Computers & Industrial Engineering 60 (2011) 690–698

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Coupled-task scheduling on a single machine subject to a fixed-job-sequence ☆

F.J. Hwang, Bertrand M.T. Lin *

Institute of Information Management, Department of Information and Finance Management, National Chiao Tung University, Hsinchu 300, Taiwan

ARTICLE INFO

Article history:
Received 7 September 2010
Received in revised form 27 December 2010
Accepted 4 January 2011
Available online 11 January 2011

Keywords:
Coupled tasks
Exact delays
Fixed-job-sequence
Makespan

ABSTRACT

This paper investigates single-machine coupled-task scheduling where each job has two tasks separated by an exact delay. The objective of this study is to schedule the tasks to minimize the makespan subject to a given job sequence. We introduce several intriguing properties of the fixed-job-sequence problem under study. While the complexity status of the studied problem remains open, an $O(n^2)$ algorithm is proposed to construct a feasible schedule attaining the minimum makespan for a given permutation of $2n$ tasks abiding by the fixed-job-sequence constraint. We investigate several polynomially solvable cases of the fixed-job-sequence problem and present a complexity graph of the problem.

1. Introduction

This paper considers the problem of scheduling n coupled tasks with exact delays on a single machine. Coupled-task scheduling, also known as the two-phased job scheduling problem (Sherali & Smith, 2005), primarily stems from operations scheduling of pulsed radar systems. A set of n two-phased jobs $\{J_1, J_2, \ldots, J_n\}$ is given to be processed on a single machine. Each two-phased job J_j consists of two separate tasks that require processing times a_j and b_j, respectively. If no confusion would arise, a_j and b_j are also used to denote the two tasks of job J_j. Under the constraint of exact delays, the starting time of the second task b_j of any job J_j *must* be exactly l_j time units after the completion of its first task a_j. The problem, denoted as $1|\text{Coup-Task}|C_{\max}$ by Orman and Potts (1997) and $1|\text{exact } l_j|C_{\max}$ by Ageev and Kononov (2006), is to find a feasible schedule such that the makespan is minimum. This problem is known to be strongly NP-hard even in some special cases (Orman & Potts, 1997). For some scheduling problems, researchers (Chen et al., 2000; Hwang et al., 2010; Lin & Hwang, 2011; Ng & Kovalyov, 2007; Shafransky & Strusevich, 1998) find that determining an optimal schedule from a given job sequence is not necessarily trivial. These interesting findings stimulate this study that aims to investigate coupled-task scheduling subject to a given job sequence. Herewith, given a fixed-job-sequence, if job J_i precedes job J_j in the specified sequence, then it is required to schedule the tasks such that a_i precedes a_j, and b_i precedes b_j. We adopt the notation of Blazewicz et al. (2010) to denote the studied problem by $1|(a_j, l_j, b_j), \text{fjs}|\, C_{\max}$, where "fjs" in the second field dictates the assumption of a fixed-job-sequence.

The first study on coupled-task scheduling with exact delays could be due to Shapiro (1980), who established that $1|(a_j, l_j, b_j)|C_{\max}$ is equivalent to the NP-hard jobshop problem $J2|\textit{no-wait}, M_2\ \textit{non-bott}|C_{\max}$, where "*no-wait*" and "M_2 *non-bott*" respectively refer to the no-wait constraint and the infinite processing capacity of the second machine. Three polynomial-bounded heuristics for numerical experiments were also presented. Orman and Potts (1997) investigated the complexity of several special cases of $1|(a_j, l_j, b_j)|C_{\max}$. All the analyzed cases are classified to be strongly NP-hard or polynomially solvable, except for the case of identical coupled tasks, $1|(a, l, b)|C_{\max}$. Ahr et al. (2004) proposed a dynamic programming algorithm based on a directed graph model for this special case with time complexity $O(nr^{2l})$, where $r \leqslant \sqrt[a-1]{a}$. The algorithm is linear in the number of jobs only for fixed l and is not polynomial in the input size which is measured by $\log a + \log l + \log b + \log n$. Then Baptiste (2010) showed that the case can be solved in $O(\log n)$ when a, l, b are fixed. To the best of our knowledge, the complexity status of identical coupled-task scheduling problem remains open. Blazewicz et al. (2010) studied $1|(1, l, 1), \textit{prec}|C_{\max}$ with strict precedence constraints and proved its NP-hardness in the strong sense. They also proposed an $O(n)$ algorithm for the special case of $l = 2$ and an in-tree or out-tree precedence constraints graph. Ageev and Kononov (2006) designed a 3.5-approximation algorithms for $1|(a_j, l_j, b_j)|C_{\max}$ and proved that a $(2-\epsilon)$ approximation algorithm does not exist unless P = NP. Yu et al. (2004) implied the strong NP-hardness of $1|(1, l_j, 1)|C_{\max}$ from the strong NP-hardness proof of $F2|(1,l_j,1)|C_{\max}$. Ageev and Baburin (2007) designed a 7/4-approximation algorithm for $1|(1,l_j,1)|C_{\max}$.

☆ This manuscript was processed by Area Editor T.C. Edwin Cheng.

* Corresponding author. Tel.: +886 3 5131472; fax: +886 3 5723792.
E-mail addresses: fjhwang.iim95g@nctu.edu.tw (F.J. Hwang), bmtlin@mail.nctu.edu.tw (B.M.T. Lin).

0360-8352/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2011.01.002

Nomenclature

π	a given plausible sequence of $2n$ tasks, $\pi = (o_1, o_2, \ldots, o_{2n})$
o_h	the task assigned to the hth position in π
σ	a schedule of $2n$ tasks
$s_j(\sigma)$	the starting time of job J_j in a schedule σ
$C_j(\sigma)$	the completion time of job J_j in a schedule σ
k	the number of segments in the task sequence $(a_1, a_2, \ldots, a_n)$
X	the sequence of subscripts $(1, 2, \ldots, n)$
H	a k-subsequence partition of X corresponding to the partition of $(a_1, a_2, \ldots, a_n)$
X_r	the r-th subsequence in X, $X_r = (n_{r-1} + 1, \ldots, n_r)$, where $n_0 = 0$ and $n_k = n$
$\widetilde{X}_r$	the set of elements in sequence X_r
$\lvert\widetilde{X}_r\rvert$	the length of X_r, $\lvert\widetilde{X}_r\rvert = n_r - n_{r-1}$
$b_{n'_r}$	the immediate predecessor of a_{n_r+1} in π
π_r	the rth fundamental cluster in π, $\pi_r = (a_{n_{r-1}+1}, \ldots, a_{n_r}, b_{n_{r-1}+1}, \ldots, b_{n_r})$
χ_r	the subsequence obtained by deleting the jobs of $\{J_j \mid j \in \widetilde{X}_{r+1} \cup \ldots \cup \widetilde{X}_k\}$ from sequence π
σ_r	the schedule of π_r constructed by the developed recursive formula
σ_{π_r}	a feasible schedule whose permutation of tasks agrees with π_r
α_r	the time span from the start of $a_{n_{r-1}+1}$ to the completion of a_{n_r}
γ_r	the idle time between a_{n_r} and $b_{n_{r-1}+1}$
S_r	a subschedule constructed by arranging the first r subschedules, $\sigma_1, \ldots, \sigma_r$
β_r^1	the idle time between $b_{n'_r}$ and $b_{n'_r+1}$ in subschedule S_r
β_r^2	the time span from the start of $b_{n'_r+1}$ to the completion of b_{n_r}
μ	the input task of **Checking routine** in **Algorithm Plausible-Task-Sequence**
ν	the task preceded by μ in χ_{l+1}
$\tilde{\nu}$	the corresponding first or second counterpart task of ν

Subsequently, Békési, Galambos, Oswald, and Reinelt (2009) improved the analysis of Ageev and Baburin (2007) to derive a better lower bound of the approximation ratio. Furthermore, Li and Zhao (2007) designed approximation algorithms for some NP-hard special cases, and developed a tabu search meta-heuristic for the general case.

In scheduling theory, sequences of jobs or operations indicate the order of processing on machines while schedules explicitly specify the starting and completion times of activities on specific machines. In most scheduling problems, schedules can be directly determined by sequences of jobs or operations on the machines involved in the problems. However, determining an optimal schedule from a given sequence could not be straightforward for some problems because other decisions such as batching (Cheng et al., 2000; Hwang et al., 2010; Ng & Kovalyov, 2007), interleaving (Lin & Hwang, 2011), and idle time insertion (Hwang et al., 2010) would be needed for optimality. For the coupled-task problem, a predetermined job sequence defines a sequence of the first tasks of all jobs and the same sequence of the second tasks of all jobs. To construct a feasible schedule subject to a fixed-job-sequence, the decision is how to interleave the task-1 sequence and the task-2 sequence.

In real-world applications, a pre-assigned sequence of jobs could be retained on one of the machines in manufacturing process owing to technological or managerial decisions (Shafransky & Strusevich, 1998). Another justification for the assumption of a fixed-job-sequence is due to the Fist-Come–First-Served (FCFS) rule, which is regarded fair by customers. From the theoretical aspect, one approach to tackle the NP-hard scheduling problem in which a schedule cannot be readily induced from a job sequence is to develop an optimal polynomial-time algorithm for its fixed-job-sequence problem. Then a heuristic or local search for the problem can exploit this algorithm to evaluate candidate job sequences. Due to the strong NP-hardness of $1|(a_j, l_j, b_j,)\ |C_{\max}$, it is interesting to study $1|(a_j, l_j, b_j,)$, fis $|C_{\max}$.

The plan of this paper is as follows. Several intriguing properties of the fixed-job-sequence problem are expounded in detail in Section 2. In Section 3, a polynomial-time algorithm is presented to construct a schedule with the minimum makespan for a given *task* sequence abiding by the fixed-job-sequence constraint. Three polynomially solvable cases for the fixed-job-sequence problem are studied in Section 4. In the last section, we conclude this note and suggest some subjects for further research.

2. Problem description and notation

Without loss of generality, we assume that the fixed-job-sequence is $(J_1, J_2, \ldots, J_n)$. Subject to the constraint of a fixed-job-sequence and the definition of coupled tasks, we thus have a directed ladder graph of precedence relationship with two long chains, $a_1 \to a_2 \to \cdots \to a_n$ and $b_1 \to b_2 \to \cdots \to b_n$, and n single-arc chains, $a_j \to b_j$ for all $j \in \mathbb{N}_n$. As a permutation of $\{a_j \mid j \in \mathbb{N}_n\} \cup \{b_j \mid j \in \mathbb{N}_n\}$, a task sequence is called *plausible* if it adheres to the precedence constraints given by the ladder graph. We first give an initial idea about how to generate a plausible task sequence. By virtue of the diagonal-avoiding paths (Davis, 2006), the following observation is presented.

Observation 1. Given n jobs, all plausible task sequences can be generated by the diagonal-avoiding paths along the edges of a grid with $n \times n$ square cells. Each diagonal-avoiding path corresponds to exactly one plausible task sequence.

A diagonal-avoiding path is the one which leads from the top-left corner O to the bottom-right corner D without backtracking, and stays on or above the diagonal without passing below it. As shown in Fig. 1 for the case $n = 5$, the illustrated diagonal-avoiding path corresponds to the plausible task sequence $(a_1, a_2, b_1, a_3, b_2, b_3, a_4, a_5, b_4, b_5)$. The number of diagonal-avoiding paths in a grid of $n \times n$ squares is given by the well-known Catalan number, $\mathscr{C}_n = \frac{1}{n+1}\binom{2n}{n}$, which grows in the order of $\Omega(4^n/\sqrt{n^3})$.

Previous experience suggests that scheduling problems with fixed-job-sequences could be resolved by dynamic programs for

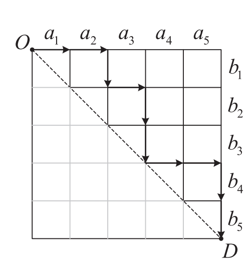

Fig. 1. The diagonal-avoiding path corresponding to the plausible task sequence $(a_1, a_2, b_1, a_3, b_2, b_3, a_4, a_5, b_4, b_5)$.

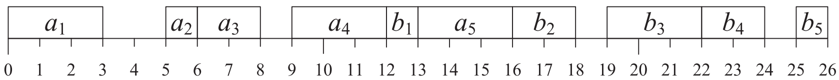

Fig. 2. The optimal schedule of the instance with $(a_1, l_1, b_1) = (3, 9, 1)$, $(a_2, l_2, b_2) = (1, 10, 2)$, $(a_3, l_3, b_3) = (2, 11, 3)$, $(a_4, l_4, b_4) = (3, 10, 2)$, $(a_5, l_5, b_5) = (3, 9, 1)$.

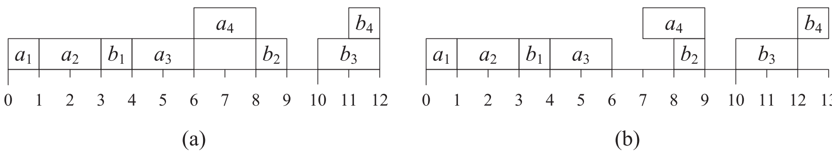

Fig. 3. An inevitable task overlap happens (a) between b_3 and b_4 or (b) between a_4 and b_2.

the objective of makespan or total completion time (Cheng et al., 2000; Hwang et al., 2010; Lin & Hwang, 2011; Ng & Kovalyov, 2007). However, it seems not to be the case for $1|(a_j, l_j, b_j)$, fjs $|C_{\max}$. The intrigue could be ascribed to the following two causes:

1. Although the job sequence is pre-assigned, the studied problem remains a problem of sequencing in which there are $\frac{1}{n+1}\binom{2n}{n}$ plausible task sequences for n jobs.
2. The time lag l_j between tasks a_j and b_j can accommodate the processing of not only tasks $\{a_{j+1}, a_{j+2}, \ldots, a_n\}$ but also tasks $\{b_1, b_2, \ldots, b_{j-1}\}$. Due to the distinctive scheduling pattern, the principle of optimality fails. Thus, it becomes not clear whether a dynamic programming approach will work. Take for example the following instance with five jobs: $(a_1, l_1, b_1) = (3, 9, 1)$, $(a_2, l_2, b_2) = (1, 10, 2)$, $(a_3, l_3, b_3) = (2, 11, 3)$, $(a_4, l_4, b_4) = (3, 10, 2)$, $(a_5, l_5, b_5) = (3, 9, 1)$. The optimal schedule for the fixed-job-sequence problem is demonstrated in Fig. 2. In the optimal schedule, the subschedule of the subsequence (J_1, J_2) attains the time span equal to 18. But the time span of the shortest subschedule constructed with the subsequence (J_1, J_2) is 16. Thus, a subschedule within the shortest complete schedule is not necessarily a shortest subschedule.

Owing to the intrigue of the fixed-job-sequence problem, we turn to aim at scheduling a given plausible task sequence in the next section.

3. Scheduling of plausible task sequences

Discussion in the previous section introduces the notion of $\frac{1}{n+1}\binom{2n}{n}$ plausible task sequence for a given job sequence. This section is dedicated to the development of a polynomial-time algorithm for determining the makespan of a plausible task sequence, if it is feasible. Given a plausible task sequence, it could be non-trivial to determine its feasibility and a schedule with the minimum makespan, if feasible.

Denote a plausible task sequence by $\pi = (o_1, o_2, \ldots, o_{2n})$, where o_h stands for the task assigned to the hth position in π. If no confusion would arise, hereafter we simply mention sequences to indicate plausible task sequences. Notice that in any schedule we consider hereafter, the constraint of exact delays is satisfied. In other words, the interval between each pair of coupled tasks a_j and b_j in any schedule is exactly l_j, $j \in \mathbb{N}_n$. Denote the starting time and the completion time of job J_j in a schedule σ by $s_j(\sigma)$ and $C_j(\sigma)$, respectively. It is obvious that $C_j(\sigma) = s_j(\sigma) + a_j + l_j + b_j$. A schedule σ is feasible if and only if at any time, at most one task is processed in σ, i.e. no overlap between tasks occurs. Sequence π is called *feasible* if and only if there exists a feasible schedule whose permutation of tasks agrees with π, i.e. a schedule in which the processing of any task o_h for $h \in \mathbb{N}_{2n-1}$ completes earlier than or exactly at the starting time of task o_{h+1}. We first consider how to determine the feasibility of a given sequence π. For any feasible sequence π, the constraint of exact delays implies that the interval induced by the exact delay l_j of any job J_j must accommodate all the tasks arranged between a_j and b_j. Namely, the following condition is necessary for the feasibility of a sequence π:

Condition (C). For any job J_j with $o_\ell = a_j$ and $o_g = b_j$, the inequality $\sum_{h=\ell+1}^{g-1} p_{o_h} \leqslant l_j$ must hold, where p_{o_h} is the processing length of task o_h.

Note that condition (C) is not sufficient to make sequence π feasible. Consider the following instance of four jobs: $(a_1, l_1, b_1) = (1, 2, 1)$, $(a_2, l_2, b_2) = (2, 5, 1)$, $(a_3, l_3, b_3) = (2, 4, 2)$, $(a_4, l_4, b_4) = (2, 3, 1)$. Condition (C) holds for sequence $\pi = (a_1, a_2, b_1, a_3, a_4, b_2, b_3, b_4)$. However, π is infeasible since an overlap between b_3 and b_4 (Fig. 3(a)) or between a_4 and b_2 (Fig. 3(b)) is inevitable in any attempt to create a feasible schedule of π.

Condition (C) only partially verifies the feasibility of π because in a schedule whether an idle time or overlap exists between o_h and o_{h+1} cannot be detected before assigning each task a starting time. Therefore, we turn to develop a procedure for constructing a schedule for sequence π and prove that the feasibility of π can be determined by the constructed schedule. If sequence π is indeed feasible, we can further prove that the constructed schedule attains the minimum makespan among those of feasible schedules.

We first introduce subsequences of a particular permutation pattern

$$(a_{i_1}, a_{i_1+1}, \ldots, a_{i_2}, b_{i_1}, b_{i_1+1}, \ldots, b_{i_2}),$$

$1 \leqslant i_1 \leqslant i_2 \leqslant n$, where all its first (respectively, second) tasks are consecutively sequenced without any second (respectively, first) task inserted. A given sequence π is derived by merging several subsequences of this pattern. Consider the sequence $\pi = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, a_6, b_4, a_7, a_8, b_5, b_6, b_7, b_8)$ as an example. As shown in Fig. 4, it can be regarded as the outcome of four interleaved subsequences (a_1, a_2, b_1, b_2), $(a_3, a_4, a_5, b_3, b_4, b_5)$, (a_6, b_6) and (a_7, a_8, b_7, b_8). Such particular subsequences are regarded as the maxi-

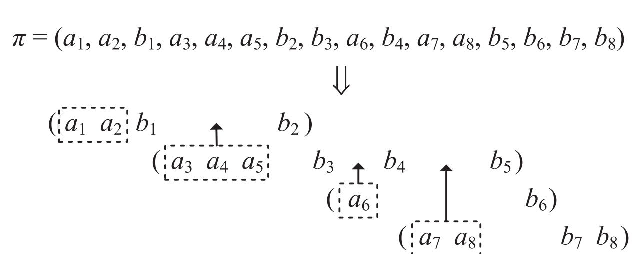

Fig. 4. The given sequence π consists of four subsequences of a particular pattern.

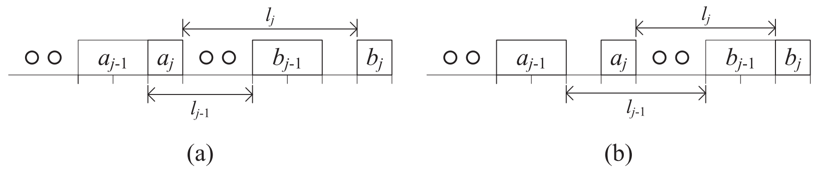

Fig. 5. J_j is interleaved with J_{j-1} by conjoining (a) a_{j-1} and a_j or (b) b_{j-1} and b_j.

mal *fundamental clusters* of a given sequence π and these subsequences are scheduled individually. Scheduling these fundamental clusters is the first attempt to examine the feasibility of sequence π. Later we will elucidate that the infeasibility of any fundamental cluster leads to the infeasibility of π. If all these fundamental clusters are feasible, then we proceed to schedule π by interleaving those obtained subschedules.

Now we define some notations for collating fundamental clusters from sequence π. Given a sequence π, a *segment* is defined as a maximal, by inclusion, subsequence of tasks $\{a_j\}$ without inserted tasks $\{b_i\}$. Assume that the task sequence $(a_1, a_2, \ldots, a_n)$ is partitioned into k disjoint segments for $1 \leqslant k \leqslant n$. To facilitate discussion, we denote by X the sequence of subscripts $(1, 2, \ldots, n)$ and by H a k-subsequence partition of X corresponding to the k-segment partition of $(a_1, a_2, \ldots, a_n)$. Partitioning X into k disjoint subsequences, we have $H = \{X_1, X_2, \ldots, X_k\}$ and $X = X_1 \oplus X_2 \oplus \cdots \oplus X_k$, where X_r denotes the rth subsequence in X and $\oplus$ is a sequence concatenation operator. For $r \in \mathbb{N}_k$, the last element of subsequence X_r is denoted by n_r, and we have $X_r = (n_{r-1} + 1, \ldots, n_r)$, where $n_0 = 0$ and $n_k = n$. Denote by $\widetilde{X}_r$ the set of elements in sequence X_r. $|\widetilde{X}_r| = n_r - n_{r-1}$ indicates the length of X_r. Denote by $b_{n'_r}$ the immediate predecessor of a_{n_r+1} in π for $r \in \mathbb{N}_{k-1}$ and $n'_{r-1} + 1 \leqslant n'_r \leqslant n_r$, where $n'_0 = 0$. Notice that any single task $a_{n_{r-1}+1} = a_{n_r}$, which is surrounded by two tasks $b_{n'_{r-1}}$ and $b_{n'_{r-1}+1}$ in π, forms a segment, i.e. $|\widetilde{X}_r| = 1$. According to the assumption of k segments, sequence π consists of k fundamental clusters in which the rth one contains jobs $\{J_j | j \in \widetilde{X}_r\}$, $r \in \mathbb{N}_k$. Denote the rth fundamental cluster in π by $\pi_r = (a_{n_{r-1}+1}, \ldots, a_{n_r}, b_{n_{r-1}+1}, \ldots, b_{n_r})$, $r \in \mathbb{N}_k$. The subsequence obtained by eliminating the jobs of $\{J_j | j \in \widetilde{X}_{r+1} \cup \cdots \cup \widetilde{X}_k\}$ from π is denoted by χ_r, $r \in \mathbb{N}_{k-1}$. Note that $\chi_k = \pi$.

To construct a schedule of fundamental cluster π_r, we propose a recursive procedure to augment the subschedule job by job, instead of task by task. Namely, coupled tasks a_j and b_j are simultaneously added into the subschedule of jobs $(J_{n_{r-1}+1}, J_{n_{r-1}+2}, \ldots, J_{j-1})$, $j \in \{n_{r-1} + 2, \ldots, n_r\}$, in each recursion step. In the proposed procedure, job J_j is interleaved with job J_{j-1} by conjoining two first tasks a_{j-1} and a_j (Fig. 5(a)) or two second tasks b_{j-1} and b_j (Fig. 5b). The obtained schedule is denoted by σ_r, and the recursive formula for the job starting times is given as follows:

$$s_j(\sigma_r) = \begin{cases} 0, & j = n_{r-1} + 1; \\ s_{j-1}(\sigma_r) + a_{j-1} & \\ + \max\{0, l_{j-1} + b_{j-1} - a_j - l_j\}, & n_{r-1} + 2 \leqslant j \leqslant n_r. \end{cases} \tag{1}$$

Eq. (1) implies that in σ_r task a_j (respectively, b_j) is started later than or exactly at the completion of a_{j-1} (respectively, b_{j-1}). Schedule σ_r is a feasible schedule of π_r if task a_{n_r} completes earlier than or exactly at the start of task $b_{n_{r-1}+1}$. The following lemma gives structural properties of fundamental clusters.

Lemma 1. *Given a subsequence $\pi_r = (a_{n_{r-1}+1}, \ldots, a_{n_r}, b_{n_{r-1}+1}, \ldots, b_{n_r})$, the following three properties hold: (i) If $s_{n_r}(\sigma_r) + a_{n_r} > C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$, then π_r is infeasible. (ii) If $s_{n_r}(\sigma_r) + a_{n_r} \leqslant C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$, then π_r is feasible and σ_r is a feasible schedule attaining the minimum makespan among those of all feasible schedules of π_r. (iii) The feasibility and the shortest schedule, if feasible, can be determined in $O\left(\left|\widetilde{X}_r\right|\right)$ time.*

Proof. If $s_{n_r}(\sigma_r) + a_{n_r} > C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$, then task a_{n_r} completes later than the start of task $b_{n_{r-1}+1}$ in σ_r. The only possible way to find a feasible schedule of π_r is to process a_{n_r} earlier or process $b_{n_{r-1}+1}$ later. In schedule σ_r, task a_j starts exactly at the completion of a_{j-1}, or task b_j starts exactly at the completion of b_{j-1}, $j \in \{n_{r-1} + 2, \ldots, n_r\}$. Starting J_{n_r} earlier or $J_{n_{r-1}+1}$ later finally results in the shifting of the whole schedule, which is futile. Therefore, a feasible schedule of π_r does not exist and π_r is infeasible. Property (*i*) proved.

Property (*ii*) is concerned about the feasibility of π_r and the optimality of σ_r. As for the feasibility of π_r, the inequality $s_{n_r}(\sigma_r) + a_{n_r} \leqslant C_{n_{r-1}+1}(\sigma_r) - b_{n_{r-1}+1}$ indicates that task a_{n_r} completes earlier than or exactly at the starting time of task $b_{n_{r-1}+1}$ in σ_r. Therefore, a feasible schedule of π_r, i.e. σ_r, exists and π_r is feasible. Next, we will show that schedule σ_r attains the minimum makespan for sequence π_r, i.e. $C_{n_r}(\sigma_r) \leqslant C_{n_r}(\sigma_{\pi_r})$, where σ_{π_r} denotes any feasible schedule of π_r. This inequality can be proved by induction on n_r. We derive the following recursive equation for $C_j(\sigma_r)$ by adapting Eq. (1).

$$C_j(\sigma_r) = \begin{cases} a_{n_{r-1}+1} + l_{n_{r-1}+1} + b_{n_{r-1}+1}, & j = n_{r-1} + 1; \\ C_{j-1}(\sigma_r) + b_j & \\ + \max\{0, a_j + l_j - l_{j-1} - b_{j-1}\}, & n_{r-1} + 2 \leqslant j \leqslant n_r. \end{cases} \tag{2}$$

We first consider the induction base $\left|\widetilde{X}_r\right| = 2$ and $\pi_r = (a_{n_{r-1}+1}, a_{n_{r-1}+2}, b_{n_{r-1}+1}, b_{n_{r-1}+2})$. If $a_{n_{r-1}+2} + l_{n_{r-1}+2} > l_{n_{r-1}+1} + b_{n_{r-1}+1}$, then it is impossible to interleave $J_{n_{r-1}+1}$ and $J_{n_{r-1}+2}$ with a completion time less than $a_{n_{r-1}+1} + a_{n_{r-1}+2} + l_{n_{r-1}+2} + b_{n_{r-1}+2}$, which is equal to $C_{n_{r-1}+2}(\sigma_r)$ from Eq. (2). On the other hand, if $a_{n_{r-1}+2} + l_{n_{r-1}+2} \leqslant l_{n_{r-1}+1} + b_{n_{r-1}+1}$, then there exists no feasible schedule σ_{π_r} where $J_{n_{r-1}+2}$ completes earlier than $a_{n_{r-1}+1} + l_{n_{r-1}+1} + b_{n_{r-1}+1} + b_{n_{r-1}+2} = C_{n_{r-1}+2}(\sigma_r)$ from Eq. (2). Hence, we have the induction base $C_{n_{r-1}+2}(\sigma_r) \leqslant C_{n_{r-1}+2}(\sigma_{\pi_r})$.

Assume, as the induction hypothesis, that the inequality holds for $\left|\widetilde{X}_r\right| = i > 2$, i.e. $C_{n_{r-1}+i}(\sigma_r) \leqslant C_{n_{r-1}+i}(\sigma_{\pi_r})$. To facilitate the notation, we denote $m = n_{r-1} + i$. If $a_{m+1} + l_{m+1} > l_m + b_m$, then the minimum time span from the completion time of J_m to that of J_{m+1} is $a_{m+1} + l_{m+1} + b_{m+1} - l_m - b_m$. In case of $a_{m+1} + l_{m+1} \leqslant l_m + b_m$, the minimum aforementioned time span is b_{m+1}. By Eq. (2), we have $C_{m+1}(\sigma_r) = C_m(\sigma_r) + b_{m+1} + a_{m+1} + l_{m+1} - l_m - b_m$ for $a_{m+1} + l_{m+1} > l_m + b_m$ and $C_{m+1}(\sigma_r) = C_m(\sigma_r) + b_{m+1}$ for $a_{m+1} + l_{m+1} \leqslant l_m + b_m$. Hence, $C_{m+1}(\sigma_r)$ is less than or equal to the completion time of J_{m+1} in any feasible schedule σ_{π_r}, i.e. $C_{m+1}(\sigma_r) \leqslant C_{m+1}(\sigma_{\pi_r})$. By induction, the inequality $C_{n_r}(\sigma_r) \leqslant C_{n_r}(\sigma_{\pi_r})$ is established. The proof of Property (*ii*) is done.

For schedule π_r, the feasibility and the shortest schedule, if feasible, can be obtained by the values of $s_j(\sigma_r)$ for $j = n_{r-1} + 1, \ldots, n_r$. By virtue of Eq. (1), the calculation involves $\left|\widetilde{X}_r\right| - 1$ iterations, each of which requires constant time. Therefore, either a feasible schedule with the minimum makespan or the infeasibility of sequence π_r can be determined in $O\left(\left|\widetilde{X}_r\right|\right)$ time. □

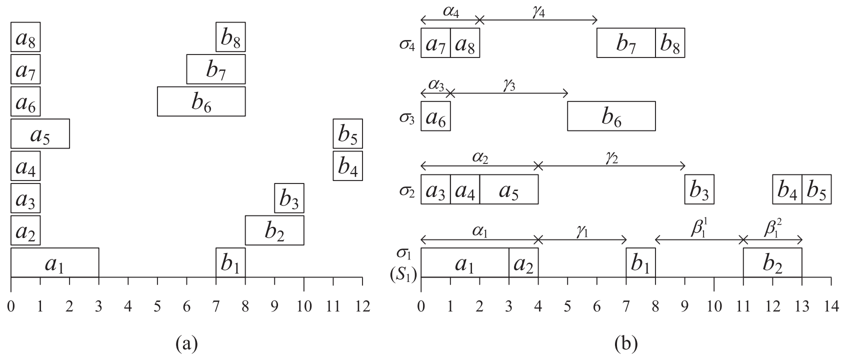

Fig. 6. (a) The instance of eight jobs and (b) subschedules $\sigma_1(=S_1)$, σ_2, σ_3 and σ_4.

The infeasibility of fundamental cluster π_r implies that there exists no feasible subschedule whose task permutation agrees with subsequence π_r, $r \in \mathbb{N}_k$. Since the subsequence π_r is a part of complete sequence π, no feasible complete schedule of π exists either. We therefore have the following property:

Property 1. *If any fundamental cluster $\pi_r, r \in \mathbb{N}_k$, is infeasible, then sequence π is infeasible.*

Before presenting a step-wise procedure for determining the infeasibility or the shortest feasible schedule of sequence π, we define some notations. For each schedule σ_r ($r \in \mathbb{N}_k$), denote by α_r the time span from the start of $a_{n_{r-1}+1}$ to the completion of a_{n_r}, and γ_r the idle time between a_{n_r} and $b_{n_{r-1}+1}$. S_r denotes a subschedule constructed by arranging the first r subschedules, $\sigma_1, \ldots, \sigma_r$. Note that S_k is the constructed complete schedule of sequence π, which consists of k clusters. Denote by β_r^1 the idle time between $b_{n'_r}$ and $b_{n'_r+1}$ in subschedule S_r, and by β_r^2 the time span from the start of $b_{n'_r+1}$ to the completion of b_{n_r}.

Algorithm Plausible-Task-Sequence

Step 1. Scan π to obtain the partition $H = (X_1, \ldots, X_k)$ and keep track of n_r, n'_r and χ_r for each $r = 1, \ldots, k-1$.

Step 2. Construct a schedule for π_r by Eq. (1) for each $r = 1, \ldots, k$. If any π_r is infeasible, then go to **Step 7**. Otherwise, set $I = 1$ and $S_I = \sigma_1$.

Step 3. If $n'_I = n_I$, then merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of b_{n_I} and go to **Step 6**.

Step 4. If $\beta_I^1 \geqslant \alpha_{I+1}$ and $\beta_I^2 \leqslant \gamma_{I+1}$, then go to **Step 4(a)**. If $\beta_I^1 < \alpha_{I+1}$ and $\beta_I^2 \leqslant \gamma_{I+1}$, then go to **Step 4(b)**. If $\beta_I^1 \geqslant \alpha_{I+1}$ and $\beta_I^2 > \gamma_{I+1}$, then go to **Step 4(c)**. Otherwise, go to **Step 4(d)**.

Step 4(a). If $\beta_I^1 + \beta_I^2 \leqslant \alpha_{I+1} + \gamma_{I+1}$, then merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of $b_{n'_I}$. Otherwise, merge S_I with σ_{I+1} by appending b_{n_I+1} to the end of b_{n_I}. Go to **Step 6**.

Step 4(b). Shift $b_{n'_I+1}$ (and $a_{n'_I+1}$ will be simultaneously shifted, i.e. shift $J_{n'_I+1}$) to extend β_I^1 such that $\beta_I^1 = \alpha_{I+1}$ and merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of $b_{n'_I}$. Go to **Step 5**.

Step 4(c). Shift $J_{n'_I+1}$ to shorten β_I^2 such that $\beta_I^2 = \gamma_{I+1}$ and merge S_I with σ_{I+1} by appending b_{n_I+1} to the end of b_{n_I}. Go to **Step 5**.

Step 4(d). If $\beta_I^1 + \beta_I^2 \leqslant \alpha_{I+1} + \gamma_{I+1}$, then shift $J_{n'_I+1}$ to extend β_I^1 such that $\beta_I^1 = \alpha_{I+1}$ and merge S_I with σ_{I+1} by appending a_{n_I+1} to the end of $b_{n'_I}$. Otherwise, shift $J_{n'_I+1}$ to shorten β_I^2 such that $\beta_I^2 = \gamma_{I+1}$ and merge S_I with σ_{I+1} by appending b_{n_I+1} to the end of b_{n_I}. Go to **Step 5**.

Step 5. Call **Checking routine** with input $b_{n'_I+1}$. Call **Checking routine** with input $a_{n'_I+1}$.

Step 6. Let S_{I+1} be the obtained schedule. If $I = k-1$, then output the schedule S_{I+1} and **stop**. Otherwise, set $I = I + 1$ and go to **Step 3**.

Step 7. Report the infeasibility of π and **stop**.

Checking routine. Denote the input task as μ, the task preceded by μ in χ_{I+1} as ν, and the corresponding first or second counterpart task of ν as $\tilde{\nu}$. If $\nu = b_1$ or a_{n_I+1} or b_{n_I+1}, then go to **Final checking**. Otherwise, go to **Checking and shifting**.

Final checking: If the completion of μ is later than the start of ν, then go to **Step 7**. Otherwise, terminate the subroutine.

Checking and shifting: If the completion of μ is later than the start of ν, shift ν (and $\tilde{\nu}$ will be simultaneously shifted) such that the task ν starts at exactly the completion of μ, call **Checking routine** with input ν, and call again **Checking routine** with input $\tilde{\nu}$. Otherwise, terminate the subroutine.

Example. *Consider an instance of eight jobs with the following parameters (Fig. 6a): $(a_1, l_1, b_1) = (3, 4, 1)$, $(a_2, l_2, b_2) = (1, 7, 2)$, $(a_3, l_3, b_3) = (1, 8, 1)$, $(a_4, l_4, b_4) = (1, 10, 1)$, $(a_5, l_5, b_5) = (2, 9, 1)$, $(a_6, l_6, b_6) = (1, 4, 3)$, $(a_7, l_7, b_7) = (1, 5, 2)$, $(a_8, l_8, b_8) = (1, 6, 1)$. The sequence $\pi = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, a_6, b_4, a_7, a_8, b_5, b_6, b_7, b_8)$ is given. Constructing a feasible schedule σ_π attaining the minimum makespan with* **Algorithm Plausible-Task-Sequence** *is demonstrated step by step as follows:*

Step 1. We obtain $k = 4$ $X_1 = (1,2)$, $X_2 = (3,4,5)$, $X_3 = (6)$, $X_4 = (7,8)$, $n_1 = 2$, $n_2 = 5$, $n_3 = 6$, $n'_1 = 1$, $n'_2 = 3$, $n'_3 = 4$, $\chi_1 = (a_1, a_2, b_1, b_2)$, $\chi_2 = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, b_4, b_5)$, and $\chi_3 = (a_1, a_2, b_1, a_3, a_4, a_5, b_2, b_3, a_6, b_4, b_5, b_6)$ as shown in Fig. 4.

Step 2. Feasible subschedules $\sigma_1, \sigma_2, \sigma_3$ and σ_4 are derived as shown in Fig. 6b. Set $I = 1$ and $S_1 = \sigma_1$.

Step 3. Since $n'_1 = 1 < n_1 = 2$, we go to **Step 4**.

Step 4. Since $\beta_1^1 = 3 < \alpha_2 = 4$ and $\beta_1^2 = 2 < \gamma_2 = 5$, we go to **Step 4(b)**.

Step 4(b). Shift J_2 such that $\beta_1^1 = \alpha_2 = 4$ and merge S_1 with σ_2 by appending a_3 to the end of b_1. Go to **Step 5**.

Step 5. Call **Checking routine** with input b_2. Call **Checking routine** with input a_2.

Checking routine. We have $\mu = b_2$, $\nu = b_3$, and $\tilde{\nu} = a_3$. Go to **Final checking**.

Final checking: Task b_2 completes (at 14) earlier than the start of b_3 (at 17). Terminate the subroutine.

Checking routine. We have $\mu = a_2$, $\nu = b_1$, and $\tilde{\nu} = a_1$. Go to **Final checking**.

Final checking: Task a_2 completes (at 5) earlier than the start of b_1 (at 7). Terminate the subroutine.

Step 6. The obtained schedule is S_2 (Fig. 7). Set $I = 2$ and go to **Step 3**.

Step 3. Since $n'_2 = 3 < n_2 = 5$, we go to **Step 4**.

Step 4. Since $\beta_2^1 = 2 > \alpha_3 = 1$ and $\beta_2^2 = 2 < \gamma_3 = 4$, we go to **Step 4(a)**.

Step 4(a). Since $\beta_2^1 + \beta_2^2 = 4 < \alpha_3 + \gamma_3 = 5$, we merge S_2 with σ_3 by appending a_6 to the end of b_3. Go to **Step 6**.

Step 6. The obtained schedule is S_3 (Fig. 8). Set $I = 3$ and go to **Step 3**.

Step 3. Since $n'_3 = 4 < n_3 = 6$, we go to **Step 4**.

Step 4. Since $\beta_3^1 = 0 < \alpha_4 = 2$ and $\beta_3^2 = 5 > \gamma_4 = 4$, we go to **Step 4(d)**.

Step 4(d). Since $\beta_3^1 + \beta_3^2 = 5 < \alpha_4 + \gamma_4 = 6$, we shift J_5 such that $\beta_3^1 = \alpha_4 = 2$ and merge S_3 with σ_4 by appending a_7 to the end of b_4, as shown in Fig. 9a. Go to **Step 5**.

Step 5. Call **Checking routine** with input b_5. Call again **Checking routine** with input a_5.

Checking routine. We have $\mu = b_5$, $\nu = b_6$, and $\tilde{\nu} = a_6$. Go to **Checking and shifting**.

Checking and shifting: Task b_5 completes (at 24) later than the start of b_6 (at 23). Shift J_6 such that the start of b_6 is exactly at 24, as shown in Fig. 9b. Call **Checking routine** with input b_6, and call again **Checking routine** with input a_6.

Checking routine. We have $\mu = b_6$, $\nu = b_7$, and $\tilde{\nu} = a_7$. Go to **Final checking**.

Final checking: Task b_6 completes (at 27) exactly at the start of b_7. Terminate the subroutine.

Checking routine. We have $\mu = a_6$, $\nu = b_4$, and $\tilde{\nu} = a_4$. Go to **Checking and shifting**.

Checking and shifting: Task a_6 completes (at 20) exactly at the start of b_4. Terminate the subroutine.

Checking routine. We have $\mu = a_5$, $\nu = b_2$, and $\tilde{\nu} = a_2$. Go to **Checking and shifting**.

Checking and shifting: Task a_5 completes (at 14) later than the start of b_2 (at 12). Shift J_2 such that the start of b_2 is exactly at 14, as shown in Fig. 9c. Call **Checking routine** with input b_2, and call again **Checking routine** with input a_2.

Checking routine. We have $\mu = b_2, \nu = b_3$, and $\tilde{\nu} = a_3$. Go to **Checking and shifting**.

Checking and shifting: Task b_2 completes (at 16) earlier than the start of b_3 (at 17). Terminate the subroutine.

Checking routine. We have $\mu = a_2, \nu = b_1$, and $\tilde{\nu} = a_1$. Go to **Final checking**.

Final checking: Task a_2 completes (at 7) exactly at the start of b_1. Terminate the subroutine.

Step 6. The obtained schedule is S_4. Since $I = 3 = k - 1$, we output the schedule S_4 (Fig. 9c) and **stop**.

Theorem 1. *Given a sequence $\pi = (o_1, \ldots, o_{2n})$, Algorithm Plausible-Task-Sequence either produces a feasible schedule attaining the minimum makespan or identifies the infeasibility of π in $O(n^2)$ time.*

Proof. Assume a feasible schedule S_k is produced by the algorithm. At the end of **Step 2**, we have the k subschedules, $\sigma_1, \ldots, \sigma_k$, each of which attains the minimum makespan with respect to its corresponding fundamental cluster. In the recursive procedure from **Step 3** to **Step 6**, S_k is obtained by tightly arranging all the k partial schedules, $\sigma_1, \ldots, \sigma_k$, one by one. In **Step 3**, we can easily merge S_I with σ_{I+1} without shifting any task of S_I because $b_{n'_I}$, the task by which a_{n_I+1} should be preceded, is known to be b_{n_I}. In case of **Step 4(a)**, σ_{I+1} can also be greedily embedded in S_I without shifting any task of S_I because $\beta_I^1 \geqslant \alpha_{I+1}$ and $\beta_I^2 \leqslant \gamma_{I+1}$. Notice that in **Steps 4(b)–(d)**, it is required to defer the processing of $J_{n'_I+1}$, but all jobs other than $J_{n'_I+1}$ are not yet shifted. **Step 5** involves calling **Checking routine**s with the two tasks $b_{n'_I+1}$ and $a_{n'_I+1}$, respectively. Whenever **Checking routine** is invoked, either **Final checking** or **Checking and shifting** will

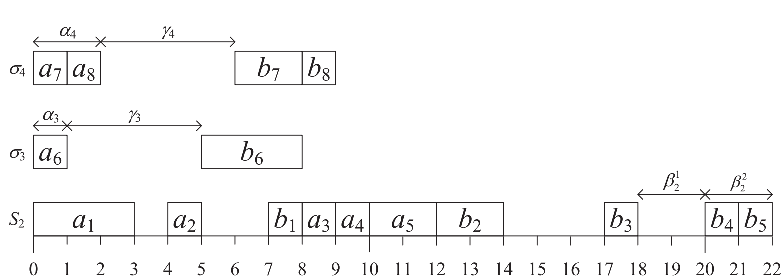

Fig. 7. Subschedules S_2, σ_3 and σ_4.

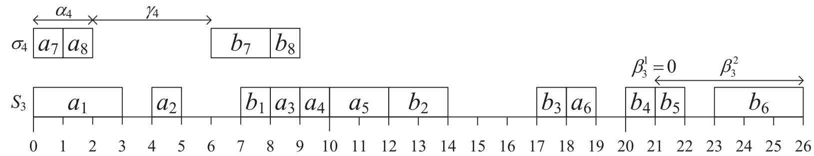

Fig. 8. Subschedules S_3 and σ_4.

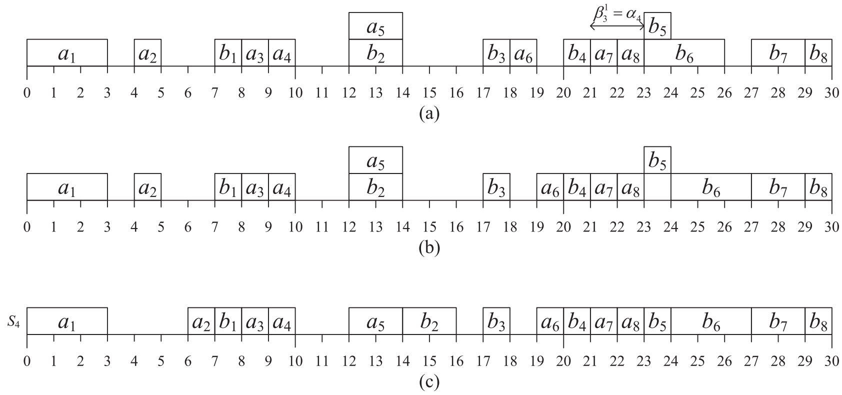

Fig. 9. The step by step construction of optimal schedule S_4.

be executed. Subroutine **Final checking** indicates that the infeasible result can be concluded whenever task b_1 (also, a_1), a_{n_l+1} or b_{n_l+1} needs to be shifted. In subroutine **Checking and shifting**, we examine whether any job needs to be shifted due to the shifting of the task passed to **Checking routine**. If the shifting of other tasks are made, then **Checking routine** will be called again. Provided that a feasible schedule S_k is obtained after the recursive procedure, either the first or second task of the job J_j in S_k tightly adjoins the task preceding it in π, for each $j = 2, \ldots, n$. No room is possible to further condense S_k.

Consider the case of infeasibility. If the infeasibility of π arises from **Step 2**, then it is due to the results of Property 1. If infeasible comes from the subroutine **Final checking**, then some task completes later than the start of b_1, a_{n_l+1} or b_{n_l+1}. It is obvious that shifting J_1 is futile and shifting J_{n_l+1} causes an infinite shifting recursion. Therefore, sequence π is infeasible if infeasibility is reported by the algorithm.

As for the running time of the algorithm, **Step 1** requires $O(n)$ time. **Step 2** takes at most $O\left(\left|\widetilde{X}_1\right| + \left|\widetilde{X}_2\right| + \cdots + \left|\widetilde{X}_k\right|\right) = O(n)$ time, and the recursion from **Step 3** to **Step 6** involves assembling k partial schedules, each of which takes no more than $O(n)$ time for the checking processes in **Step 5**. Since $k \leqslant n$, the overall running time is $O(n^2)$. □

4. Polynomially solvable cases

This section discusses three polynomially solvable cases for the fixed-job-sequence problem. Notice that the complexity result in this section is presented subject to the assumption of input size such that, for example, in the case of identical jobs, we have n copies of processing times and delay times for the n jobs (Orman & Potts, 1997).

4.1. $1|(p_j, p_j, p_j), fjs|C_{max}$

Consider the case where $a_j = l_j = b_j = p_j$ for all $j \in \mathbb{N}_n$. Despite the strong NP-hardness of the $1|(p_j, p_j, p_j)|C_{\max}$ problem (Orman & Potts, 1997), its corresponding fixed-job-sequence version is polynomially solvable. An optimal schedule can be obtained by the following procedure:

Algorithm PSC1

Step 1. Set $j = 1$.
Step 2. If $p_j = p_{j+1}$, then go to **Step 4**. Otherwise, append J_{j+1} to the end of J_j, and set $j = j + 1$.
Step 3. If $j = n$, then output the schedule and **stop**. Otherwise, go to **Step 2**.
Step 4. Interleave J_j and J_{j+1}. Append J_{j+2} to the end of J_{j+1}. Set $j = j + 2$. Go to **Step 3**.

Theorem 2. *The $1|(p_j, p_j, p_j), fjs|C_{max}$ problem can be solved in $O(n)$ by Algorithm PSC1. The makespan of the optimal schedule is $2\sum_{j\in E} p_j + 3\sum_{j\in \mathbb{N}_n\setminus E} p_j$, where E is the set of jobs interleaving with each other.*

Proof. It is obvious that no interleaving is possible for any two jobs other than two adjacent identical jobs, J_j and J_{j+1} with $p_j = p_{j+1}$. By examining each pair of adjacent jobs, **Algorithm PSC1** matches any un-interleaved J_j with J_{j+1} if $p_j = p_{j+1}$. Since no more interleaving is possible, **Algorithm PSC1** produces an optimal schedule. In the obtained optimal schedule, each interleaved pair of jobs, J_j and J_{j+1} for $j, j + 1 \in E$, contributes $2(p_j + p_{j+1})$ to the makespan. Any job J_h that cannot be interleaved contributes $3p_h$ to the makespan. Thus, $2\sum_{j\in E} p_j + 3\sum_{j\in \mathbb{N}_n\setminus E} p_j$. From **Step 2** to **Step 4**, at most n iterations are required, each of which takes a constant time. The overall running time of **Algorithm PSC1** is $O(n)$. □

4.2. $1|(p, p, b_j), fjs\ |C_{max}$

Without the assumption of a fixed-job-sequence, this special case can be solved in $O(n)$ time (Orman & Potts, 1997). Since $a_j = l_j = p$ for $j \in \mathbb{N}_n$, any job cannot be interleaved with more than one job. Subject to a given job sequence, we have the following property of this special case.

Property 2. *If the interleaving of jobs exists in a feasible schedule for problem $1|(p, p, b_j, fjs|C_{max}$, then the interleaved pair are some two consecutive jobs J_j and J_{j+1}, where $b_j \leqslant p,\ j \in \mathbb{N}_{n-1}$.*

With Property 2, a forward dynamic program can be designed. A job is called isolated if it is not interleaved with any other job. A subschedule of $\{J_1, J_2, \ldots, J_j\}$ can be completely characterized by the 2-tuples (j,λ), where j and λ are the number of jobs in the sub-

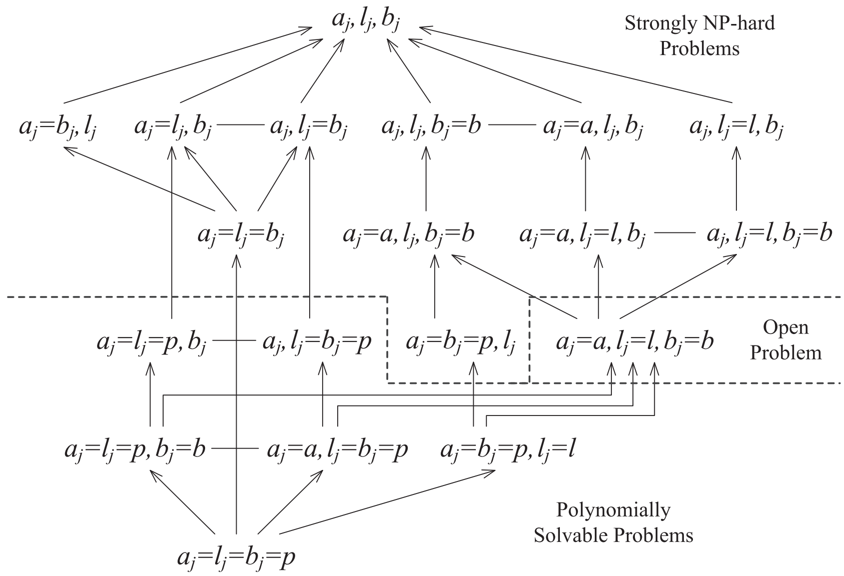

Fig. 10. The complexity graph of the prototypical problems (Orman & Potts, 1997).

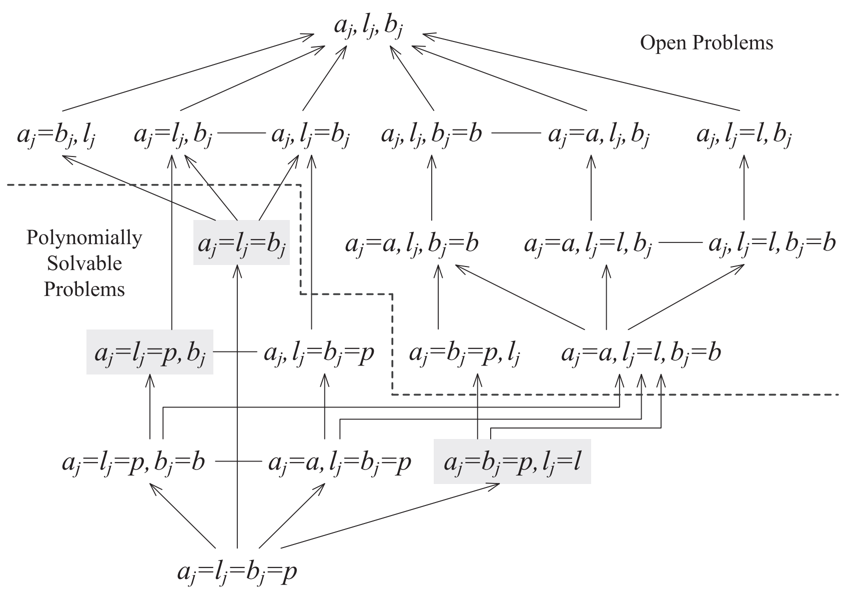

Fig. 11. The complexity graph of the fixed-job-sequence problems.

schedule and the interleaving status of job J_j, respectively. If $\lambda = 0$, then job J_j is isolated. If $\lambda = 1$, then job J_j is interleaved with job J_{j-1}. Denote the corresponding minimum makespan as $f(j, \lambda)$ for $1 \leqslant j \leqslant n$ and $\lambda \in \{0, 1\}$.

Algorithm PSC2

Initialization: $f(1, 0) = 2p + b_1$ and $f(1, 1) = \infty$.
Recursive function: For $2 \leqslant j \leqslant n$,

$$f(j,0) = \min\{f(j-1,0), f(j-1,1)\} + 2p + b_j. \tag{3}$$

$$f(j,1) = \begin{cases} f(j-1,0) + p + b_j - b_{j-1}, & b_{j-1} \leqslant p; \\ \infty, & \text{otherwise}. \end{cases} \tag{4}$$

Goal: $\min_{\lambda \in \{0,1\}} f(n, \lambda)$.

Theorem 3. *An optimal schedule for the $1|(p, p, b_j), fjs \mid C_{max}$ problem can be produced in $O(n)$ by Algorithm PSC2.*

Proof. Eq. (3) indicates that any isolated job J_j adjoins J_{j-1} which is either isolated or interleaved with J_{j-2}. In Eq. (4), job J_j can be interleaved with job J_{j-1} if J_{j-1} is isolated and $b_{j-1} \leqslant p$. A subschedule in the state (j, λ) with value $f(j, \lambda)$ dominates all other subschedules in the same state in the sense that it contributes the minimum value to the makespan among those of all subschedules in this state. The principle of optimality holds and **Algorithm PSC2** can generate an optimal schedule. To obtain $\min_\lambda f(n, \lambda)$, at most $n - 1$ iterations are required, each of which takes a constant time. The overall running time of **Algorithm PSC2** is $O(n)$. □

Corollary 1. *The $1|(a_j, p, P), fjs \mid C_{max}$, problem is solvable in $O(n)$.*

Proof. Orman & Potts (1997) proved that the coupled-task makespan problem and its reverse are equivalent. Given the fixed-job-sequence constraint, the equivalence still holds. By virtue of Lemma 3, this corollary follows. □

4.3. $1|(p, l, p)$, fjs | C_{max}

Since all jobs are identical, any feasible schedule for $1|(p, l, p)|C_{\max}$ satisfies the fixed-job-sequence constraint. By the results of Orman & Potts (1997) for problem $1|p, l, p,)|C_{\max}$, the fixed-job-sequence problem $1|(p, l, p)$, fis $|C_{\max}$ can be solved in $O(n)$.

By virtue of these three polynomially solvable cases, we can put the borderline between polynomially solvable problems and open problems in the complexity graph. In correspondence with the complexity graph of the coupled-task scheduling problems shown in Fig. 10, that of the fixed-job-sequence problems is given in Fig. 11. The strongly NP-hard problem $1|(p_j, p_j, p_j) \mid C_{\max}$, becomes polynomially solvable when the fixed-job-sequence assumption is imposed. For each polynomially solvable case of the prototypical problem, the corresponding fixed-job-sequence problem is also solvable in $O(n)$ time. However, it cannot be concluded that a fixed-job-sequence problem is easier to deal with than its counterpart problem without the fixed-job-sequence assumption

5. Concluding remarks

In this paper, we studied a single machine coupled-task makespan minimization problem subject to a fixed-job-sequence. To schedule a given task sequence abiding by the fixed-job-sequence constraint, we designed an $O(n^2)$ algorithm determine its feasibility and a schedule with the minimum makespan, if such a feasible schedule exists. Three polynomially solvable cases for the fixed-job-sequence problem were discussed. We also presented a complexity graph to depict the complexity statuses of the studied cases.

Although the complexity status of $1|(a_j, l_j, b_j)$, fjs | $C_{\max}$ remains open, the results presented in this study could inspire further research attention on this subject. It is also interesting to investigate the complexity status of the open problems indicated in the complexity graph of the fixed-job-sequence problems. Further research could also be conducted in developing branch-and-bound procedures in which our proposed algorithm for plausible task sequences could be exploited. In addition, different fixed-sequence constraint, e.g. given a fixed task-1 sequence or a fixed task-2 sequence, can be considered.

References

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

Ageev, A. A., & Kononov, A. V. (2006). Approximation algorithms for scheduling problems with exact delays. *Lecture Notes in Computer Sciences, 4368*, 1–14.

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

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

Békési, J., Galambos, G., Oswald, M., & Reinelt, G. (2009). Improved analysis of an algorithm for the coupled task problem with UET jobs. *Operations Research Letters, 37*, 93–96.

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

Cheng, T. C. E., Lin, B. M. T., & Toker, A. (2000). Makespan minimization in the two-machine flowshop batch scheduling problem. *Naval Research Logistics, 47*, 128–144.

Davis, T. (2006). *Catalan number*. <http://www.geometer.org/mathcircles>.

Hwang, F. J., Kovalyov, M. Y., & Lin, B. M. T. (2010). Minimization of total completion time in flowshop scheduling subject to fixed job sequences. In *Proceedings of 12th international workshop on project management and scheduling, Tours, France* (pp. 249–252).

Li, H., & Zhao, H. (2007). Scheduling coupled-tasks on a single machine. In *Proceedings of 2007 IEEE symposium on computational intelligence in scheduling, Honolulu, Hawaii* (pp. 137–142).

Lin, B. M. T., & Hwang, F. J. (2011). Total completion time minimization in a 2-stage differentiation flowshop with fixed sequences per job type. *Information Processing Letters, 111*, 208–212.

Ng, C. T., & Kovalyov, M. Y. (2007). Batching and scheduling in a multi-machine flow shop. *Journal of Scheduling, 10*, 353–364.

Orman, A. J., & Potts, C. N. (1997). On the complexity of coupled-task scheduling. *Discrete Applied Mathematics, 72*, 141–154.

Shafransky, Y. M., & Strusevich, V. A. (1998). The open shop scheduling problem with a given sequence of jobs on one machine. *Naval Research Logistics, 45*, 705–731.

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

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

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