On the complexity of coupled-task scheduling

A.J. Orman, C.N. Potts

Faculty of Mathematical Studies, University of Southampton, UK

Received 28 February 1994; revised 10 May 1995

Abstract

A coupled-task is a job consisting of two distinct operations. These operations require processing in a predetermined order and at a specified interval apart. This paper considers the problem of sequencing \( n \) coupled-task jobs on a single machine with the objective of minimizing the makespan. By making assumptions about processing times, we obtain many special cases and explore the complexity of each case. NP-hardness proofs, or polynomial algorithms, are given to all except one of these special cases. The practical scenario from which this problem originated is also discussed.

1. Introduction

A coupled-task is a job consisting of two distinct operations. These operations require processing in a predetermined order and at a specified interval apart. Each coupled-task job \( i \) can be denoted by the triple \((a_i, L_i, b_i)\), which represents the processing time of the first task, the time interval between the tasks and the processing time of the second task, respectively. The structure of a coupled-task is displayed in Fig. 1. Note that the second task starts exactly \( L_i \) units after the first task completes. We refer to \( L_i \) as the separation time for job \( i \).

The jobs may be scheduled in any way with the restriction that no two tasks occupy the machine at any one time. Preemption is not allowed.

Two jobs, \( i \) and \( j \), are said to be interleaved if their tasks are scheduled as shown in Fig. 2.

Our motivation for studying this class of problems stems from work on a pulsed radar system. In such a system, the jobs take the coupled-task form. To track an object, or to survey a volume of space, a pulse of electromagnetic energy, of a predetermined

![Fig. 1. The structure of a coupled-task job.](image-url)
length, must be transmitted, taking a time $a$, and then received, for a time $b$. The interval between the transmission and reception of the pulse, $L$, is dependent upon the distance of the target, or the volume of space, from the radar. The radar has the capacity to process only one task at a time. In such a system, the objective is usually to minimize the time that the radar is idle.

The notation used for addressing this type of scheduling problem will be the standard three-field notation, $a|b|\gamma$, see Graham et al. [3]. These terms refer to the machine environment, the job characteristics, and the optimality criterion of the problem, respectively. It is assumed that all problems have $n$ jobs and that $\{i\} (i = 1, \ldots, n)$ is their indexing set. The term $\text{Coup-Task}$ in the second field signifies all the jobs take the coupled-task format.

For an example of the $x|\beta|\gamma$ notation, we have that $1|\text{Coup-Task}, a_i = a, L_i = a + b_i|C_{\max}$ is the problem of minimizing the maximum completion time of the jobs with each job $i$ having a common first task processing time $a$, an arbitrary second task processing time $b_i$, and a separation time $L_i = a + b_i$.

Very little research has been conducted on this type of scheduling problem. The coupled-task problem $1|\text{Coup-Task}|\gamma$ is equivalent to $J2|\text{no-wait}, M_2, \text{non-bott}|\gamma$ with all jobs requiring three operation (see [6]). This problem is discussed at a very elementary level by Shapiro [8], who gives three simple heuristics for the $1|\text{Coup-Task}|C_{\max}$ problem and discussed numerous practical situations where the problem arises. Dell’Amico [1] considers two-machine shop problems with time lags, $t_l_i$. The time lag problems are relaxed versions of the corresponding coupled-task problems as they involve only a lower bound on the interval between the tasks. Rinnooy Kan [6] shows that the permutation version of $F2|t_l_i|C_{\max}$ problem is solvable in $O(n \log n)$. However, the general $F2|t_l_i|C_{\max}$ is unary NP-hard [5], whereas $F2||C_{\max}$ is polynomially solvable by Johnson’s algorithm [4].

In this paper, we study the complexity of minimizing the maximum completion time, or makespan, for various special cases of the coupled-task scheduling problem. Following this introduction, Section 2 looks at the complexity issues by exploring the complexity of many special cases of the coupled-task problem with a $C_{\max}$ objective, and provides a proof that the coupled-task scheduling problem is reversible, that is, a problem defined by $(a_i, L_i, b_i)$ is equivalent to one defined by $(b_i, L_i, a_i)$. Section 3
offers the unary NP-hardness results. Polynomial time algorithms for some coupled-task problems are given in Section 4, with some concluding remarks presented in Section 5.

2. Complexity issues

Before trying to solve any scheduling or combinatorial optimization problem, it is often important to know whether optimality can be achieved in a reasonable amount of computation time, considering the size of the problem instance. The hope is to obtain an optimizing algorithm bounded by a polynomial in the size of the instance; however, this is not possible with most combinatorial optimization problems. The theory of computational complexity is well established and it helps the analysis of scheduling problems to see if there is a reasonable chance of finding such a polynomial algorithm.

For the $C_{\text{max}}$ objective function, we study the various special cases that arise by having a common first or second task processing time, or a common separation time. Another class of special cases results when, for each job, two of its defining processing or separation times are equal. To avoid technical complications for special cases in which all jobs are identical, we assume that processing times and separation times are input for all $n$ jobs, thus giving an input size of $O(n)$ rather than $O(\log n)$. For any special case, there is a corresponding recognition problem: does there exist a schedule with $C_{\text{max}} \leq y$, for some given threshold value $y$? The recognition problem clearly lies in the class NP (we refer to Garey and Johnson [2] for relevant definitions). For each special case, we aim to derive a polynomial time algorithm, or to prove that the corresponding recognition problem is unary NP-complete (NP-complete in the strong sense). In the latter case, the problem is unary NP-hard, which indicates that the existence of a polynomial or pseudopolynomial time algorithm is unlikely.

It is useful to show the equivalence of some of these special cases. More precisely a makespan problem defined by $(a_i, L_i, b_i)$ $(i = 1, \ldots, n)$ is equivalent to one defined by $(b_i, L_i, a_i)$. We refer to the latter problem as the reverse of the first.

**Theorem 1.** A makespan problem and its reverse are equivalent.

**Proof.** Consider any feasible schedule $S$ with makespan $C_{\text{max}}(S)$ in which job $i$ completes at time $C_i$ for $i = 1, \ldots, n$. For the reverse problem, the schedule in which job $i$ starts at time $C_{\text{max}}(S) - C_i$ for $i = 1, \ldots, n$ is also feasible and has makespan $C_{\text{max}}(S)$. Similarly, any schedule for the reverse problem converts into a schedule for the original problem with the same makespan. Thus, the two problems are equivalent. □

The results of our analysis are displayed in complexity graphs $A, B$ and $C$ (see Figs. 3–5). The key for the graphs is as follows.

- $a_i, L_i, b_i$ indicates the general, unrestricted coupled-task scheduling problem.
- $a_i = L_i, b_i$ and similar forms indicate the problems where $b_i$ is unrestricted and where $a_i = L_i$ $(i = 1, \ldots, n)$. 

\( a_i = a \) \( L_i \) \( b_i \) and similar forms indicate the problems where \( L_i \) and \( b_i \) are unrestricted and where \( a_i = a \) (\( i = 1, \ldots, n \)).

\( a_i = L_i = p \) \( b_i \) and similar forms indicate the problems where \( b_i \) is unrestricted and where \( a_i = L_i = p \) (\( i = 1, \ldots, n \)) for some constant \( p \).

Also, the arcs in the graphs are directed from a special case to a more general problem and edges link identical problems. Thus, \textbf{NP}-hardness of a special case implies \textbf{NP}-hardness for all problems that appear on a directed path from the special case.

In view of Theorem 1, for verification that the above stated problems are unary \textbf{NP}-hard, only the proof of three of them need be given, namely

\[ 1|\text{Coup-Task}, a_i - L_i - b_i|C_{\text{max}}. \]
The proofs for these problems are given below in Theorems 2, 3 and 4. Only one problem with the $C_{\text{max}}$ objective function remains open, namely

$$1 | C_{\text{Task}}, a_i = a, L_i = L, b_i = b | C_{\text{max}}.$$
Theorem 2. The $1\text{|Coup-Task, } a_i = L_i = b_i \text{|C_{max}}$ problem is unary NP-hard.

Proof. The proof shows that the unary NP-complete 3-PARTITION problem is reducible to the decision version of the $1\text{|Coup-Task, } a_i = L_i = b_i \text{|C_{max}}$ problem.

Given an instance of 3-PARTITION, we construct the following instance of the decision version of $1\text{|Coup-Task, } a_i = L_i = b_i \text{|C_{max}}$.

There are $n = 4q$ jobs with

$$a_i = L_i = b_i = 2e_i qE^2 + i$$

for $i = 1, \ldots, 3q$,

$$a_i = L_i = b_i = 6qE^3 + 27q + (i - 3q)$$

for $i = 3q + 1, \ldots, 4q$.

Does there exist a schedule with $C_{max} \leq y$, where $y = 3(6q^2E^3 + 27q^2 + q(q + 1)/2)$?

We refer to jobs $1, \ldots, 3q$ as partition jobs and to $3q + 1, \ldots, 4q$ as dividing jobs.

Case 1: We want to show that if 3-PARTITION has a solution, then $1\text{|Coup-Task, } a_i = L_i = b_i \text{|C_{max}}$ has a solution with $C_{max} \leq y$.

If there is a solution to 3-PARTITION, then there exists $Q_j$ for $j = 1, \ldots, q$ such that $\sum_{i \in Q_j} e_i = E$.

Schedule the partition jobs corresponding to $Q_j$ in the separation time of dividing job $j$, as shown in Fig. 6, where $Q_j = \{g, h, i\}$. The dividing jobs are scheduled without interleaving and without idle time between successive jobs.

Therefore,

$$C_{max} = \sum_{j=3q+1}^{4q} (a_j + l_j + b_j) = 3(6q^2E^3 + 27q^2 + q(q + 1)/2) = y.$$

Case 2: We want to show that if there is a schedule for $1\text{|Coup-Task, } a_i = L_i = b_i \text{|C_{max}}$ with $C_{max} \leq y$, then 3-PARTITION has solution.

First note that since each dividing job has a different size from any other job no interleaving of dividing jobs is possible. Moreover, since $\sum_{j=3q+1}^{4q} (a_j + l_j + b_j) = y$, all partition jobs must be scheduled within the separation times of the dividing jobs.

Let $Q_j$ denote the set of partition jobs scheduled within the separation time $L_{3q+j}$ for $j = 1, \ldots, q$. Note that the partition jobs cannot be interleaved with each other as they have different sizes.
If $\sum_{i \in Q_j} e_i > E$, then

$$\sum_{i < q_j} (a_i + L_i + b_i) \geq 6qE^2(E + 1) > 6qE^3 + 27q + j,$$

since $E \geq 3$, and consequently the jobs of $Q_j$ cannot be scheduled within the time separation $L_{3q+j}$.

Therefore, $\sum_{i \in Q_j} e_i \leq E$ for $j = 1, \ldots, q$. Moreover, $\sum_{j=1}^{q} \sum_{i \in Q_j} e_i = qE$ since all the partition jobs are scheduled within the time separations of the dividing jobs. It follows that $\sum_{i \in Q_j} e_i = E$ for $j = 1, \ldots, q$, and consequently $Q_1, \ldots, Q_q$ defines a 3-PARTITION. $\square$

Theorem 3. The 1|Coup-Task, $L_i = L, b_i = b|C_{\max}$ problem is unary NP-hard.

Proof. The proof shows that the unary NP-complete 3-PARTITION problem is reducible to the decision version of the 1|Coup-Task, $L_i = L, b_i = b|C_{\max}$ problem.

Given an instance of 3-PARTITION, we construct the following instance of the decision version of 1|Coup-Task, $L_i = L, b_i = b|C_{\max}$.

There are $n = 4q$ jobs with

$$(a_i, L_i, b_i) = (e_i, E, b) \quad \text{for } i = 1, \ldots, 3q,$$

$$(a_i, L_i, b_i) = (E + 1, E, b) \quad \text{for } i = 3q + 1, \ldots, 4q,$$

where $b = \min_{j=1, \ldots, 3q} \{e_j\}$.

Does there exist a schedule with $C_{\max} \leq y$, where $y = q(3E + 1 + b)$?

We refer to jobs $1, \ldots, 3q$ as partition jobs and to $3q + 1, \ldots, 4q$ as dividing jobs.

Case 1: We want to show that if 3-PARTITION has a solution, then 1|Coup-Task, $L_i = L, b_i = b|C_{\max}$ has a solution with $C_{\max} \leq y$.

If there is a solution to 3-PARTITION, then there exists $Q_j$ for $j = 1, \ldots, q$ such that $\sum_{i \in Q_j} e_i = E$.

Schedule the jobs by constructing $q$ blocks. For block $j$, the first tasks of the partition jobs corresponding to $Q_j$ are scheduled in the separation time of a dividing job as shown in Fig. 7, where $Q_j = \{g, h, i\}$ The complete schedule is obtained by allowing no idle time between blocks.

Note that the idle time for each subschedule is $E - 3b$. The $q$ blocks each contribute $3E + 1 + b$ to the objective value, thus giving $C_{\max} = q(3E + 1 + b) = y$. 

Fig. 7. One of the $q$ blocks of jobs.
Case 2: We want to show that if 3-PARTITION has no solution, then any schedule for $1|\text{Coup-Task}, L_i = L, b_i = b|C_{\text{max}}$ has $C_{\text{max}} > y$.

Firstly, we note that no two dividing jobs can be interleaved. Also, a partition job cannot be interleaved with a dividing job that starts after it. Assume without loss of generality that any partition jobs that are not interleaved with dividing jobs appear at the start of the schedule, and that the dividing job are sequenced in the order $1, \ldots, q$.

Thus, the first task of each dividing job partitions the schedule into $q + 1$ blocks, where block 0 consists of partition jobs $J_0$ scheduled before any dividing job. Let $J_j$ be the set of partition jobs in block $j$ that are interleaved with dividing job $j$ for $j = 1, \ldots, q$. Clearly, $C_{\text{max}} = \sum_{j=0}^{q} T_j$, where $T_j$ is the total time to process all jobs of block $j$.

From feasibility, we have that $\sum_{i \in J_j} e_i \leq E$ for $j = 1, \ldots, q$. The non-existence of a 3-PARTITION implies that one of these inequalities is strict, and therefore $J_0 \neq \emptyset$. Thus,

$$ T_0 \geq \sum_{i \in J_0} e_i + E + b \geq \sum_{i \in J_0} e_i + b. $$

(1)

Now consider block $j$ for $j = 1, \ldots, q$. If $J_j = \emptyset$, then

$$ T_j = E + 1 + \sum_{i \in J_j} e_i + E + b. $$

(2)

On the other hand, if $J_j \neq \emptyset$, then

$$ T_j \geq E + 1 + \sum_{i \in J_j} e_i + E + b. $$

(3)

From (1)–(3), we obtain

$$ C_{\text{max}} = \sum_{j=0}^{q} T_j \geq q(2E + 1 + b) + \sum_{j=0}^{q} \sum_{i \in J_j} e_i + h > y, $$

as required. $\square$

Corollary 1. The $1|\text{Coup-Task}, a_i = a, L_i = L|C_{\text{max}}$ problem is unary NP-hard.

Proof. Using problem reversibility, Theorem 1 shows that the $1|\text{Coup-Task}, a_i = a, L_i = L|C_{\text{max}}$ problem is also unary NP-hard. $\square$

Theorem 4. The $1|\text{Coup-Task}, a_i = b_i = p|C_{\text{max}}$ problem is unary NP-hard.

Proof. The proof shows that the unary NP-complete 3-PARTITION problem is reducible to the decision version of the $1|\text{Coup-Task}, a_i = b_i = p|C_{\text{max}}$ problem.

Given an instance of 3-PARTITION, we construct the following instance of the decision version of $1|\text{Coup-Task}, a_i = b_i = p|C_{\text{max}}$. 
There are $n = 4q$ jobs with

$$(a_i, L_i, b_i) = (a, e_i, b) \quad \text{for } i = 1, \ldots, 3q,$$

$$(a_i, L_i, b_i) = (a, 13E, b) \quad \text{for } i = 3q + 1, \ldots, 4q,$$

where $a = b = 2E$.

Does there exist a schedule with $C_{\text{max}} \leq y$, where $y = 17qE$?

We refer to jobs $1, \ldots, 3q$ as partition jobs and to $3q + 1, \ldots, 4q$ as dividing jobs.

**Case 1:** We want to show that if 3-PARTITION has a solution, then $1|\text{Coup-Task}, a_i = b_i = p|C_{\text{max}}$ has a solution with $C_{\text{max}} \leq y$.

If there is a solution to 3-PARTITION, then there exists $Q_j$ for $j = 1, \ldots, q$ such that $\sum_{i \in Q_j} e_i = E$.

Schedule the jobs of $Q_j$ in the separation time of one of the dividing jobs, as shown in Fig. 8, where $Q_j = \{g, h, i\}$. The dividing jobs are scheduled without interleaving and without idle time between successive jobs.

Therefore,

$$C_{\text{max}} = \sum_{j=3q+1}^{4q} (a_j + L_j + b_j) = q(a + 13E + b) = y.$$  

**Case 2:** We want to show that if there is a schedule for $1|\text{Coup-Task}, a_i = b_i = p|C_{\text{max}}$ with $C_{\text{max}} \leq y$, then 3-PARTITION has a solution.

First note that a partition job cannot be interleaved with any other job. Thus the separation times for the partition jobs represent unavoidable idle time. Since $\sum_{i=1}^{4q} (a_i + b_i) + \sum_{i=1}^{3q} L_i = 4q(a + b) + qE = y$, the only idle time in a schedule with $C_{\text{max}} \leq y$ corresponds to the separation times for the partition jobs. In particular, all of the separation time for each dividing job must be occupied either by processing, or by separation time for the partition jobs.

We claim that exactly three partition jobs are scheduled in the separation time of each dividing job, and that the separation times of these three jobs sum exactly to $E$. Since the total processing time of each partition job is $4E$, no more than three partition jobs can be scheduled in the separation time of $13E$. If less than three partition jobs are scheduled in the separation time of a dividing job, then the bounds on $e_i$ in 3-PARTITION show that their separation times sum to less than $E$, which results in the separation time of $13E$ not being fully occupied. Thus, exactly three partition jobs are scheduled in the separation time of each dividing job. Moreover, the separation time of
13E must be occupied by 12E units of processing and E units of separation time for the three partition jobs. Thus, our claim is established, and the partition jobs scheduled in the separation times of dividing jobs define a 3-PARTITION. \(\square\)

**Corollary 2.** The 1\(|\text{Coup-Task}, a_i = a, b_i = b|C_{\text{max}}\) problem is unary NP-hard.

**Proof.** This follows because it contains the unary NP-hard problem 1\(|\text{Coup-Task}, a_i = b_i = p|C_{\text{max}}\) as a special case. \(\square\)

### 4. Polynomial time algorithms

Firstly, we concentrate on problem 1\(|\text{Coup-Task}, a_i = L_i = p, b_i|C_{\text{max}}\). It is possible to interleave jobs \(i\) and \(j\) if \(b_i \leqslant p\) as shown in the first schedule of Fig. 2. Note that the separation times are not large enough to allow more than two jobs to be interleaved with each other. Thus, jobs \(i\) and \(j\) in Fig. 2 contribute \(a_i + a_j + L_j + b_j = 3p + b_j\) to the makespan since \(a_i = a_j = L_i = p\). Any job \(k\) that cannot be interleaved contributes \(2p + b_k\) to the makespan. In any schedule, let \(I, J\) and \(K\) denote the sets of jobs that form the first of an interleaved pair, that form the second of an interleaved pair, and which are not interleaved, respectively. The makespan for this schedule is

\[
C_{\text{max}} = 3p|J| + 2p|K| + \sum_{h \in J \cup K} b_h,
\]

which can be expressed as

\[
C_{\text{max}} = 3p|J| + 2p|K| + \sum_{h=1}^{n} b_h - \sum_{h \in I} b_h. \tag{4}
\]

Thus, to obtain an optimal schedule as many jobs as possible should be interleaved, and the total (second task) processing time for the jobs of \(I\) should be as large as possible.

Let \(S = \{h | b_h \leqslant p\}\) and \(T = \{h | b_h > p\}\). If \(|S| \leq |T|\), then each job in \(S\) can be interleaved with a job in \(T\). On the other hand, if \(|S| > |T|\), then each job of \(T\) can be interleaved with a job of \(S\) (where the job of \(S\) is the first in the interleaved pair), while other jobs of \(S\) are interleaved with each other: if \(n\) is odd, then a single job remains that cannot be interleaved. The value of \(C_{\text{max}}\) in (1) when \(|S| > |T|\) is minimized by ensuring that the first of each interleaved pair of jobs has a (second task) processing time that is as large as possible. These features are included in the following algorithm.

**Algorithm 1**

1. **Step 1:** Partition the jobs into subsets \(S = \{h | b_h \leq p\}\) and \(T = \{h | b_h > p\}\). If \(|S| \leq |T|\), go to Step 4.
Step 2: If \( n \) is odd, choose \( l \in S \) for which \( b_l \) is as small as possible, and set \( S = S - \{ l \} \). Use a median finding technique to partition \( S \) into subsets \( S_1 \) and \( S_2 \), such that \( |S_1| = \lfloor n/2 \rfloor \) and \( |S_2| = \lfloor n/2 \rfloor - |T| \), and \( b_i \geq b_j \) for all \( i \in S_1 \) and \( j \in S_2 \).

Step 3: Interleave jobs by selecting a job of \( S_1 \) to be the first and a job of \( S_2 \cup T \) to be the second job of a pair. Schedule each pair contiguously. If \( n \) is odd, then append job \( l \) to the end of the schedule. Terminate.

Step 4: Interleave each job of \( S \) with a job of \( T \) and schedule each pair contiguously. If \( |S| < |T| \), then append the remaining jobs to the end of the schedule. Terminate.

We now show that Algorithm 1 generates an optimal schedule, and we derive its computational complexity.

**Theorem 5.** Algorithm 1 generates an optimal schedule for the \( 1 \mid \text{Coup-Task}, a_i = L_i = p \mid C_{\text{max}} \) problem in \( O(n) \) time.

**Proof.** If \( |S| > |T| \), then all jobs can be interleaved, unless \( n \) is odd in which case one job remains after interleaving. If \( i \in S \) is the first job of an interleaved pair, and \( j \in S \) is the second job of an interleaved pair, then Steps 2 and 3 of Algorithm 1 ensure that \( b_i \geq b_j \). Thus, (1) shows that any alternative interleaving of jobs cannot decrease the makespan. Alternatively, if \( |S| \leq |T| \), then at most \( |S| \) jobs can be interleaved and (1) shows that the makespan does not depend on how this interleaving is performed. Since Step 4 interleaves \( |S| \) jobs, Algorithm 1 generates an optimal schedule in this case also.

To derive the time complexity of Algorithm 1, we first note that Steps 1, 3 and 4 require \( O(n) \) time. To implement Step 2 efficiently, linear time median finding techniques (Schonhage et al. [7]) are used. To find sets \( S_1 \) and \( S_2 \) in Step 2 it is sufficient to find \( k \in S \) such that \( S'_1 = \{ h \mid h \in S, b_h > b_k \} \), \( S'_2 = \{ h \mid h \in S, b_h < b_k \} \), \( |S'_1| \leq n/2 \) and \( |S'_2 \cup T| < n/2 \). At a general stage of the procedure for finding job \( k \) in Step 2, it is known that \( R_1 \subseteq S'_1 \) and \( R_2 \subseteq S'_2 \) and \( R = S - R_1 \cup R_2 \), where initially \( R_1 = R_2 = \emptyset \). The median second stage processing time \( b \) for the jobs of \( R \) is computed, and sets \( \hat{S}_0 = \{ h \mid h \in R, b_h = b \} \), \( \hat{S}_1 = \{ h \mid h \in R, b_h > b \} \) and \( \hat{S}_2 = \{ h \mid h \in R, b_h < b \} \) are identified. There are three cases to be considered. Firstly, if \( |R_1 \cup \hat{S}_1| \leq n/2 \) and \( |R_2 \cup \hat{S}_2 \cup T| < n/2 \), then \( b_k = b \), \( S'_1 = R_1 \cup \hat{S}_1 \) and \( S'_2 = R_2 \cup \hat{S}_2 \), and the procedure terminates. Secondly, if \( |R_1 \cup \hat{S}_1| > n/2 \), then \( \hat{S}_0 \subseteq S'_2 \subseteq S_2 \), and the search for job \( k \) in Step 2 is restricted to jobs of \( \hat{S}_1 \). Thus, we set \( R_2 = R_2 \cup \hat{S}_0 \cup \hat{S}_2 \) and \( R = R - (\hat{S}_0 \cup \hat{S}_2) \). Thirdly, if \( |R_2 \cup \hat{S}_2 \cup T| > n/2 \), then \( \hat{S}_0 \cup \hat{S}_1 \subseteq S'_1 \), and the search for job \( k \) in Step 2 is restricted to jobs of \( \hat{S}_2 \). Thus, we set \( R_1 = R_1 \cup \hat{S}_0 \cup \hat{S}_1 \) and \( R = R - (\hat{S}_0 \cup \hat{S}_1) \). In the latter two cases, the updating of \( R \) removes at least half of the jobs of the previous iteration. Thus, \( O(\log n) \) iterations of the procedure are carried out over sets which contain at most \( n, n/2, n/4, \ldots \) jobs, and job \( k \) is found in \( O(n) \) time. We now deduce that the overall time requirement of Algorithm 1 is \( O(n) \).

**Corollary 3.** The \( 1 \mid \text{Coup-task}, a_i, L_i = b_i = p \mid C_{\text{max}} \) problem is solvable in \( O(n) \) time.
Proof. Theorems 1 and 5 show that an optimal solution is obtained in $O(n)$ time by applying Algorithm 1 to the inverse problem. □

For problem $1|Coup-Task, a_i = L_i = p, b_i = b|C_{max}$, all jobs are identical and Algorithm 1 simplifies. If $b > p$, then no interleaving is possible, and $C_{max} = n(a+L+b) = n(2p+b)$. On the other hand, if $b \leq p$, then all jobs, except for one in the case that $n$ is odd, are interleaved. Thus, $C_{max} = \lfloor n/2 \rfloor (3p+b) + (n-2\lfloor n/2 \rfloor)(2p+b)$. By setting $b = p$, we observe that the analysis for this latter case is applicable to $1|Coup-Task, a_i = L_i = b_i = p|C_{max}$.

Using Theorem 1, the above analysis for $1|Coup-Task, a_i = L_i = p, b_i = b|C_{max}$ is easily converted to the problem $1|Coup-Task, a_i = a, L_i = b_i = p|C_{max}$.

For the remainder of this section, we concentrate on the $1|Coup-Task, a_i = b_i = p, L_i = L|C_{max}$ problem. Assume without loss of generality that $L = mp + r$ for some non-negative integer $m$, where $0 \leq r < p$. We propose an algorithm in which jobs are scheduled in turn so that each starts as early as possible. This produces blocks of jobs, where each block contains $m+1$ contiguously scheduled first tasks, an idle time of $r$ (if $r > 0$), and $m+1$ contiguously scheduled second tasks. It takes time $2(m+1)p+r$ to process each block. At the end of the schedule, there may be a partial block containing less than $m+1$ jobs. Fig. 9 shows the structure of each complete block comprising contiguously scheduled jobs for $m = 3$. A formal statement of the algorithm is as follows.

Algorithm 2

**Step 1**: Compute $m = \lfloor L/p \rfloor$, $r = L - mp$ and $q = \lfloor n/(m + 1) \rfloor$.

**Step 2**: Form $q$ blocks of jobs, where each block contains $m+1$ jobs for which their first tasks are scheduled contiguously. If $n > (m+1)q$, then form a partial block containing all the remaining jobs where their first tasks are scheduled contiguously. Schedule the blocks and any partial block contiguously.

We now show that Algorithm 2 generates an optimal schedule, and we derive its computational complexity.

**Theorem 6.** Algorithm 2 generates an optimal schedule for the $1|Coup-Task, a_i = b_i = p, L_i = L|C_{max}$ problem in $O(n)$ time.
Proof. We prove first that there exists an optimal schedule in which, at times 0, $p, \ldots, h p$, where $h = \min\{m, n - 1\}$, the machine starts to process the first task of some job. Clearly, we may assume that some job starts its processing at time zero. Suppose that, in some optimal schedule $S$, no task starts processing at time $k p$, where $k \in \{1, \ldots, h\}$ and $k$ is chosen as small as possible. Note that this choice of $k$ ensures that a second stage task is processed in the interval $[k p + L, (k + 1) p + L]$. If some job starts its processing at time $t$, where $k p < t < (k + 1) p$, then this job can be rescheduled to start at time $k p$. This rescheduling is possible because the machine is idle in the interval $[(k + 1) p + L, t + p + L]$ which is too short to process a task. Alternatively, the machine is idle throughout the interval $[(k + 1) p, (k + 1) p + L]$. If the machine is also idle throughout the interval $[(k + 1) p + L, (k + 2) p + L]$, then any job that starts after time $(k + 1) p$ can be rescheduled to that its first and second tasks occupy the intervals $[(k + 1) p, (k + 1) p + L]$ and $[(k + 1) p + L, (k + 2) p + L]$, respectively. Thus, it remains to consider the case that the first task of some job starts its processing at time $t$, where $(k + 1) p + L \leq t < (k + 2) p + L$. However, this job can also be rescheduled so that its first and second tasks occupy the intervals $[k p, (k + 1) p]$ and $[(k + 1) p + L, (k + 2) p + L]$, respectively. Repetition of this argument shows that there is an optimal schedule in which first tasks of jobs are processed throughout the interval $[0, (h + 1) p]$, the machine is idle throughout the interval $[(h + 1) p, p + L]$, and second tasks of jobs are processed throughout the interval $[p + L, (h + 2) p + L]$.

Suppose that $n > m + 1$. Since the interval of machine idle time $[(h + 1) p, p + L]$ is too small to process any jobs, we may assume that the next job starts its processing at time $(h + 2) p + L$. Identical analysis so that used above establishes the existence of an optimal schedule in which the next block of $\min\{m + 1, n - m - 1\}$ jobs are scheduled as in Algorithm 2. Further repetitions of this argument to subsequent blocks and partial blocks of jobs establishes the required result. \(\square\)

5. Concluding remarks

We have shown that the coupled-task scheduling problem with makespan objective must have highly constrained values of $(a_i, L_i, b_i)$ before it becomes solvable in polynomial time. Theorems 3 and 4 showed that even if one of the above three fields is allowed to vary, then the problem remains unary NP-hard. Derivation of a polynomial time algorithm for the one open problem (with $a_i = a, L_i = L, b_i = b$ for each job $i$) would be an interesting addition to finish the complexity analysis.

Extending the analysis by considering other optimality criteria, it seems that most problems will be unary NP-hard. In particular, the NP-hardness of special cases of maximum lateness, (weighted) number of late jobs and total (weighted) tardiness problems can be deduced from the corresponding results for the makespan objective.

In the radar problem, the scheduling must be conducted, in real time, in a dynamic and stochastic environment and, since the transmitted and received pulses travel at the speed of light, making each task duration of the order of micro-seconds, the scheduling
must be done very quickly. A detailed simulation model of a radar environment has been developed which aids the analysis of the various scheduling heuristics which have evolved.

Acknowledgements

This research was funded by the DRA, research grant CPC 112/0262 PO 1848. The authors would like to thank anonymous referees for their suggested improvements to the paper.

References