available solution. 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, notice 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 ("-" in Table 2 shows this).

Table 2: Gap from the best obtained solution and computation time for the heuristic and Gurobi.

n	Setting for β	Feasible			Optimal		
		Heuristic	Gurobi[1]	Gurobi[4]	Heuristic	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	< 0.01	3600.00	3600.03
	(0, 0.2)	0.14	4.45	3.81	< 0.01	3600.01	3600.03
50	(0, 0.1)	0.00	75.42	75.42	< 0.01	3600.01	3600.07
	(0, 0.2)	0.00	-	-	< 0.01	3600.02	3600.02
75	(0, 0.1)	0.00	77.16	77.16	< 0.01	3600.15	3600.26
	(0, 0.2)	0.00	-	-	< 0.01	3600.28	3600.39
100	(0, 0.1)	0.00	-	-	< 0.01	-	-
	(0, 0.2)	0.00	-	-	< 0.01	-	-

Table 3 summarizes the outcomes of heuristic and both versions of Gurobi. 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 0.3%, while its worst gap is nearly 1.2%. In addition, it is very efficient since it instantly solves even the problems with 100 jobs. The average time of both Gurobi[1] and Gurobi[4] is almost 40 minutes, and significantly increases with the number of jobs.

Table 3: Overall results for the heuristic and the solver Gurobi.

Method	Feasible	Optimal	Gap (%)		Time (sec)	
			Ave	Max	Ave	Max
Heuristic	**120**	18	**0.30**	**1.22**	$<$ **0.01**	$<$ **0.01**
Gurobi[1]	71	37	13.25	77.16	2521.18	3600.28
Gurobi[4]	71	**40**	12.93	77.16	2332.69	3600.39

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. For general 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. 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 to obtain good quality solutions instantly.

Acknowledgments

Mostafa Khatami is the recipient of UTS International Research Scholarship (IRS) and UTS President's Scholarship (UTSP). Dr Amir Salehipour is the recipient of an Australian Research Council Discovery Early Career Researcher Award (DECRA) funded by the Australian Government.

References

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

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.

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.

Brauner, N., Finke, G., Lehoux-Lebacque, V., Potts, C., and Whitehead, J. (2009). "Scheduling of coupled tasks and one-machine no-wait robotic cells". *Computers & Operations Research* 36(2), 301 –307.

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.

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., Salehipour, A., and Cheng, T. C. E. (2019). "Coupled task scheduling with exact delays: Literature review and models". *Optimization Online*.

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.

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.

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., Giroudeau, R., and König, J.-C. (2011). "Complexity and approximation for scheduling problem for a torpedo". *Computers & Industrial Engineering* 61(2), 352 –356.