A binary search algorithm for the general coupled task scheduling problem

Mostafa Khatami* Amir Salehipour†

October 13, 2020

Abstract

The coupled task scheduling problem aims to schedule a set of jobs, each with at least two tasks and there is an exact delay period between two consecutive tasks, on a set of machines to optimize a performance criterion. We study the problem of scheduling a set of coupled jobs to be processed on a single machine with the objective of minimizing the makespan, which is known to be strongly NP-hard. We obtain competitive lower bounds for the problem through different procedures, including solving 0-1 knapsack problems. We obtain an upper bound by applying a heuristic algorithm. We then propose a binary search heuristic algorithm for the coupled task scheduling problem. We perform extensive computational experiments and show that the proposed method is able to obtain quality solutions. The results also indicate that the proposed solution method outperforms the standard exact solver Gurobi.

Key words: coupled task scheduling; single machine; minimizing makespan; binary search; heuristic; bounds

1 Introduction

We study the problem of minimizing the makespan for scheduling a set $J = \{1, \ldots, n\}$ of "coupled task" jobs on a single machine. A coupled task job consists of two separated tasks where there is an "exact" delay period between them. The second (completion) task of job $j \in J$ must be processed after the completion of its first (initial) task plus an exact duration of the delay. A job j is shown by a triple (a_j, L_j, b_j), where parameters a_j, L_j and b_j take positive integer values and denote the processing time of the initial task, the duration of the delay and the processing time of the completion task (see Figure 1). Using the three-field notation of Graham et al. (1979), minimizing the makespan for the single machine coupled task scheduling problem is commonly denoted as $1|(a_j, L_j, b_j)|C_{max}$ (Khatami et al., 2020).

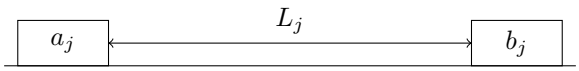

Figure 1: A coupled-task job.

*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@gmail.com

The coupled task scheduling problem has various real-world applications. Shapiro (1980) introduced the coupled task scheduling problem to model a pulsed radar system, where a pulse of electromagnetic energy is utilized to track an object. The transmission of the pulse and the reception of its reflection after a period of time help to measure the size and/or the shape of the tracking object. Therefore, the initial task includes transmission of the pulse and the completion task deals with the reception of its reflection. The system aims at detecting as many objects as possible, which is equivalent to minimizing the maximum completion time of all coupled task jobs. Simonin (2009); Simonin et al. (2011a) and Simonin et al. (2011b) utilized the coupled task scheduling problem to process the environmental data in submarine torpedoes. The sensors on a torpedo transmit pulses to the water and receive their echoes after a period of time. The problem has similar characteristics to that of the pulsed radar system. Ageev and Baburin (2007) modeled a chemistry manufacturing process as a coupled task scheduling problem. Here, it is typical to have an exact technological delay between the consecutive tasks of jobs. Brauner et al. (2009) and Lehoux-Lebacque et al. (2015) utilized the coupled task problem to model a single-machine no-wait scheduling problem in a robotic cell system.

The coupled task scheduling problem has also been utilized to model certain problems in the healthcare domain. In an outpatient chemotherapy clinic, Condotta and Shakhlevich (2014) showed that the problem of scheduling patients' appointments can be formulated as a coupled task scheduling problem. Consider the problem of scheduling patients in a pathology laboratory, where the performance criterion is to minimize the waiting time of the patients (Marinagi et al., 2000; Azadeh et al., 2014). 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. As another application, consider scheduling the appointment of patients in a nuclear medicine clinic. This problem is more complex than its counterpart, e.g., in a typical medical imaging clinic, due to its very strict multi-stage sequential procedures (Pérez et al., 2011; Pérez et al., 2013). Here, 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 procedure justify minimizing a performance criterion related to the patients flow time, which is equal to maximizing the number of treated patients.

Shapiro (1980) showed that problem $1|(a_j, L_j, b_j)|C_{max}$ is NP-hard, due to its equivalence to a class of NP-hard two-machine job shop scheduling problems. By reductions from the 3-Partition problem, Orman and Potts (1997) and Sherali and Smith (2005) proved that minimizing the makespan for the single coupled task scheduling problem, i.e., $1|(a_j, L_j, b_j)|C_{max}$ is strongly NP-hard. The problem remains strongly NP-hard even if the sequence for the initial (or completion) tasks is given (Condotta and Shakhlevich, 2012). Several special cases were also shown to be strongly NP-hard (Orman and Potts, 1997). We refer the interested reader to Khatami et al. (2020) for a comprehensive review of the coupled task scheduling problem, its applications and the mathematical models.

Shapiro (1980) proposed two heuristics of "interleaving" and "nesting" for problem $1|(a_j, L_j, b_j)|C_{max}$. The interleaving heuristic consists of sequencing the jobs so that the completion tasks arrive for processing in the same order as the initial tasks (Figure 2a), while the completion tasks arrive in the reverse order of the initial tasks in the nesting heuristic (Figure 2b). Li and Zhao (2007) and Condotta and Shakhlevich (2012) proposed tabu search algorithms for the problem. To the best of our knowledge, the only exact method for the coupled task scheduling problem is the branch-and-bound algorithm of Békési et al. (2014) that is able to solve instances with up to 20 jobs to optimality in reasonable amounts of

time. The study generated four types of instances: The first type includes instances with up to 20 jobs and types 2, 3 and 4 consist of instances between 7 and 12 jobs.

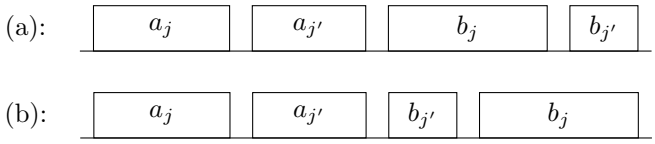

Figure 2: Interleaving jobs j and j' (a) and nesting jobs j and j' (b).

The emerging interest in the coupled task scheduling problem in the recent years, together with the new applications of the problem indicate an increasing demand for effective and efficient solution methods. The previous tabu search algorithms were not compared with other methods. Also, they were tested on instances that are not publicly available, making therefore difficult to conduct a comparison study and evaluate their performance.

We propose an effective and efficient solution method for problem $1|(a_j, L_j, b_j)|C_{max}$ and assess its performance against a standard exact solver. The contributions of our study include (1) improving the best available lower bound for the problem by proposing a 0-1 knapsack formulation, (2) applying a fast local search to calculate an upper bound, (3) developing a binary search heuristic algorithm that utilizes lower and upper bounds to deliver quality solutions, (4) testing the performance of the binary search heuristic under different search initialization strategies, and (5) solving a set of 240 benchmark instances that are publicly available and showing that the proposed binary search heuristic is able to produce better solutions than the standard solver Gurobi.

The remainder of this paper is organized as follows. In Section 2, we formally state the problem and model it as a mixed-integer program. In Section 3, we discuss the proposed solution method for the problem. We also discuss the components of the method including the lower and upper bounds. We detail the results of the computational experiments in Section 4. The paper ends with a few conclusions and future research directions in Section 5.

2 Problem statement and formulation

We shall formally state the problem of this study, i.e., $1|(a_j, L_j, b_j)|C_{max}$ as follows. A set $J = \{1, \ldots, n\}$ of coupled task jobs are to be processed on a single machine and there is an exact delay period between the two tasks of each job. A job $j \in J$ is represented by (a_j, L_j, b_j), where a_j and b_j denote the processing time of the initial and the completion tasks of job j and L_j is the delay period between the tasks of job j. We let $H = \{1, \ldots, 2n\}$ denote the set of tasks, where H_{2j-1} and H_{2j} represent the initial and completion tasks of job j. All processing times and delay durations are integral. Preemption of tasks is not allowed, i.e., once the operation of a task is started it must be completed with no interruption. The tasks of other jobs, however, can be processed during the delay period. The objective is to minimize the length of the schedule, i.e., the makespan denoted as C_{max}.

There are a number of mathematical programs available in the literature for the coupled task scheduling problem. Khatami et al. (2020) reviewed these models and conducted a comprehensive computational experiment and showed that the model proposed by Békési et al. (2014) is among the top performing ones. Hence, we utilize 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 any pair of tasks h, h',

a binary variable $x_{hh'}$ is defined, which takes the value of 1 if task h' starts after task h in the sequence, and 0 otherwise. The problem P1 below presents this formulation.

Problem P1

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

subject to

$$C_{max} \geq s_{2j} + b_j, \quad j \in J, \tag{2}$$

$$x_{2j-1,2j} = 1, \quad j \in J, \tag{3}$$

$$x_{hh'} + x_{h'h} = 1, \quad (h, h') \in H^2, h < h', \tag{4}$$

$$x_{hh'} + x_{h'h''} + x_{h''h} \leq 2, \quad (h, h', h'') \in H^3, h \neq h', h \neq h'', h' \neq h'', \tag{5}$$

$$s_{2j} = s_{2j-1} + a_j + L_j, \quad j \in J, \tag{6}$$

$$s_{2j} \leq UB - b_j, \quad j \in J, \tag{7}$$

$$s_{h'} \geq s_h + p_h - UB(1 - x_{hh'}), \quad (h, h') \in H^2, \quad h \neq h', \tag{8}$$

$$s_h \geq 0, \quad h \in H, \tag{9}$$

$$x_{hh'} \in \{0, 1\}, \quad (h, h') \in H^2, \quad h \neq h', \tag{10}$$

where s_{2j} is the start time of the completion task of job j and UB is an upper bound for C_{max}. Constraints (3) indicate that the completion task of each job should be scheduled after its initial task. Constraints (4) represent the relative order of any pair of tasks. The so-called "3-dicycle inequalities" are presented in constraints (5) for any triple distinct tasks. Constraints (6) define the relation between the start time of the tasks of a job, where constraints (7) set an upper bound on the start time of the completion tasks. Constraints (8) relate the start time variables to the linear ordering variables, where p_h is the processing time of task h. Specifically, the constraints ensure that if task h' is scheduled after task h, its start time must be at least as large as the completion time of task h.

The coupled task scheduling problem of this study, i.e., $1|(a_j, L_j, b_j)|C_{max}$ is strongly NP-hard. Khatami et al. (2020) showed that solving the available mathematical programs for the problem, including problem P1, by standard exact solvers leads to efficient solutions (i.e., in a reasonable amount of time) only for small instances. We therefore propose a binary search heuristic algorithm to solve larger instances. Next, we detail the proposed algorithm.

3 A binary search heuristic

In this section we present a binary search heuristic algorithm for problem $1|(a_j, L_j, b_j)|C_{max}$. The general idea of the proposed heuristic is as follows. First, a lower bound (LB) and an upper bound (UB) on the value of the optimal makespan are calculated. A point in the interval $[LB, UB]$ is selected as the binary bound bb. Then, the heuristic investigates whether it is possible (feasible) to schedule all jobs such that the makespan is bounded from above by the binary bound, i.e., $C_{max} \leq bb$. For this reason, the heuristic utilizes an exact solver to solve a feasibility problem associated with problem P1. A feasible solution indicates that bb is a valid makespan, implying that UB can be tightened to bb, i.e., $UB = bb$. If there is no feasible solution, we update the lower bound to bb, i.e., $LB = bb$. Therefore, at each iteration either LB or UB is tightened. If the length of interval between the lower and upper bounds is 1, i.e., $UB - LB = 1$, the upper bound is equal to the optimal makespan because we deal with integer values for all problem input data. We note that because problem $1|(a_j, L_j, b_j)|C_{max}$ is strongly NP-hard it is less likely that we observe a quick convergence of the proposed binary search heuristic in a reasonably short time. Therefore, we set the stopping criterion of the heuristic as either $UB - LB = 1$ or when a time limit is reached, whichever occurs the first. Under the latter condition the most recent upper bound is a valid makespan for the problem. Algorithm 1 summarizes the proposed binary search heuristic.

Algorithm 1: The binary search heuristic for the coupled task scheduling problem.

```
Input: lb (a lower bound), ub (an upper bound), time_limit.
Output: A schedule.
UB := ub;
LB := lb;
elapsed_time := 0;
while UB − LB > 1 and elpased_time < time_limit do
    Calculate bb, bb ∈ [LB, UB];
    Set C_max ≤ bb;
    if a feasible solution exists then
        UB := bb;
    else
        LB := bb;
    end
end
return UB;
```

We note that the binary search algorithm is also known as dichotomous search in the literature. Dichotomous search has been successfully applied to various optimization problems, including the traveling salesman problem (França et al., 1995), the project scheduling problem (Carlier and Néron, 2003) and the job-shop scheduling problem (Grimes and Hebrard, 2015). Next, we detail the main components of Algorithm 1 including the calculation of lower and upper bounds and solving the feasibility problem.

3.1 Lower bound

In the coupled task scheduling problem, a trivial LB on C_{max} can be obtained by scheduling all tasks without any idle time between them (Li and Zhao, 2007). We denote this lower bound by lb_0, where $lb_0 = \sum_{j \in J} (a_j + b_j)$. However, exclusion of parameter $L_j, j \in J$ from lb_0 results in a loose lower bound.

Therefore, we propose two remedies for this. We note that not always the values of L_j can be included in lb_0, for example, when $a_j = b_j = L_j = p, p \in \mathbb{Z}^+$.

As the first remedy, we can improve lb_0 by checking whether there are some jobs with delays smaller than the minimum processing times, i.e., if $L_j < p_{min}, j \in J$, where $p_{min} = \min_{j \in J}\{a_j, b_j\}$. These jobs are known as "singleton" in the literature (Li and Zhao, 2007). For the singleton jobs the delay period cannot be utilized to schedule any other task. Therefore, L_j associated with the singleton jobs can be included in the lower bound. We let lb_1 denote this. Equation (11) shows the calculation of lb_1.

$$lb_1 = \sum_{j \in J} (a_j + b_j) + \sum_{\substack{j \in J \\ L_j < p_{min}}} L_j. \tag{11}$$

It is clear that $lb_1 \geq lb_0$ for any instance of the problem.

As the second remedy, we may improve lb_0 if we are able to locate those jobs that their delay periods cannot be completely utilized to schedule other tasks. Assume that we aim to concatenate as much tasks as possible in the delay period of job j, which has a length of L_j. This can be modeled as a 0-1 knapsack problem. In other words, the initial and completion tasks of all jobs other than job j are eligible to be inserted into this delay period. We define p_h the processing time of task h. We note that if task h is an initial task for some job j, then we have $p_h = a_j$. Similarly, if task h is a completion task for some job j, then we have $p_h = b_j$. We also define a binary decision variable y_h that takes the value of 1 if task h is selected. Such a concatenation problem can be formulated as problem K.

Problem K

$$z = \max \sum_{\substack{h \in H \\ h \notin \{2j, 2j-1\}}} p_h y_h \tag{12}$$

subject to

$$\sum_{\substack{h \in H \\ h \notin \{2j, 2j-1\}}} p_h y_h \leq L_j, \tag{13}$$

$$y_{2j'-1} + y_{2j'} \leq 1, \quad \forall j' \in J \setminus \{j\}, \quad \text{if} \quad a_{j'} + L_{j'} + b_{j'} > L_j, \tag{14}$$

$$y_{2j'-1} - y_{2j'} \geq 0, \quad \forall j' \in J \setminus \{j\}, \quad \text{if} \quad L_{j'} < a_j, \tag{15}$$

$$y_{2j'} - y_{2j'-1} \geq 0, \quad \forall j' \in J \setminus \{j\}, \quad \text{if} \quad L_{j'} < b_j, \tag{16}$$

where, the objective function (12) maximizes the summation of processing time of the selected tasks, and it is forced to be no larger than the delay period of job j (constraint (13)). Constraints (14) imply that only either of tasks of job j' can be selected if the nesting of job j' inside job j is not possible. Constraints (15) (constraints (16)) ensure that if the delay period of job j' is not as large as the initial (completion) task of job j, the completion (initial) task of job j' would only be selected if its initial (completion) task is selected as well.

Solving problem K for job j results in whether L_j can be completely filled with some tasks. If not, it means that there is an idle time I_j within L_j, which is equal to $L_j - z_K^*$. This implies that in the optimal

schedule of problem P1 there will be an idle time (inside L_j) at least equal to I_j. If we solve problem K for all jobs $j \in J$, i.e., solving n 0-1 knapsacks, and add the maximum idle time among all found idle times, i.e., $\max_{j\in J}\{I_j\}$ to lb_0, a tighter lower bound can be obtained. We let lb_2 denote this lower bound (see Equation (17)). The Knapsack problem is NP-hard in the ordinary sense. Nonetheless, the problem can be efficiently solved even for large inputs by the pseudo-polynomial time dynamic program of Martello et al. (1999). That algorithm is able to solve instances with up to 10,000 items in less than a second.

$$lb_2 = \sum_{j\in J}(a_j + b_j) + \max_{j\in J}\{I_j\}. \tag{17}$$

It is clear that lb_2 is a tighter lower bound than lb_0 since $lb_2 \geq lb_0$ for any instance of the problem. It should be noted that we cannot add the summation of the idle times (instead of their maximum value) to lb_0 because the idle times may be overlapped. Given $lb_1 \geq lb_0$ and $lb_2 \geq lb_0$, Equation (18) follows directly:

$$C_{max} \geq \max\{lb_1, lb_2\}. \tag{18}$$

Next, we propose an upper bound for the problem.

3.2 Upper bound

A trivial upper bound on C_{max} can be calculated as $\sum_{j\in J}(a_j + L_j + b_j)$. That is, to schedule the jobs one by one without any interleaving or nesting. This is called "appending" in the literature. This bound, however, is expected to be of a large value, and might not therefore be tight because interleaving, nesting and pairwise interchange operations are excluded from the calculation of the bound. We may tighten this bound by applying a local search algorithm. The proposed local search algorithm, which is summarized in Algorithm 2, starts with a given sequence π_0 of jobs, and then iteratively improves it by performing adjacent pairwise interchanges. We obtain $\pi_0 = (1, \ldots, n)$, i.e., we add the jobs index to π_0 in increasing order of indices. We implement the first improvement criterion, i.e., once a sequence with an improved (feasible) schedule is obtained we accept it and we update the sequence.

For a sequence π, the algorithm generates a feasible schedule with nesting, interleaving and appending operations as follow. For a pair of consecutive (adjacent) jobs $j, j' \in \pi$, where j comes before j', if nesting of job j' inside job j is possible, i.e., $a_{j'} + L_{j'} + b_{j'} \leq L_j$, then job j' is inserted inside job j. However, if nesting is not possible but interleaving, i.e., $L_j \geq a_{j'} \wedge L_{j'} \geq b_j$, then the interleaving is performed where job j is the first job in the pair. If neither nesting nor interleaving of a pair of jobs is possible, job j' is adjacently appended after job j. The algorithm performs this procedure for all consecutive (adjacent) pairs of jobs until all jobs are scheduled. We note that the worst-case time complexity of the local search algorithm is $O(n^2)$.

Algorithm 2: The local search algorithm.

Input: $\pi_0 = (1, \ldots, n)$, C_{π_0}, $j = 1$.
Output: A sequence π with makespan C_π.

```
π := π_0;
C_π := C_{π_0};
while j ≤ n − 1 do
    improve := false;
    k := j + 1;
    for k ≤ n do
        π' ← swap(j, k);
        S_{π'} ← Generate a feasible schedule for π';
        C_{π'} ← makespan(S_{π'});
        if C_{π'} < C_π then
            π := π';
            C_π := C_{π'};
            improve := true;
        end
    end
    if improve is false then
        j := j + 1;
    end
end
return C_π;
```

3.3 The feasibility problem

Given lower and upper bounds on the value of the makespan, the next step is to systematically tighten the gap between the bounds until the stopping criterion is met, i.e., either no further tightening is possible or the computation time limit is reached.

We tighten the gap between the lower and upper bounds by iteratively solving a feasibility problem associated with problem P1. We generate such a feasibility problem by changing the objective function of problem P1 into a constant and bounding C_{max} from above by bb. We let P2 represent the new problem. Problem P2 is shown in the following.

Problem P2

$$z = \min \zeta \tag{19}$$

subject to

(2), (4) to (10),

$$C_{max} \leq bb, \tag{20}$$

where ζ is a constant. We do not include constraints (3) in problem P2 because these constraints do not impact the feasibility of the problem. Constraint (20) sets the binary bound as an upper bound on the

value of the makespan.

Solving problem P2 is essentially equivalent to identifying a feasible solution for problem P1. Also, if problem P2 does not have a feasible solution, neither does problem P1. We may use standard optimization solvers for solving problem P2. Even though problem P2 might be easier to solve than problem P1 because at least no "optimization" phase is performed, it might be still challenging to find a feasible solution for problem P2, implying that the process may be computationally expensive, particularly for large problem instances, i.e., for problems with large number of jobs. We therefore propose two speed-up techniques to decrease, to some extent, the computation burden of solving problem P2.

Speed-up 1. The first speed-up that we propose benefits from the "call-back" functionality of the solver. A call-back provides additional information during the solve process of the solver. We can use such additional information to stop the solver as soon as a feasible solution is found.

Speed-up 2. The second speed-up focuses on modifying the "solve focus" of the solver. Various solvers have different name for this feature. For example, the solver Gurobi names the feature "MIPfocus". Changing the focus of the solver towards obtaining a feasible solution, rather than, e.g., an optimal solution (which focuses on proving the optimality of the current solution), is an effective computation burden reduction strategy.

Next, we evaluate the performance of the proposed binary search heuristic algorithm, i.e., Algorithm 1.

4 Computational experiment

We perform an extensive computational experiment to evaluate the performance of the proposed binary search heuristic. We test the algorithm on the general benchmark instances of the coupled task scheduling problem proposed in Khatami et al. (2020), denoted as "general set". The benchmark consists of 240 instances in eight different sizes, where $n \in \{5, 10, 15, 20, 25, 40, 50, 100\}$. Per each value of n, 30 instances were randomly generated and that in three categories of small (S), medium (M) and large (L). Each category includes 10 instances. The processing time of tasks and the duration of delays are from the discrete uniform distribution with parameters $a_j, b_j \sim U(1, 20), L_j \sim U(10, 80)$ for the category S, $a_j, b_j \sim U(1, 50), L_j \sim U(25, 200)$ for the category M and $a_j, b_j \sim U(1, 100), L_j \sim U(50, 400)$ for the category L.

We utilize Gurobi version 8.0.0 (Gurobi Optimization, 2018), as the solver within the binary search heuristic algorithm, to solve problem P2. We use the same solver Gurobi as the stand-alone solver to solve the same instances by optimizing problem P1. Also, we utilize Gurobi to solve problem K (for delivering LB). We implement problems P1, P2 and K, as well as all algorithms in the programming language Python version 3.6. Unless otherwise stated we perform all computational experiments on a standard PC with Intel® Core™ i5-7500 CPU clocked at 3.40GHz with 8GB of memory under Linux Ubuntu 18.04 operating system, and we set the time limit to 3600 seconds for both the binary search heuristic and the stand-alone Gurobi and we utilize four processors. We use the default value for the remaining parameters of Gurobi.

Recall that the binary bound is in the ranges $[LB, UB]$, i.e., $bb \in [LB, UB]$. We investigate two strategies to choose bb: (1) $bb = \frac{lb+ub}{2}$, i.e., it is equal to the midpoint of the interval, and (2) $bb = \frac{lb+3ub}{4}$, i.e., it is equal to the three fourth of the interval. We solve the 240 instances with both strategies. We let $bb_{1/2}$ and $bb_{3/4}$ denote these two strategies.

We use four criteria of "feas", "best", "opt" and "gap" (in %) to compare the proposed binary search heuristic and Gurobi. The criteria feas, best and opt denote the number of feasible, best and optimal

solutions obtained by each method. The criterion gap reports the average gap of 10 instances. The gap of an instance is calculated as $\frac{z-z^*}{z^*} \times 100$, where z is the objective function value obtained by the tested method/strategy and z^* is the best objective function value among the tested methods/strategies. We note that the criterion gap is calculated over the instances for which a feasible solution is delivered. We report the outcomes of the computational experiments in Tables 1 to 4, where a table represents one evaluation criterion.

Table 1 summarizes the performance of the methods for the criterion feas. The table shows that the binary search heuristic under both $bb_{1/2}$ and $bb_{3/4}$ strategies obtain feasible solution for all 240 instances, whereas Gurobi is unable to find feasible solution for 43 instance, for which $n = 40, 50, 100$. In particular, Gurobi cannot deliver feasible solution for any of the instances with 100 jobs. This shows that Gurobi is not able to even generate acceptable solution for large instances. It is also clear that Gurobi's performance deteriorates for medium sized instances.

Table 1: The number of feasible solutions (criterion feas) obtained by the binary search and Gurobi.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	10	10	10
	M	10	10	10
	L	10	10	10
10	S	10	10	10
	M	10	10	10
	L	10	10	10
15	S	10	10	10
	M	10	10	10
	L	10	10	10
20	S	10	10	10
	M	10	10	10
	L	10	10	10
25	S	10	10	10
	M	10	10	10
	L	10	10	10
40	S	10	10	9
	M	10	10	10
	L	10	10	10
50	S	10	10	4
	M	10	10	6
	L	10	10	8
100	S	10	10	0
	M	10	10	0
	L	10	10	0
Total		240	240	197

Table 2 shows that Gurobi is slightly superior to the binary search in obtaining the optimal solution for the instances with a small number of jobs. We note that the binary search heuristic under both strategies still obtains the optimal solution for all instances with 5 jobs but one. Both the binary search heuristic and Gurobi find the optimal solution for all instances with 10 jobs. For the instances with 15 jobs, Gurobi finds the optimal solution for only two instances. Overall, Gurobi delivers optimal solution for 62 instances, i.e., only for three more instances than the binary search heuristic.

Table 2: The number of optimal solutions (criterion opt) obtained by the binary search and Gurobi.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	10	10	10
	M	10	10	10
	L	9	9	10
10	S	10	10	10
	M	10	10	10
	L	10	10	10
15	S	0	0	1
	M	0	0	0
	L	0	0	1
20	S	0	0	0
	M	0	0	0
	L	0	0	0
25	S	0	0	0
	M	0	0	0
	L	0	0	0
40	S	0	0	0
	M	0	0	0
	L	0	0	0
50	S	0	0	0
	M	0	0	0
	L	0	0	0
100	S	0	0	0
	M	0	0	0
	L	0	0	0
Total		59	59	62

For the criterion best, i.e., the best obtained solutions, which is reported in Table 3, in general, the strategy $bb_{3/4}$ outperforms the strategy $bb_{1/2}$ and also Gurobi, in particular, for large instances. We note that Gurobi is not comparable to the binary search heuristic when the number of jobs increases, more precisely, when $n = 25, 40, 50, 100$. Similar outcomes is observed for the criterion gap, which is reported in Table 4. Indeed, Gurobi performs well only for instances with small number of jobs. The Gurobi's solutions are of poor quality when $n \geq 25$. It is clear that the strategy $bb_{3/4}$ performs better than $bb_{1/2}$ and also than Gurobi, because it has the smallest average gap. In Table 4, the "-" indicates that the criterion gap cannot be calculated because not a single feasible solution was reported by Gurobi.

Table 3: The number of best solutions (criterion best) obtained by the binary search and Gurobi.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	10	10	10
	M	10	10	10
	L	9	9	10
10	S	10	10	10
	M	10	10	10
	L	10	10	10
15	S	0	1	9
	M	0	0	10
	L	1	0	9
20	S	0	4	8
	M	1	2	8
	L	1	4	5
25	S	6	3	1
	M	1	7	2
	L	3	6	1
40	S	7	3	0
	M	7	2	1
	L	6	3	1
50	S	1	8	1
	M	3	5	2
	L	2	4	4
100	S	7	10	0
	M	9	10	0
	L	10	10	0
Total		124	141	122

Table 4: The gap (in %) from the best solution.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	0.0	0.0	0.0
	M	0.0	0.0	0.0
	L	0.0	0.0	0.0
10	S	0.0	0.0	0.0
	M	0.0	0.0	0.0
	L	0.0	0.0	0.0
15	S	1.8	1.6	0.0
	M	2.6	1.5	0.0
	L	2.2	1.7	0.0
20	S	3.0	1.0	0.2
	M	3.3	1.1	0.2
	L	3.1	0.6	0.5
25	S	0.8	0.5	2.1
	M	1.8	0.4	1.8
	L	1.5	0.8	2.1
40	S	1.2	3.1	37.4
	M	2.5	5.9	28.3
	L	3.9	3.5	30.8
50	S	5.6	0.8	7.8
	M	5.5	1.7	43.4
	L	4.4	5.8	23.2
100	S	3.5	0.0	-
	M	1.3	0.0	-
	L	0.0	0.0	-
Average		2.0	1.25	7.48

We summarize the outcomes of both tested strategies of the binary search heuristic and Gurobi in Table 5, where the highlighted values denote the superior ones. The table shows that all methods perform close to one another for small instances in which $n = 5, 10, 15, 20$. Once the number of jobs increases, Gurobi cannot even deliver feasible solution for all problem instances. When Gurobi does find a feasible

solution, however, it is usually of poor quality. The binary search heuristic reports superior solution for large instances. The results also indicate that the binary search heuristic under the strategy $bb_{3/4}$ outperforms the one under the strategy $bb_{1/2}$.

Table 5: An overview of the outcomes of the binary search heuristic and Gurobi.

Criterion	$bb_{1/2}$	$bb_{3/4}$	Gurobi
Feas	**240**	**240**	197
Opt	59	59	**62**
Best	124	**141**	122
Gap (in %)	2.00	**1.25**	7.48

According to Table 2, The stand-alone Gurobi is able to optimally solve only two instances with $n = 15$ within the time limit of 3600 seconds. Also, we observe that the stand-alone Gurobi may not benefit from an additional computation time, even to the extent of a few days. Therefore, we use the solver Gurobi on the NEOS server to solve the instances that Gurobi cannot solve to optimality. This is important to further demonstrate the quality of the solutions produced by our binary search heuristic. The NEOS server (Czyzyk et al., 1998) is a free cloud service for solving optimization problems that includes various solvers including Gurobi. The solvers are run on distributed high-performance machines, resulting in solving problems that cannot be generally solved on standard machines, even though, each solver on the NEOS server is limited to 3 GB of memory and 8 hours of run time and that the solver is restricted to a maximum of 4 processors per an optimization problem submitted to the NEOS server. The NEOS server is hosted by the Wisconsin Institute for Discovery at University of Wisconsin–Madison.

In total, with the help of the NEOS server we obtain the optimal solution for an additional 12 instances with $n = 15$. We report the outcomes in Table 6. In the table, we also report the detailed outcomes for each strategy of the binary search heuristic and the stand-alone Gurobi, including the objective function value (denoted by z), the computation time (in seconds) and the gap (in %) from the best found solution. We highlight the optimal solutions. For the remaining 16 instances with $n = 15$ that we still do not have the optimal solution, we attempt an exhaustive search to optimally solve them. For this reason, we use the best solution reported by the NEOS server in order to warm-start the stand-alone Gurobi solver on our standard PC, however, with an increased time limit of 24 hours. With this procedure, we obtain the optimal solution for one additional instance, more precisely, for instance 15-1-S. In Table 6, we use the "*" to denote the optimal objective function value for that instance.

In summary, we now have the optimal solution for 15 instances with $n = 15$, i.e., for 50% of the instances. The results show that the average gap for the instances under the strategy $bb_{3/4}$ is 2.4, while it is equal to 3.1 for the strategy $bb_{1/2}$. Those results further indicate that the binary search heuristic is able to produce quality solutions, needless to say that most of the optimal solutions even for the instances with $n = 15$ are obtained after 8 (or more) hours of computational time on the high-performing machines of the NEOS server. Clearly, those large computational times may not justify the marginal improvement in the objective function value, particularly, for practical applications, e.g., in the healthcare appointment scheduling (see Section 1). More importantly, these applications frequently demand for (quality) schedules, typically on a daily or weekly basis.

Our efforts in solving the instances with $n = 20$ by using the Gurobi solver of the NEOS server failed as the solver could not solve any of those instances to optimality. That further indicates the challenges of obtaining the optimal solution even for small instances. Therefore, we may argue that none of larger instances with $n > 20$ can be solved to optimality, neither on the NEOS server nor on our machine, within a reasonable amount of time.

Table 6: The detailed results for instances with $n = 15$.

Instance	$bb_{1/2}$			$bb_{3/4}$			Gurobi			NEOS		
	z	Time	Gap (in %)	z	Time	Gap (in %)	z	Time	Gap (in %)	z	Time	Gap (in %)
15_1_S	280	3600.4	2.6	280	3600.3	2.6	275	3600.0	0.7	**273**	40030.4*	0.0
15_2_S	386	3600.5	2.4	384	3600.4	1.9	379	3600.0	0.5	377	28800.0	0.0
15_3_S	316	3600.4	2.6	314	3600.4	1.9	308	3600.0	0.0	310	28800.0	0.6
15_4_S	261	3600.3	1.6	264	3600.3	2.7	258	3600.0	0.4	**257**	10861.9	0.0
15_5_S	324	3600.3	0.6	332	3600.3	3.1	322	3600.0	0.0	**322**	2653.8	0.0
15_6_S	322	3600.4	0.3	321	3600.4	0.0	322	3600.1	0.3	322	28800.0	0.3
15_7_S	333	3600.3	3.7	331	3600.4	3.1	327	3600.0	1.9	**321**	26122.7	0.0
15_8_S	323	3600.3	4.9	317	3600.3	2.9	313	3600.0	1.6	**308**	23576.8	0.0
15_9_S	278	3600.3	3.0	273	3600.4	1.1	**270**	1266.9	0.0	**270**	893.5	0.0
15_10_S	278	3600.3	1.8	278	3600.3	1.8	273	3600.0	0.0	**273**	18268.9	0.0
15_1_M	856	3600.4	3.0	846	3600.3	1.8	832	3600.0	0.1	**831**	26934.9	0.0
15_2_M	985	3600.3	2.3	980	3600.3	1.8	969	3600.0	0.6	963	28800.0	0.0
15_3_M	754	3600.3	2.0	750	3600.3	1.5	740	3600.0	0.1	**739**	26781.4	0.0
15_4_M	718	3600.4	3.5	714	3600.3	2.9	694	3600.0	0.0	694	28800.0	0.0
15_5_M	662	3600.3	2.5	658	3600.3	1.9	654	3600.0	1.2	**646**	28349.9	0.0
15_6_M	886	3600.4	8.4	848	3600.4	3.8	831	3600.0	1.7	817	28800.0	0.0
15_7_M	744	3600.4	3.5	739	3600.3	2.8	734	3600.0	2.1	**719**	13668.6	0.0
15_8_M	840	3600.3	2.1	835	3600.4	1.5	827	3600.0	0.5	823	28800.0	0.0
15_9_M	910	3600.4	3.2	921	3600.4	4.4	902	3600.0	2.3	882	28800.0	0.0
15_10_M	754	3600.4	5.3	734	3600.4	2.5	725	3600.0	1.3	716	28800.0	0.0
15_1_L	1516	3600.3	6.7	1462	3600.3	2.9	1421	3600.0	0.0	**1421**	26741.1	0.0
15_2_L	1763	3600.4	1.4	1780	3600.4	2.4	1753	3600.0	0.8	1739	28800.0	0.0
15_3_L	1877	3600.3	4.9	1850	3600.4	3.4	1826	3600.0	2.0	**1790**	26382.8	0.0
15_4_L	1304	3600.3	2.1	1298	3600.3	1.6	1294	3600.0	1.3	**1277**	28327.4	0.0
15_5_L	1582	3600.4	5.0	1524	3600.3	1.1	1518	3600.0	0.7	1507	28800.0	0.0
15_6_L	2004	3600.4	5.4	1971	3600.4	3.6	1939	3600.0	1.9	1902	28800.0	0.0
15_7_L	1308	3600.3	2.5	1300	3600.3	1.9	1292	3600.0	1.3	1276	28800.0	0.0
15_8_L	1370	3600.3	1.5	1412	3600.3	4.6	1373	3600.0	1.7	1350	28800.0	0.0
15_9_L	1802	3600.6	1.2	1821	3600.4	2.2	1794	3600.0	0.7	1781	28800.0	0.0
15_10_L	1426	3600.3	2.1	1442	3600.3	3.3	**1396**	1871.6	0.0	**1396**	1416.2	0.0
Average			3.1			2.4			0.9			0.0

* The reported time for the instance 15-1-S is the summation of the time elapsed on the NEOS server, i.e., 8 hours, and the run time on our standard PC.

5 Concluding remarks

There are a few solution methods available in the literature for the strongly NP-hard single machine coupled task scheduling problem with the objective function of minimizing the makespan. Due to the increasing interest in the coupled task scheduling problem in recent years, as well as the emerging applications of the problem, there is a demand for efficient solution methods for the problem. In order to fulfill this aim, we proposed a binary search heuristic algorithm, which systematically tightens a lower and an upper bound. We improved the best available lower bound by solving multiple 0-1 knapsacks. We applied a fast local search to calculate an upper bound. Our computational results indicate that the proposed binary search heuristic is an effective solution method for the single machine coupled task scheduling problem under the optimization criterion of minimizing the makespan.

The proposed binary search heuristic algorithm relies on quality bounds, as well as on an efficient solution to the feasibility problem. Future research directions can therefore focus on alternative feasibility problems that can be solved in short time because the overall procedure requires solving a large number of such feasibility problems, particularly, for large instances.

Acknowledgments

Mostafa Khatami is the recipient of UTS International Research Scholarship (IRS) and 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.

Conflict of interest

The authors declare that they have no conflict of interest.

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.

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.

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.

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.

Carlier, J. and Néron, E. (2003). "On linear lower bounds for the resource constrained project scheduling problem". *European Journal of Operational Research* 149(2), 314–324.

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.

Czyzyk, J., Mesnier, M. P., and Moré, J. J. (1998). "The NEOS Server". *IEEE Journal on Computational Science and Engineering* 5(3), 68 –75.

França, P. M., Gendreau, M., Laporte, G., and Müller, F. M. (1995). "The m-traveling salesman problem with minmax objective". *Transportation Science* 29(3), 267–275.

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.

Grimes, D. and Hebrard, E. (2015). "Solving variants of the job shop scheduling problem through conflict-directed search". *INFORMS Journal on Computing* 27(2), 268–284.

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

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

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.

Li, H. and Zhao, H. (2007). "Scheduling coupled-tasks on a single machine". *IEEE Symposium on Computational Intelligence in Scheduling*, 137–142.

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.

Martello, S., Pisinger, D., and Toth, P. (1999). "Dynamic programming and strong bounds for the 0-1 knapsack problem". *Management Science* 45(3), 414–424.

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

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

Simonin, G. (2009). "L'impact de l'introduction du graphe de compatibilité dans les problèmes d'ordonnancement en présence de tâches-couplées". PhD thesis. Montpellier, France: Universite de Montpellier II.