IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received May 3, 2020, accepted May 7, 2020, date of publication May 11, 2020, date of current version May 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993857

Dynamic Programming Algorithms for Two-Machine Hybrid Flow-Shop Scheduling With a Given Job Sequence and Deadline

QI WEI AND YONG WU
Department of Logistics Management, Ningbo University of Finance and Economics, Ningbo 315175, China
Corresponding author: Qi Wei (weiqi_0574@163.com)

This work was supported in part by the Zhejiang Provincial Philosophy and Social Science Program of China under Grant 19NDJC093YB, in part by the National Social Science Foundation of China under Grant 19BGL001, in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LY19A010002, and in part by the Ningbo Natural Science Foundation of China under Grant 2019A610048 and Grant 2019A610095.

ABSTRACT In "Shared Manufacturing" environment, orders are processed in a given job sequence which is based on the time of receipt of orders. This paper studies a problem of scheduling two-task jobs in a two-machine hybrid flow-shop subject to a given job sequence which is used in production of electronic circuits under shared manufacturing. Each job has two tasks: the first one is a flexible task, which can be processed on either of the two machines, and the second one is a preassigned task, which can only be processed on the second machine after the first task is finished. Each job has a processing deadline. Three objective functions related to deadlines are considered. The computational complexity of the problem for any of three objective functions is showed to be ordinary NP-hard, a dynamic programming algorithm (DPA) is presented for each case and the time complexity of each algorithm is given. The results of computational experiments show the relationship between the running time of DPA and the parameters, and also show the advantages of DPA in dealing with this problem compared with branch-and-bound algorithm and iterated greedy algorithm.

INDEX TERMS Shared manufacturing, hybrid flow shop, dynamic programming algorithm, computational complexity.

I. INTRODUCTION

Two-machine hybrid flow-shop problem is a sort of scheduling problem which is widely used in CNC machining, production of electronic circuits [1], computer graphics processing [2], [3] and the health care systems [4]. For example, the production of electronic circuit usually needs two procedures. The first procedure usually requires low precision and can be processed by low-level machine. And the second procedure requires high-level machines for more precise processing. These high-level machines often utilize detachable tool magazines that allow for off-line setups. If necessary, these high-level machines can process the first procedure by replacing low-precision tools. I.e., the first procedure can be processed by any one of low-level machine and high-level machine with the same processing time but the second procedure only can be processed by high-level machine. Once the machine starts to process the electronic circuit, it is not allowed to interrupt, otherwise the products will be scrapped. So pre-emption is not allowed in this production scenario.

In "Shared Manufacturing" environment, manufacturing platforms arrange order processing sequence based on the importance of the customers or the time of receipt of orders, i.e., the processing order is given in advance according to some principle. So, in shared manufacturing environment, the electronic circuit manufacturing problem can be described as a two-machine hybrid flow-shop scheduling with a given job sequence.

This two-machine hybrid flow-shop problem can be described as follows. A set of n jobs $J = \{J_1, J_2, \cdots, J_n\}$ is processed in a two-stage two-machine flow-shop which consists of Machine M_1 at stage 1 and Machine M_2 at stage 2. Each job J_i has two tasks A_i and B_i. The first task A_i is a

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Zakarya.

flexible task, which can be processed on any of Machine M_1 and Machine M_2 for the same processing time a_i; the second task B_i is a preassigned task, which can only be processed on Machine M_2 for b_i time units and must be processed after A_i is finished. Each job J_i has a deadline d_i and needs to be completed as soon as possible before its deadline. If some jobs miss their deadlines, the objective function value related to the deadline will increase, and this is what we should avoid as far as possible. All tasks and machines are available at time 0. Pre-emption is not allowed, i.e., once a job starts being processed on a machine, it has to be finished before any other job can be processed on that machine. All tasks are processed in the order of subscription (given job sequence constraint), i.e., if task A_i and A_j are both arranged to be processed on machine M_1 and satisfy $i < j$, the task A_i is processed before A_j (The processing sequence of tasks on Machine M_2 also meets this requirement). Obviously, all jobs will be completed in the order of subscription from 1 to n on Machine M_2. The goal of the problem is to minimize one of the following three objective functions: the maximum lateness ($L_{\max}$), the total weighted tardiness ($\sum w_iT_i$) or the weighted number of tardy jobs ($\sum w_iU_i$).

In this paper, we consider a two-stage two-machine hybrid flow-shop problem with a given job sequence which is applied to the production of electronic circuit in shared manufacturing environment. For three objectives of this problem, the computational complexity is analysed and the pseudo-polynomial time dynamic programming algorithm (denoted as DPA) is designed respectively. According to the three-field representation, the three problems discussed in this paper can be expressed as:

$$(1)\ FS_2\,|FJS, Hybrid|\,L_{\max}$$
$$(2)\ FS_2\,|FJS, Hybrid|\sum w_iT_i$$
$$(3)\ FS_2\,|FJS, Hybrid|\sum w_iU_i$$

The problem (1) is a hybrid Flow-Shop problem with Fixed job sequence whose objective is minimize the maximum Lateness. So it is denoted as FSFL. Similarly, the problem (2) and (3) are denoted as FSFT and FSFU in the following discussion. The rest of this paper is arranged as follows: Section II presents a brief review of the literature. In Section III, we give the basic symbolic hypothesis and the characteristics of the optimal solution of the problem, then prove that the three problems are all NP-hard in ordinary sense. In Section IV, we present the DPA for problem FSFL. In Section V we give the DPAs for problem FSFT and FSFU. Computational experiments are carried out and the results are analyzed in Section VI. Finally, we conclude the paper and suggest future research topics in Section VII.

II. LITERATURE REVIEW

If the presumption of a given job sequence is not considered and the objective is to minimize the maximum completion time (makespan), the problem was first proposed by Wei and He [3] in 2005. They called it Semi-Hybrid Flow-Shop problem (denoted as SHFS). They showed that the problem is NP-hard, and gave a pseudo-polynomial time algorithm and a polynomial time approximation algorithm with a worst-case ratio of 2. Then Wei and Jiang [5] gave an improved polynomial time approximation algorithm with the worst-case ratio of 8/5. Lately, Wei *et al.* [6] presented constant-time solution algorithms for the cases with identical jobs and analysed the relationship between the hybrid benefits and performance difference between the two machines. It is obvious that the researches on the makespan objective of this problem have been more in-depth, but the researches on other objective functions have not been reported yet.

Other typical models for two-stage hybrid flow shop problems include the following: Vairaktarakis and Lee [1] discussed the problem that two tasks both can be processed on any machine, and gave an approximate algorithm with the worst-case ratio of 1.618; Tan *et al.* [7] considered a flexible flowshop scheduling problem with batch processing machines at each stage, and gave an iterative stage-based decomposition approach to solve this problem; Feng *et al.* [8] studied a two-stage hybrid flowshop with uncertain processing time and gave a heuristic algorithm for their problem; Ahonen and Alvarenga [9] proposed a new two-stage hybrid flow shop problem where the job's processing time is related to the starting time of the job, and used annealing algorithm and tabu search method to solve their problem; Hidri *et al.* [10] addressed a two machines hybrid flow shop scheduling problem with transportation times between two machines, and presented a heuristic based on the optimal solution of the parallel machine scheduling problem with release date and delivery time; Zhang *et al.* [11] considered a hybrid flowshop problem with four batching machines, and used the clustering and genetic algorithm to calculate the good solution for this problem. For multi-stage hybrid flow shop problems, recently, Jiang and Zhang [12] investigated an energy-oriented scheduling problem deriving from the hybrid flow shop with limited buffers. They developed an efficient multi-objective optimization algorithm under the framework of the multi-objective evolutionary algorithm based on decomposition. However, none of the above problems is considered the presumption of a given job sequence.

In recent years, with the rise of intelligent manufacturing modes such as shared manufacturing and cloud manufacturing, the researches with the presumption of a given job sequence became very meaningful. Another important industrial application of the given job sequence setting is the scheduling of bar-coding operations in inventory or stock control systems [13]. The earliest scheduling problem with a given job sequence constraint was proposed by Shafransky and Strusevich [14]. They studied an open shop problem to minimize makespan with a given job sequence, proved the problem is strongly NP-hard and gave an approximate algorithm with a worst-case ratio of 5/4. Afterwards, Liaw *et al.* [15] studied the same problem, but the objective function changed to minimize the total completion time. They

TABLE 1. Notation used in the paper.

Notation	Description
a_i	The processing time of the first task A_i of job J_i on M_1 or M_2 ;
b_i	The processing time of the second task B_i of job J_i on M_2 ;
C_i	The complete time of the job J_i which is equal to the complete time of B_i ;
w_i	The weight of the job J_i ;
d_i	The processing deadline of the job J_i ;
T_i	The tardiness of the job J_i which is equal to $\max\{0, \quad C_i - d_i\}$;
U_i	Represents whether the job J_i is delayed which is equal to 1 for $C_i - d_i > 0$ and 0 for $C_i - d_i \le 0$;
$L_{\max}$	The maximum lateness of all jobs which is equal to $\max_{1\le i\le n}\{T_i\}$;
V_1	The job subset $\{j_i \mid A_i \text{ is processed on } M_1\}$;
V_2	The job subset $\{j_i \mid A_i \text{ is processed on } M_2\}$.

presented a heuristic and a branch-and-bound algorithm for this problem. Lässig *et al.* [16] introduced the constraint of given job sequence into the common due-date scheduling problem, and presented a linear algorithm for this problem. Cheref *et al.* [17] considered an integrated production and outbound delivery scheduling problem with a given job sequence, showed this problem is NP-hard, and gave polynomial time algorithms for some particular cases. Lately, Cheng *et al.* [18] considered server scheduling on parallel dedicated machines with fixed job sequences to minimize the makespan. They designed a polynomial time algorithm to solve the two machine case of the problem and proved the problem is strongly NP-hard when the number of machines is arbitrary. They also designed two heuristic algorithms to treat the case where the number of machines is arbitrary and all the loading times are unit.

As can be seen from the above, two-machine hybrid flow-shop scheduling with a given job sequence and deadline has not been investigated with any exact or heuristic method in the literature so far. Hence, the DPA presented in this article provides a feasible method to solve this problem. The computational experiments show that DPA has obvious advantages in running time compared with branch-and-bound algorithm and has more than 30% advantages in the accuracy of calculation results compared with iterated greedy algorithm.

III. SYMBOLIC HYPOTHESIS, STRUCTURE OF SOLUTIONS AND COMPUTATIONAL COMPLEXITY

In this section, firstly we give the basic symbols needed in the following sections. And then we analyse the properties of the optimal solution of SHFS with a given job sequence. Finally, we show the problems studied in this paper are all NP-hard.

A. NOTATION

The notations used in the rest of this paper are listed in Table 1.

B. THE STRUCTURAL CHARACTERISTICS OF THE OPTIMAL SCHEDULE OF THE PROBLEMS

An optimal schedule is a scheduling scheme to minimize the objective function of the problem. Whether the objective function is minimizing the maximum lateness, the total weighted tardiness or the weighted number of tardy jobs, it is easy to get that there is an optimal schedule of this problem satisfying the following properties.

Proposition 3.1: There is an optimal schedule where machine M_1 does not have idle time from time 0 to the end of processing.

Proof: Suppose that there is an optimal schedule ϕ where machine M_1 has idle time between some successive processed tasks. Using schedule ϕ, we construct another schedule φ: All tasks on machine M_1 are processed as early as possible to fill all the idle time in the same order as that in schedule ϕ; the tasks on machine M_2 are processed in the same way as that in schedule ϕ. For the jobs in V_1, the first tasks in φ on machine M_1 are finished no later than them in ϕ, and the second tasks on machine M_2 in φ and ϕ start at the same time. So the second task of each job in V_1 is processed after the first task of this job is finished in φ. We have φ is feasible. For the tasks on machine M_2 are processed in the same way as that in schedule ϕ, the complete time of each job in φ is the same as that in ϕ. So the maximum lateness, the total weighted tardiness and the weighted number of tardy jobs in φ are all the same as those in ϕ. Since ϕ is optimal, φ is also optimal and there is on idle on machine M_1 in schedule φ. So Proposition 3.1 holds.

Proposition 3.2: There exists an optimal schedule which satisfies that the idle time on machine M_2 appears only before the second tasks of some jobs in V_1, but can't appear elsewhere.

Proof: Suppose that there is an optimal schedule ϕ where machine M_2 has idle time before the tasks of some jobs in V_2. Using the idea similar to the proof of Proposition 3.1, we construct a schedule φ: The tasks on machine M_1 and the tasks of the jobs in V_1 on machine M_2 are all processed in the same way as those in schedule ϕ; the tasks of the jobs in V_2 on machine M_2 are processed as early as possible to fill all the idle time before them in the same order as that in schedule ϕ. Since the jobs in V_1 are processed the same as that in ϕ, the second task of each job in V_1 starts to be processed after the first task of this job is finished. So φ is feasible. In V_1, the completion time of each job in φ is the same as that in ϕ. In V_2, the completion time of each job in φ is less than or equal to that in ϕ. So the maximum lateness, the total weighted tardiness and the weighted number of tardy jobs in φ are not more than those in ϕ. Since ϕ is optimal, φ is also optimal and there is on idle before the tasks of the jobs in V_2 on machine M_2 in φ. So Proposition 3.2 holds.

Proposition 3.3: There exists an optimal schedule which satisfies that the task A_1 of first job J_1 is processed on Machine M_2, and the task A_n of the last job J_n is processed on Machine M_1.

Proof: Suppose that there is an optimal schedule ϕ where the task A_1 of first job J_1 is processed on Machine M_1. Using schedule ϕ, we construct another schedule φ. Firstly, change the processing mode of job J_1, i.e., A_1 and B_1 are all processed together on Machine M_2. In ϕ, considering that B_1 can't be processed until A_1 is completed and B_1 is the first task to be processed on Machine M_2, there is an idle with length a_1 before task B_1 is processed on Machine M_2. So, in φ, we can process A_1 on this idle before task B_1 on Machine M_2. It is easy to see that the complete time of J_1 has not changed in φ. Then we can processed the other jobs in the same way and at the same time as those in schedule ϕ. Obviously, the complete time of all jobs is the same in ϕ and φ, so the maximum lateness, the total weighted tardiness and the weighted number of tardy jobs in φ are the same as those in ϕ. Since ϕ is optimal, φ is also optimal. This establishes the first part of the proposition. Using the similar way, we can get the second part of the proposition.

According to Property 3.1, Property 3.2 and Property 3.3, it can be concluded that there must be an optimal schedule of the problem as shown in the following figure:

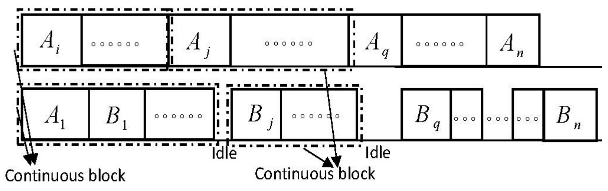

FIGURE 1. The structure of optimal schedule of this problem.

There must be an optimal schedule where the task A_1 is processed on Machine M_2, the task A_n is processed on Machine M_1 and the continuous processing tasks are separated from the idle time before the second tasks of some jobs in V_1 on machine M_2. The continuous processing tasks between two idle time periods do not contain any idle time which are called "Continuous Blocks". In Section IV and Section V, the continuous blocks will help us design DPAs for the problems studied in this paper. In the following, we only need to find the optimal solution in the feasible solutions which meet the above three properties.

C. COMPUTATIONAL COMPLEXITY ANALYSIS

Now, we use polynomial time Turing Reduction to prove that FSFL, FSFT and FSFU are all NP-hard.

Theorem 3.4: Problem FSFL is NP-hard.

Proof: Firstly, we present an instance of Partition Problem which is a known NP-hard problem. We denote this instance as Instance I: Let set of integers $S = \{s_1, s_2, \cdots s_n\}$ and an integer bound $s = {}^1\!/_2 \sum_{i=1}^{n} s_i$. Is there a partition S_1 and S_2 of set S, such that $S = S_1 \cup S_2, S_1 \cap S_2 = \emptyset$ and $\sum_{s_i \in S_1} s_i = \sum_{s_i \in S_2} s_i = s$?

We create an instance of FSFL with $n+2$ jobs denoted as Instance II: Let a job set $V = \{J_0, J_1, J_2, \cdots J_n, J_{n+1}\}$ where

$$\begin{aligned} &\text{job } J_0 : a_0 = s - \varepsilon, \quad b_0 = \varepsilon, \\ &\text{job } J_i : a_i = s_i, \quad b_i = \varepsilon, 1 \le i \le n, \\ &\text{job } J_{n+1} : a_{n+1} = s + n\varepsilon, \quad b_0 = 0. \end{aligned}$$

Let the deadline of job $J_i d_i = 0$ $(i = 0, 1, \cdot \cdot \cdot, n, n+1)$. If all jobs are processed in the order of subscription, is there a feasible schedule that makes the maximum lateness $L_{\max} = 2s + n\varepsilon$?

Next, we prove that the solutions of Instance I and Instance II can be derived from each other. Let S_1 and S_2 be a partition of S in Instance I. A feasible schedule is constructed as follow. Let $V_1 = \{J_i \,|s_i \in S_1\} \cup \{J_{n+1}\}$ and $V_2 = \{J_0\} \cup \{J_j \,|s_j \in S_2\}$, all tasks be processed in the order of their subscription. Obviously, A_i is processed on machine M_1 if $J_i \in V_1$ and A_j is processed on machine M_2 if $J_j \in V_2$. So we have the maximum lateness

$$\begin{aligned} L_{\max} &= \max \left\{ \sum\nolimits_{J_i \in V_1} a_i, \ \sum\nolimits_{J_j \in V_2} a_j + \sum_{t=0}^{n+1} b_t \right\} \\ &= \sum\nolimits_{s_i \in S_1} s_i + (s + n\varepsilon) \\ &= (s - \varepsilon) + \sum\nolimits_{s_j \in S_2} s_j + (n+1)\,\varepsilon = 2s + n\varepsilon. \end{aligned}$$

(V_1, V_2) is a solution of II as depicted in Fig. 2.

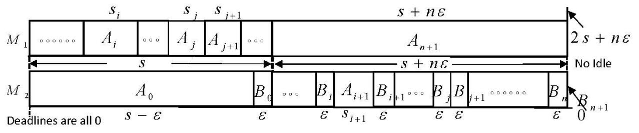

FIGURE 2. Configuration of a feasible schedule with maximum lateness of $2s + n\varepsilon$.

Assume now that there is a feasible schedule the maximum lateness of which is exactly $2s+n\varepsilon$. Since deadlines of all jobs are 0, the maximum lateness of this feasible schedule is equal to its makespan. So the makespan of this feasible schedule is also $2s + n\varepsilon$. Since the sum of the processing loads of all jobs is $4s + 2n\varepsilon$, this feasible schedule is also an optimal schedule and no idle time is allowed on either machine. By Proposition 3.3, without loss of generality, we let ϕ be an optimal schedule of Instance II with maximum lateness $L_{\max} = 2s + n\varepsilon$ where task A_0 of the first job J_0 is processed on Machine M_2 and the task A_{n+1} of the last job J_{n+1} is processed on Machine M_1. We let V_1 be the job subset $\{j_i \ |A_i \ is \ processed \ on \ M_1 \ in\phi\}$ and V_2 be the job subset $\{j_i \ |A_i \ is \ processed \ on \ M_2 in\phi\}$. According the sum of the processing loads of all jobs is $4s + 2n\varepsilon$ and the maximum lateness is $2s + n\varepsilon$, we have the loads of two machines are all $2s + n\varepsilon$, i.e., the load of Machine M_1 $\sum_{J_i \in V_1} a_i = 2s + n\varepsilon$ and the load of Machine M_2 $\sum_{J_j \in V_2} a_j + \sum_{t=0}^{n+1} b_t = 2s + n\varepsilon$. Since $J_{n+1} \in V_1$ and $a_{n+1} = s + n\varepsilon$, $\sum_{J_i \in V_1 - \{J_{n+1}\}} a_i = (2s + n\varepsilon) - (s + n\varepsilon) = s$. And since $J_0 \in V_2$, $a_0 = s - \varepsilon$, $b_i = \varepsilon (0 \leq i \leq n)$ and $b_{n+1} = 0$, $\sum_{J_j \in V_2 - \{J_0\}} a_j = (2s + n\varepsilon) - (s - \varepsilon) - (n+1)\varepsilon = s$. Now let $S_1 = \{s_i \,|J_i \in V_1 - \{J_{n+1}\}\}$ and $S_2 = \left\{s_j \,\middle|J_j \in V_2 - \{J_0\}\right\}$. Obviously, we have $\sum_{s_i \in S_1} s_i = \sum_{J_i \in V_1 - \{J_{n+1}\}} a_i = s$, $\sum_{s_i \in S_2} s_i = \sum_{J_j \in V_2 - \{J_0\}} a_j = s$ and $S = S_1 \cup S_2$, $S_1 \cap S_2 = \emptyset$. So we have (S_1, S_2) is a solution of Instance I.

Since Partition Problem is a known NP-hard problem and reduction process is polynomial time. We have problem FSFL is NP-hard.

Corollary 3.5: Problem FSFL is NP-hard even if $b_i = 0, i = 1, 2, \cdots, n$.

Only need to let $\varepsilon = 0$ in the proof of Theorem 3.4, we have problem FSFL is also NP-hard even if $b_i = 0$, $i = 1, 2, \cdots, n$.

Theorem 3.6: Problem FSFT and FSFU are all NP-hard.

Proof: We create an instance of FSFT denoted as Instance III and an instance of FSFU denoted as Instance IV from Instance I.

Instance III: The assumption of the job set $V = \{J_0, J_1, J_2, \cdots J_n, J_{n+1}\}$ is the same as that in the proof of Theorem 3.4. But let the deadline $d_i = 2s+n\varepsilon$ and the weight $w_i = 1$ for $i = 0, 1, 2, \cdots, n, n+1$. If all jobs are processed in the order of subscription, is there a schedule that makes the total weighted tardiness $\sum w_i T_i = 0$?

Instance IV: Replace $\sum w_i T_i = 0$ in the Instance III with $\sum w_i U_i = 0$.

By using techniques similar to the proof in Theorem 3.4, it can be proved that the solutions of Instance I and Instance III (Instance IV) can be derived from each other. So FSFT and FSFU are all NP-hard.

Since we will give pseudo-polynomial time algorithms for these problems in the following sections, FSFL, FSFT and FSFU are all NP-hard in ordinary sense.

IV. A DYNAMIC PROGRAMMING ALGORITHM FOR FSFL

According to the structure of the optimal schedule obtained by Property 3.1 and Property 3.2, we have the optimal schedule is composed of several continuous blocks and the idle time between them as shown in Fig. 1. So the DPA we designed includes two stages: the first stage is to construct the optimal continuous block, and the second stage is to arrange the optimal continuous blocks into the optimal schedule of the problem through idle time periods.

A. CONSTRUCT THE OPTIMAL CONTINUOUS BLOCK

Firstly, a strict definition of continuous block of FSFL is given as following.

Definition 4.1: A subschedule named five-element **Continuous Block** in state (m, i, j, h, l) as a subschedule for jobs $J_i, J_{i+1}, \cdots, J_j$ satisfying the following conditions (see Fig. 3):

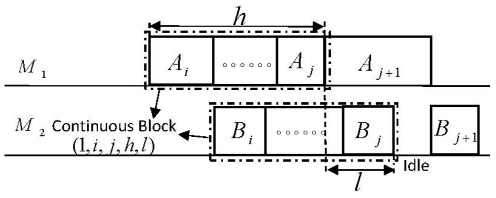

FIGURE 3. The structure of Continuous Block $(1, i, j, h, l)$.

(1) There is no idle time between any two consecutive tasks on both machine M_1 and M_2;
(2) The flexible task of the first job $J_i A_i$ is processed on machine M_m where $m \in \{1, 2\}$;
(3) The load of jobs $J_i, J_{i+1}, \cdots, J_j$ on machine M_1 is exactly h;
(4) The gap between the complete time of jobs $J_i, J_{i+1}, \cdots, J_j$ on machine M_1 and M_2 is exactly l.

Obviously, continuous block (m, i, j, h, l) can be constructed by continuous block $(m, i, j-1, h', l')$ in two ways (see Fig. 4).

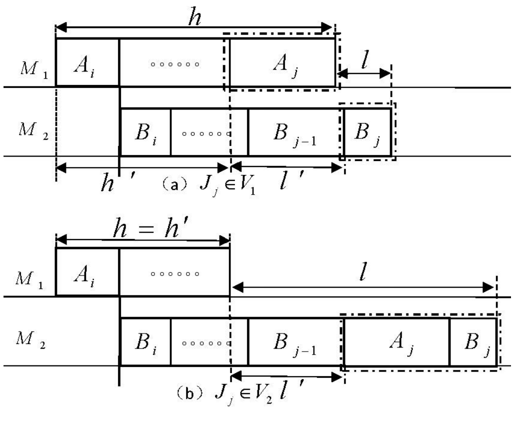

FIGURE 4. Processing diagram from $(1, i, j-1, h', l')$ to $(1, i, j, h, l)$.

One way is to obtain continuous block (m, i, j, h, l) by adding job $J_j \in V_1$ to continuous block $(m, i, j-1, h', l')$ (see Fig. 4a). The other way is to obtain continuous block (m, i, j, h, l) by adding job $J_j \in V_2$ to continuous block $(m, i, j-1, h', l')$ (see Fig. 4b). Let $f(m, i, j, h, l)$ be the maximum lateness of the optimal continuous block composed of the jobs $J_i, J_{i+1}, \cdots, J_j$. For ease of description, define the following function:

$$\sigma(m) = \begin{cases} 0, & if \ m = 1; \\ 1, & if \ m = 2; \end{cases} \quad m \in \{1, 2\}.$$

For $i < j$, according to the definition of continuous block (m, i, j, h, l), the following formula was clearly established:

$$a_i\sigma(m) + \sum_{t=i}^{j} b_t - \sum_{t=i+1}^{j} a_t \le l \tag{1}$$

$$l \le a_i\sigma(m) + \sum_{t=i}^{j} b_t + \sum_{t=i+1}^{j} a_t \tag{2}$$

$$l \ge b_j \tag{3}$$

Let $\underline{l} = \max\left\{b_j, a_i\sigma(m) + \sum_{t=i}^{j} b_t - \sum_{t=i+1}^{j} a_t\right\}$, $\bar{l} = a_i\sigma(m) + \sum_{t=i}^{j} b_t + \sum_{t=i+1}^{j} a_t$, we have the value interval of the gap l is $[\underline{l}, \bar{l}]$. The range of other parameters is obvious: $m \in \{1, 2\}$, $1 \le i < j \le n$ and $0 \le h \le \sum_{t=i}^{j} a_t$. Now we present the DPA of $f(m, i, j, h, l)$ as following.

DPA CB(L)

Initial conditions:

$$f(m, i, j, h, l) = \begin{cases} a_i + b_i - d_i, & \text{if } m = 1, i = j, h = a_i, l = b_i, \text{ or} \\ & m = 2, i = j, h = 0, l = a_i + b_i; \\ +\infty, & \text{otherwise.} \end{cases}$$

Recursions:

For each m, i, j, h, l satisfying $m \in \{1, 2\}$, $1 \le i < j \le n$, $0 \le h \le \sum_{t=i}^{j} a_t$, $\underline{l} \le l \le \bar{l}$,

Case 1: A_j is processed on M_1 as shown in Fig. 4a.

$$f_1 = \max\left\{f(m, i, j-1, h-a_j, l+a_j-b_j),\ h+l-d_j\right\} \tag{4}$$

Case 2: A_j is processed on M_2 as shown in Fig. 4b.

$$f_2 = \begin{cases} \max\left\{f(m, i, j-1, h, l-a_j-b_j),\ h+l-d_j\right\} \\ \qquad\qquad \text{if } l \ge a_j + b_j; \\ +\infty \qquad\qquad \text{otherwise.} \end{cases} \tag{5}$$

$f(m, i, j, h, l) = \min\left\{f_1, f_2\right\}$.

The initial conditions in DPA CB (L) are obviously valid. The recursions are analysed as following. For a feasible combination of m, i, j, h, l, the derivation of $f(m, i, j, h, l)$ can be given by considering two ways regarding the assignment of the flexible task of the last job J_j. In Case 1, task A_j is processed on machine M_1 as shown in Fig. 4a. We have $h' + a_j = h$ and $l' + b_j - a_j = l$. Thus, it can be shown that $h' = h - a_j$ and $l' = l + a_j - b_j$, as given in Equation (4), subject to the condition $l' \ge a_j$, i.e. $l \ge b_j$, which is satisfied anyway according to the value range of l. In Case 2, task A_j is processed on machine M_2 as shown in Fig. 4b. We have $h = h'$ and $l = l' + a_j + b_j$, i.e. $h' = h$ and $l' = l - a_j - b_j$, as given in Equation (5), subject to the condition $l' = l - a_j - b_j \ge 0$, i.e. $l \ge a_j + b_j$.

B. COMPLETE DYNAMIC PROGRAMMING ALGORITHM

After the continuous block construction, a complete schedule can be generated with a concatenation of appropriate optimal continuous blocks in backward recursion. According to Properties 3.1 and 3.2, we note that every two adjacent optimal continuous blocks are separated by an idle time period on machine M_2. Let's first define partial schedule set (m, i) which represents the set of all partial schedules of job subset $\{J_i, J_{i+1}, \cdots, J_n\}$ where the first job J_i is processed by machine M_m. Denote by $g(m, i)$ the minimum maximum lateness among all the partial schedules in set (m, i). A DPA for calculating $g(m, i)$ is given as following. For easy narration, we set up a dummy job J_{n+1} with $a_{n+1} = b_{n+1} = +\infty$ beforehand.

DPA Sch(L)

Initial conditions:

$$g(m, n+1) = -\infty, \qquad m = 1, 2;$$

Recursions (A_{j+1} can only be processed on M_1 as shown in Fig. 5):

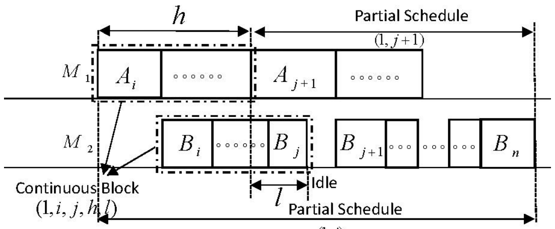

FIGURE 5. (m, i) is constructed from $(m, j+1)$ and (m, i, j, h, l).

For each m, i satisfying $m \in \{1, 2\}$, $1 \le i \le n$,

$$g = \begin{cases} \max\left\{f(m, i, j, h, l),\ h + g(1, j+1)\right\} \\ \qquad\qquad \text{if } l < a_{j+1}; \\ +\infty \qquad\qquad \text{otherwise.} \end{cases}$$

$$g(m, i) = \min_{\substack{i \le j \le n \\ 0 \le h \le \sum_{t=i}^{j} a_t \\ \underline{l} \le l \le \bar{l}}} \{g\}; \tag{6}$$

Goal: $\min L_{\max} = \min_{m \in \{1,2\}} \{g(m, 1)\}$.

The initial conditions and the goal are apparently true. The recursions are analysed as following. In recursions of **Sch(L)**, when the optimal continuous block (m, i, j, h, l) is given, partial schedule (m, i) can be structured by $(m, j+1)$ and (m, i, j, h, l). Since there is an idle time period between $(m, j+1)$ and (m, i, j, h, l), according to Property 3.2 we have $m = 1$ in $(m, j+1)$, i.e. task A_{j+1} is processed on M_1 (see Fig. 5). It is easy to see that $g(m, i) = \min\max\left\{f(m, i, j, h, l),\ h + g(1, j+1)\right\}$. Since there is an idle time period between job J_j and J_{j+1}, there must be $l < a_{j+1}$. So we have that Equation (6) holds.

Next, we will give the time complexity of DPA Sch(L).

Theorem 4.2: Problem FSFL is solvable in $O(n^2(\sum_{i=1}^{n} a_i)^2)$ time. So problem FSFL is NP-hard in ordinary sense.

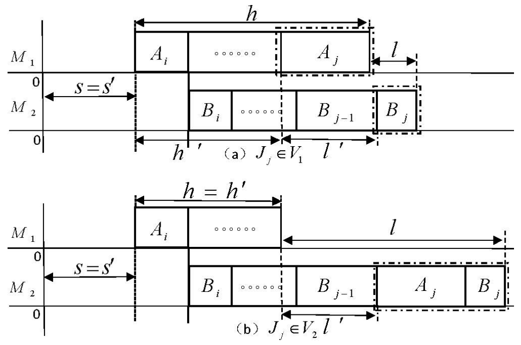

FIGURE 6. Processing diagram from $(1, i, j-1, h', l', s')$ to $(1, i, j, h, l, s)$.

Proof: To calculate the optimal continuous block, we need to search variable l from $\underline{l}$ to $\bar{l}$, variables i, j from 1 to n, variable h from 0 to $\sum_{i=1}^{n} a_i$ and variable m from 1 to 2. So we have that it takes $O(n^2(\sum_{i=1}^{n} a_i)^2)$ time to calculate all optimal continuous blocks. When all optimal continuous blocks are given, to calculate the optimal schedule, there are $O(n)$ states, each of which takes at most $O(n^2(\sum_{i=1}^{n} a_i))$ time due to the loops over all possible subscripts of the min operator. So the run time for calculating the optimal schedule is also $O(n^2(\sum_{i=1}^{n} a_i)^2)$. We have problem FSFL can be solved in $O(n^2(\sum_{i=1}^{n} a_i)^2)$ time. So problem FSFL is NP-hard in ordinary sense.

V. DYNAMIC PROGRAMMING ALGORITHMS FOR PROBLEM FSFT AND FSFU

Next, we will use techniques similar to Section IV to design DPAs for problem FSFT and FSFU. The tardiness of the jobs within the continuous blocks and partial schedules created via the procedures for the maximum lateness cannot be fathomed. We therefore instead determine the optimal objective value for the total weighted tardiness or the weighted number of tardy jobs within the continuous blocks and partial schedules subject to the condition that the first job starts at a specified time point. We introduce an extra parameter S in the continuous blocks and partial schedules, which is the start time of the continuous block or partial schedule, i.e. the interval from time 0 to the start time of the first task of the continuous block or partial schedule.

Definition 5.1: A six-element **Continuous Block** in state (m, i, j, h, l, s) as a subschedule for jobs $J_i, J_{i+1}, \cdots, J_j$ satisfying the following conditions (see Fig. 6):

(1) There is no idle time between any two consecutive tasks on both machine M_1 and M_2;
(2) The flexible task of the first job $J_i A_i$ is processed on machine M_m where $m \in \{1, 2\}$;
(3) The load of jobs $J_i, J_{i+1}, \cdots, J_j$ on machine M_1 is exactly h;
(4) The gap between the complete time of jobs $J_i, J_{i+1}, \cdots, J_j$ on machine M_1 and M_2 is exactly l.
(5) The start time of job J_i is exactly s.

One way is to obtain continuous block (m, i, j, h, l, s) by adding job $J_j \in V_1$ to continuous block $(m, i, j-1, h', l', s')$ (see Fig. 6a). The other way is to obtain continuous block (m, i, j, h, l, s) by adding job $J_j \in V_2$ to continuous block $(m, i, j-1, h', l', s')$ (see Fig. 6b). Let $f(m, i, j, h, l, s)$ be the total weighted tardiness of the optimal continuous block composed of the jobs $J_i, J_{i+1}, \cdots, J_j$. To facilitate notation, we denote the tardiness of job J_i completing at time C in some continuous block by

$$T_i(C) = \max\left\{0,\ C - d_i\right\}. \tag{7}$$

For easy narration, we set up a dummy job J_0 with $a_0 = b_0 = 0$ for the following procedures as shown below.

DPA CB(T)

Initial conditions: For any $s \in \left[0,\ \sum_{t=0}^{i-1} a_t\right]$,

$$f(m, i, j, h, l, s) = \begin{cases} w_i T_i(s + a_i + b_i), \\ \quad if\ m = 1, i = j, h = a_i, l = b_i \\ \quad or\ m = 2, i = j, h = 0, l = a_i + b_i; \\ +\infty, \\ \quad \text{otherwise.} \end{cases} \tag{8}$$

Recursions:

For each m, i, j, h, l, s satisfying $m \in \{1, 2\}$, $1 \le i < j \le n$, $0 \le h \le \sum_{t=i}^{j} a_t$, $\underline{l} \le l \le \bar{l}$, $0 \le s \le \sum_{t=0}^{i-1} a_t$,

Case 1: A_j is processed on M_1 as shown in Fig. 6a.

$$f_1 = f(m, i, j-1, h-a_j, l+a_j-b_j, s) + w_j T_j(s+h+l) \tag{9}$$

Case 2: A_j is processed on M_2 as shown in Fig. 6b.

$$f_2 = \begin{cases} f(m, i, j-1, h, l-a_j-b_j, s) \\ +w_j T_j(s+h+l), \quad if\ l \ge a_j + b_j; \\ +\infty \quad \text{otherwise.} \end{cases}$$

$$f(m, i, j, h, l, s) = \min\left\{f_1, f_2\right\}. \tag{10}$$

Next, we use techniques similar to Section IV to construct the optimal partial schedule for problem FSFT. Let's first define partial schedule set (m, i, s) which represents the set of all partial schedules of job subset $\{J_i, J_{i+1}, \cdots, J_n\}$ where the first job J_i is processed at time s by machine M_m. Denote by $g(m, i, s)$ the minimum total weighted tardiness among all the partial schedules in set (m, i, s). For easy narration, we also set up a dummy job J_{n+1} with $a_{n+1} = b_{n+1} = +\infty$ beforehand. A DPA for calculating $g(m, i, s)$ is given as following.

DPA Sch(T)

Initial conditions: For any $m = 1, 2$ and $s \in \left[0,\ \sum_{i=1}^{n} a_i\right]$

$$g(m, n+1, s) = 0.$$

Recursions (A_{j+1} can only be processed on M_1 as shown in Fig. 7):

For each m, i, s satisfying $m \in \{1, 2\}$, $1 \le i \le n$ and $0 \le s \le \sum_{t=0}^{i-1} a_t$

$$g = \begin{cases} f(m, i, j, h, l, s) + g(1, j+1, h+s) \\ \quad if\ l < a_{j+1}; \\ +\infty \quad \text{otherwise.} \end{cases}$$

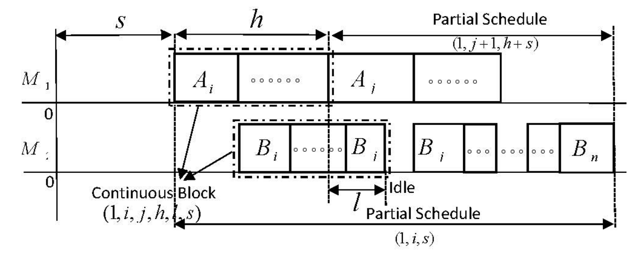

FIGURE 7. (m, i, s) is constructed from ($m, j + 1, h + s$) and (m, i, j, h, l, s).

TABLE 2. The detailed results of the three scheduling problems.

Objective	Complexity	Time Complexity	Remark
$L_{\max}$	Ordinary NP-hard	$O(n^2(\sum_{i=1}^n a_i)^2)$	Theorem 4.2
$\sum w_i T_i$	Ordinary NP-hard	$O(n^2(\sum_{i=1}^n a_i)^3)$	Theorem 5.2
$\sum w_i U_i$	Ordinary NP-hard	$O(n^2(\sum_{i=1}^n a_i)^3)$	Theorem 5.3

$$g(m, i, s) = \min_{\substack{i \le j \le n \\ 0 \le h \le \sum_{t=i}^{j} a_t \\ \underline{l} \le l \le \bar{l}}} \{g\};$$

Goal: $\min \sum w_i T_i = \min_{m \in \{1,2\}} \{g(m, 1, 0)\}$.

About the time complexity of DPA Sch(T), through an analysis similar to Section IV, it's easy for us to get the following theorem.

Theorem 5.2: Problem FSFT is solvable in $O(n^2(\sum_{i=1}^n a_i)^3)$ time. So it is also NP-hard in ordinary sense.

The above two procedures for problem FSFT can be easily adapted for problem FSFU with the same time complexity by replacing function $T_i(C)$ with $U_i(C)$ in Equation (8), (9) and (10) in DPA CB(T) where $U_i(C)$ is defined as following:

$$U_i(C) = \begin{cases} 1, & \text{if } C > d_i; \\ 0, & \text{if } C \le d_i. \end{cases}$$

Theorem 5.3: Problem FSFU is also solvable in $O(n^2(\sum_{i=1}^n a_i)^3)$ time and NP-hard in ordinary sense.

VI. COMPUTATIONAL EXPERIMENTS

A. TIME COMPLEXITY ANALYSIS OF OUR ALGORITHM

In Section III-V, we present the computational complexity of the three scheduling problems considered in this paper, their DPAs and the theoretical time complexity of algorithms. The detailed results are provided in Table 2.

To demonstrate the practical performance of the PDAs, we conducted computational experiments for the problem FSFT whose objective is minimize the total weighted tardiness. The computational experiments were implemented in MATLAB R2017b on a notebook computer equipped with an Intel Core i7 5500U CPU, 8GB RAM and Windows 10 64-bit operating system. The specific experimental environment was as follows: the weight of each job w_i, the processing time of the second task of each job b_i and the deadline of each job d_i were generated as uniformly distributed random numbers within the interval [0, 1], [0, 10] and [0, 1000] respectively; the processing times of the flexible tasks were produced by a random partition of a given amount $\sum_{i=1}^n a_i$ into n values such that $a_i \ge 0$ $(1 \le i \le n)$. The computational experiments were performed for $20 \le n \le 200$ with an interval of 20 and $100 \le \sum_{i=1}^n a_i \le 1000$ with an interval of 100. We generated 30 random test instances for each combination of n and $\sum_{i=1}^n a_i$. The average running times for all combinations are given in Table 3 where the leftmost column represents the number of jobs n and the top row represents the sum of processing times of all flexible tasks $\sum_{i=1}^n a_i$.

From Table 3, we can get even if the number of jobs reaches 200 and $\sum_{i=1}^n a_i$ reaches 500, the average running time only needs to be less than 138s. When $\sum_{i=1}^n a_i$ reaches 1000, the running time needs to be less than 567s. It shows that the actual running time of DPA is acceptable even when the parameters are large.

Next, according to the data in Table 3, we first analyze the change of running time with $\sum_{i=1}^n a_i$ under different number of jobs. When $\sum_{i=1}^n a_i$ is equal to 20, 100, and 200, the change of running time is shown in Fig. 8. Obviously, we can get that when the number of jobs is larger, the greater the slope of the curve, i.e., the growth rate of running time will be accelerated with the increase of $\sum_{i=1}^n a_i$.

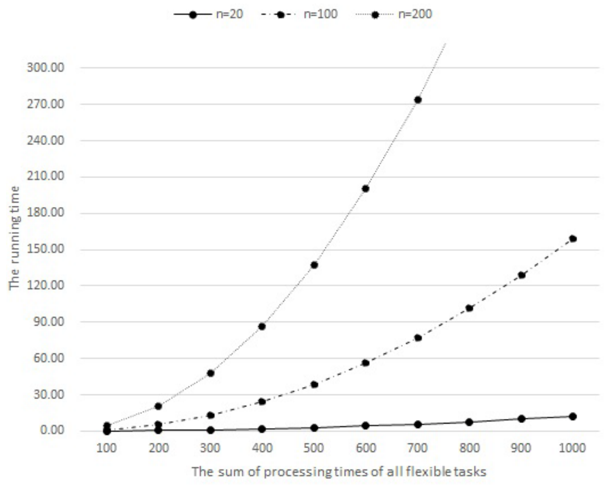

FIGURE 8. When $n = 20, 100, 200$, the running time changes with $\sum_{i=1}^n a_i$.

Then, according to the data in Table 3, we consider the change of running time with the number of jobs n under different $\sum_{i=1}^n a_i$. When $\sum_{i=1}^n a_i$ is equal to 100, 500 and 1000, the change of running time is shown in Fig. 9. We have that when $\sum_{i=1}^n a_i$ is larger, the growth rate of running time will be accelerated with the increase of the number of jobs.

B. COMPARISON WITH TRADITIONAL ALGORITHMS

Because other algorithms specifically for this problem have not been reported, we mainly analyze the effect of this DPA

TABLE 3. Average running times (in seconds) for $20 \le n \le 200$ and $100 \le \sum_{i=1}^{n} a_i \le 1000$.

n	$\sum_{i=1}^{n} a_i$									
	100	200	300	400	500	600	700	800	900	1000
20	0.11	0.45	1.03	1.87	2.95	4.27	5.85	7.68	9.75	12.07
40	0.27	1.23	2.88	5.23	8.28	12.03	16.48	21.64	27.49	34.04
60	0.51	2.36	5.56	10.11	16.03	23.29	31.92	41.90	53.24	65.93
80	0.82	3.84	9.07	16.52	26.18	38.06	52.16	68.47	87.00	107.74
100	1.21	5.67	13.42	24.44	38.75	56.33	77.20	101.34	128.76	159.47
120	1.67	7.86	18.60	33.89	53.72	78.10	107.04	140.52	178.54	221.12
140	2.20	10.40	24.61	44.85	71.11	103.38	51.63	186.00	236.33	292.69
160	2.81	13.29	31.46	57.33	90.90	132.16	181.13	237.78	302.13	374.18
180	3.49	16.53	39.15	71.34	113.11	164.45	241.76	295.87	375.94	465.59
200	4.24	20.12	47.66	86.86	137.72	200.24	274.42	360.26	457.76	566.92

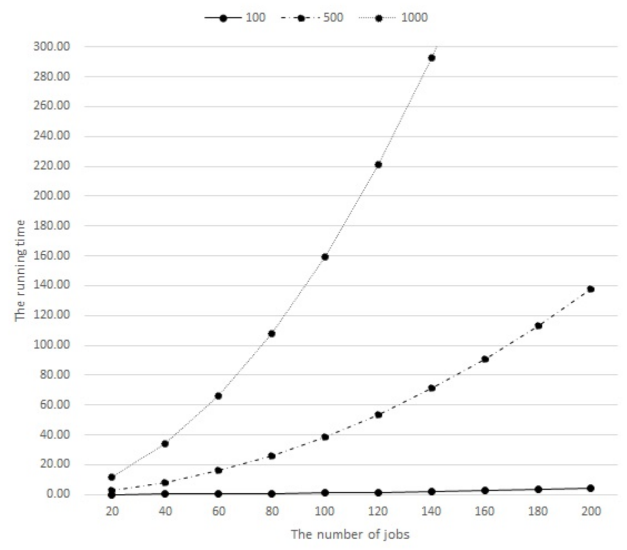

FIGURE 9. When $\sum_{i=1}^{n} a_i = 100, 500, 1000$, the running time changes with n.

by comparing it with other common algorithms with good effect for similar hybrid flow-shop problem.

In the existing research, there are two main kinds of algorithms used to solve similar hybrid flow-shop problems considered in this paper: one kind is the exact algorithm whose time complexity is exponential, such as enumeration method, branch-and-bound algorithm; the other kind is the heuristic algorithm which gives approximate solution in polynomial time, such as ant colony algorithm, greedy algorithm [19]. By retrieving the research results, branch-and-bound algorithm is a commonly used and effective method in getting the exact solution of hybrid flow-shop problem [10], [20]–[22]. And, in the aspect of heuristic algorithm, there are many effective algorithms based on iterated greedy idea for hybrid flow-shop problem [23]–[25]. Next, we also take the problem FSFT as an example to compare the effects of DPA (given in this paper), branch-and-bound algorithm (denoted as B&B, based on Lee and Kim [22]) and iterated greedy algorithm (denoted as IG, based on Wang and Wang [24]) on the two-machine hybrid flow-shop problem considered in this paper. The time complexity of above three algorithms in the worst-case for problem FSFT is as follows: DPA is an exact pseudo polynomial time algorithm whose time complexity is $O(n^2(\sum_{i=1}^{n} a_i)^3)$; B&B is an exact exponential algorithm whose time complexity is $O(n2^n)$ [22]; IG is a polynomial time approximation algorithm whose time complexity is $O(n^2)$ [24]. IG has advantages in terms of time complexity in the worst-case. B&B looks terrible, and the time complexity of DPA is between IG and B&B. However, considering the accuracy of the solution and the actual running time, the effect analysis of the three algorithms still needs to be verified by following computational experiments.

The software and hardware environment of computational experiments were the same as the previous subsection. Considering that the computational time complexity of B&B and IG is independent of $\sum_{i=1}^{n} a_i$, in order to make a comparison in the same standard, the computational experiments in this subsection were no longer classified according to $\sum_{i=1}^{n} a_i$ as in the previous subsection. We only grouped experiments according to the number of jobs. The specific experimental environment was as follows: the weight w_i, the processing time of the first task a_i, the processing time of the second task b_i and the deadline d_i were generated as uniformly distributed random numbers within the interval [0, 1], [0, 10], [0, 10] and [0, 1000] respectively; the first experiment was concerned with small-scale instances where $10 \le n \le 50$ with an interval of 5 and the second experiment was concerned with large-scale instances where $100 \le n \le 250$ with an interval of 50. We generated 20 random test instances for each n. Table 4 shows the test results of two kinds of experiments.

From Table 4, we can get even if the number of jobs is only 50 (not large) most of the instances processed by B&B algorithm can't be completed in 1200 seconds. It is easy to see that B&B algorithm is only suitable for small-scale instances.

TABLE 4. Average running times (in seconds) of DPA, B&B and IG for small-scale instances and large-scale instances.

	Small-scale instances									Large-scale instances			
n	10	15	20	25	30	35	40	45	50	100	150	200	250
DPA	1. 84	3. 10	4. 60	6. 34	8. 31	10. 51	12. 96	15. 63	18. 55	60. 66	122. 23	216. 62	329. 25
B&B	4. 15	9. 55	18. 05	32. 60	64. 23	153. 80	453. 11	980. 22[a] (8)[b]	1120 (16)	/[c]	/	/	/
IG	0. 93	2. 18	3. 46	4. 75	6. 06	7. 38	8. 71	10. 06	11. 41	25. 50	40. 35	55. 81	71. 81

[a]The number outside the parentheses indicates the average running time of the instance completed in 1200 seconds.
[b]The number in parentheses indicates the number of instances that cannot be completed in 1200 seconds.
[c]The B&B algorithm takes too much time to test in large-scale instances.

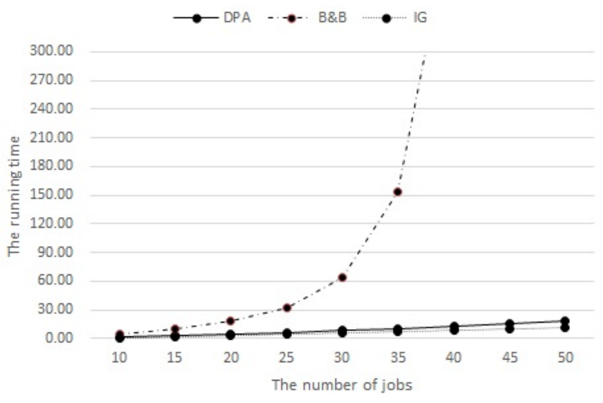

FIGURE 10. The average running times of DPA, B&B and IG for small-scale instances.

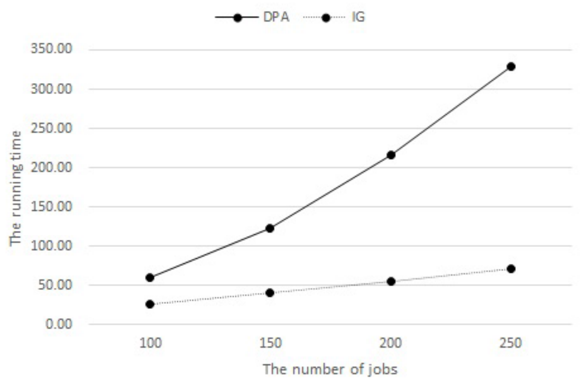

FIGURE 11. The average running times of DPA and IG for large-scale instances.

For large-scale instances, the B&B algorithm takes too much time to complete the experiment.

For small-scale instances, the average running times of DPA, B&B and IG are shown in Fig.10. It is easy to see that the running time of B&B increases with the number of jobs n much faster than the other two algorithms. When the number of jobs exceeds 45, B&B algorithm runs too long to solve this problem. The difference between DPA and IG in terms of running time is not obvious for small-scale instances.

For large-scale instances, the average running times of DPA, and IG are shown in Fig.11. Compared with IG, the disadvantage of DPA in running time is obvious and it gradually expands with the increase of the number of jobs n. But the average running times of DPA are still within the acceptable range for large-scale instances.

Although IG has advantages in running time, it can only get approximate solution rather than exact solution which can be given by DPA and B&B. The Average Relative Percentage Deviations [26] (denoted as ARPD) is usually used to compare the approximate effect of heuristic algorithm. Since we can get optimal solution by B&B and DPA, we let $ARPD = \frac{C_A - C^*}{C^*} \times 100\%$, where C_A is the solution of algorithm A and C^* is the optimal solution of the problem. We use ARPD to measure the approximation degree of IG, that is, the closer ARPD of IG is to 0, the better the approximation effect of IG is. Using the results of the small-scale experiment and the large-scale experiment, we get the ARPD of IG listed in Table 5.

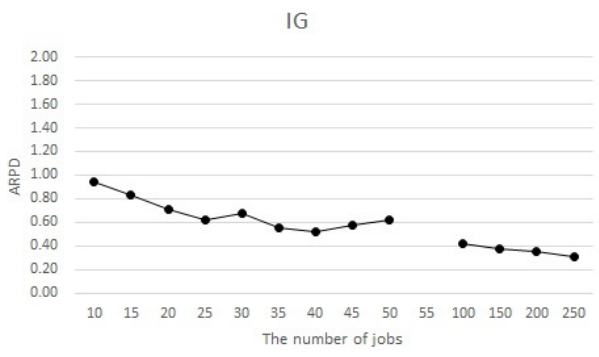

FIGURE 12. The average relative percentage deviations of IG change with the number of jobs for small-scale and large-scale instances.

The ARPD of IG for all instances is shown in Fig.12. We have that ARPD of IG is basically stable between 0.3 and 0.6 when the number of jobs exceeds 25 and it decreases slightly with the increase of the number of jobs.

C. SUMMARY OF ALGORITHM COMPARISON

Considering the running time and the accuracy of the calculation results, we have:

(1) When the number of jobs is less than 15, the difference of the three algorithms in running time has little effect on the actual situation. The DPA and B& B can get exact solution, so they have greater advantages in the accuracy of calculation results.

TABLE 5. Average relative percentage deviations of DPA, B&B and IG for small-scale instances and large-scale instances.

ARPD	n	Small-scale instances								Large-scale instances			
	10	15	20	25	30	35	40	45	50	100	150	200	250
DPA	0[a]	0	0	0	0	0	0	0	0	0	0	0	0
B&B	0[a]	0	0	0	0	0	0	0	0	/[b]	/	/	/
IG	0.94	0.83	0.71	0.62	0.68	0.55	0.52	0.58	0.62	0.42	0.37	0.35	0.31

[a]DPA and B&B are exact algorithm, so their ARPD is 0.
[b]The B&B algorithm takes too much time to test in large-scale instances.

(2) For small-scale instances with more than 15 jobs, compared with B&B, DPA has obvious advantages in running time. And compared with IG, DPA has obvious advantages in the accuracy of calculation results, while the difference in running time is not obvious.

(3) For large-scale instances, although DPA is not as good as IG in running time, it is still within the acceptable range (average running time less than 350 seconds), and has more than 30% advantages in the accuracy of calculation results. However, B&B needs too long running time to be used in practice.

VII. CONCLUSIONS

This paper discusses a two-stage two-machine hybrid flow-shop problem, which is widely used in shared manufacturing, cloud manufacturing and bar-coding operations in inventory or stock control systems. We mainly consider three objective functions with respect to deadline: minimizing the maximum lateness ($L_{\max}$), the total weighted tardiness ($\sum w_iT_i$) and the weighted number of tardy jobs ($\sum w_iU_i$). Firstly, we prove that they are all NP-hard in ordinary sense. Then, the pseudo-polynomial time DPA is designed respectively, and the time complexity of the algorithm is analysed. Finally, through computational experiments, we get that when the number of jobs is larger the growth rate of running time will be accelerated with the increase of $\sum_{i=1}^{n} a_i$ and when $\sum_{i=1}^{n} a_i$ is larger the growth rate of running time will be accelerated with the increase of n. The results of computational experiments also show that DPA has obvious advantages in running time compared with B&B and has more than 30% advantages in the accuracy of calculation results compared with IG.

For future research, we could consider designing efficient polynomial time approximation algorithms for these problems. The efficiency and effectiveness of the approximation algorithms and the DPAs in this paper will be compared.

REFERENCES

[1] G. L. Vairaktarakis and C.-Y. Lee, "Analysis of algorithms for two-stage flowshops with multi-processor task flexibility," *Nav. Res. Logistics*, vol. 51, no. 1, pp. 44–59, Feb. 2004.

[2] Y. W. Jiang and Q. Wei, "An improved algorithm for a class of flow-shop problems in graphics processing," *J. Autom.*, vol. 37, no. 11, pp. 1381–1386, 2011.

[3] W. Qi and H. Yong, "A two-stage semi-hybrid flowshop problem in graphics processing," *Appl. Math.-A J. Chin. Univ.*, vol. 20, no. 4, pp. 393–400, Dec. 2005.

[4] W. Zhong and Y. Shi, "Two-stage no-wait hybrid flowshop scheduling with inter-stage flexibility," *J. Combinat. Optim.*, vol. 35, no. 1, pp. 108–125, Jan. 2018.

[5] Q. Wei and Y.-W. Jiang, "Approximation algorithms for a two-stage hybrid flow shop," *J. Softw.*, vol. 23, no. 5, pp. 1073–1084, Aug. 2012.

[6] Q. Wei, Y. Wu, Y. Jiang, and T. C. E. Cheng, "Two-machine hybrid flow-shop scheduling with identical jobs: Solution algorithms and analysis of hybrid benefits," *J. Oper. Res. Soc.*, vol. 70, no. 5, pp. 817–826, May 2019.

[7] Y. Tan, L. Mönch, and J. W. Fowler, "A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines," *J. Scheduling*, vol. 21, no. 2, pp. 209–226, 2017.

[8] X. Feng, F. Zheng, and Y. Xu, "Robust scheduling of a two-stage hybrid flow shop with uncertain interval processing times," *Int. J. Prod. Res.*, vol. 54, no. 12, pp. 3706–3717, Jun. 2016.

[9] H. Ahonen and A. G. de Alvarenga, "Scheduling flexible flow shop with recirculation and machine sequence-dependent processing times: Formulation and solution procedures," *Int. J. Adv. Manuf. Technol.*, vol. 89, hboxnos. 1–4, pp. 765–777, Mar. 2017.

[10] L. Hidri, S. Elkosantini, and M. M. Mabkhot, "Exact and heuristic procedures for the two-center hybrid flow shop scheduling problem with transportation times," *IEEE Access*, vol. 6, pp. 21788–21801, 2018.

[11] Y. Zhang, S. Liu, and S. Sun, "Clustering and genetic algorithm based hybrid flowshop scheduling with multiple operations," *Math. Problems Eng.*, vol. 2014, pp. 1–8, 2014.

[12] S.-L. Jiang and L. Zhang, "Energy-oriented scheduling for hybrid flow shop with limited buffers through efficient multi-objective optimization," *IEEE Access*, vol. 7, pp. 34477–34487, 2019.

[13] B. M. T. Lin and F. J. Hwang, "Total completion time minimization in a 2-stage differentiation flowshop with fixed sequences per job type," *Inf. Process. Lett.*, vol. 111, no. 5, pp. 208–212, Feb. 2011.

[14] Y. M. Shafransky and V. A. Strusevich, "The open shop scheduling problem with a given sequence of jobs on one machine," *Nav. Res. Logistics*, vol. 45, no. 7, pp. 705–731, Oct. 1998.

[15] C.-F. Liaw, C.-Y. Cheng, and M. Chen, "The total completion time open shop scheduling problem with a given sequence of jobs on one machine," *Comput. Oper. Res.*, vol. 29, no. 9, pp. 1251–1266, Aug. 2002.

[16] J. Lassig, A. Awasthi, and O. Kramer, "Common due-date problem: Linear algorithm for a given job sequence," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng.*, Dec. 2014, pp. 97–104.

[17] A. Cheref, A. Agnetis, C. Artigues, and J.-C. Billaut, "Complexity results for an integrated single machine scheduling and outbound delivery problem with fixed sequence," *J. Scheduling*, vol. 20, no. 6, pp. 681–693, Dec. 2017.

[18] T. C. E. Cheng, S. A. Kravchenko, and B. M. T. Lin, "Server scheduling on parallel dedicated machines with fixed job sequences," *Nav. Res. Logistics*, vol. 66, no. 4, pp. 321–332, Jun. 2019.

[19] T.-S. Lee and Y.-T. Loong, "A review of scheduling problem and resolution methods in flexible flow shop," *Int. J. Ind. Eng. Comput.*, vol. 10, pp. 67–88, 2019.

[20] S. J. Wang, X. D. Wang, and L. Yu, "Two-stage no-wait hybrid flowshop scheduling with sequence-dependent setup times," *Int. J. Syst. Sci., Oper. Logistics*, to be published, doi: 10.1080/23302674.2019.1575997.

[21] Y.-K. Lin, D.-H. Huang, and C.-F. Huang, "Estimated network reliability evaluation for a stochastic flexible flow shop network with different types of jobs," *Comput. Ind. Eng.*, vol. 98, pp. 401–412, Aug. 2016.

[22] G.-C. Lee and Y.-D. Kim, "A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness," *Int. J. Prod. Res.*, vol. 42, no. 22, pp. 4731–4743, Nov. 2004.

[23] W. Shao, Z. Shao, and D. Pi, "Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem," *Knowl.-Based Syst.*, Jan. 2020, Art. no. 105527.

[24] J.-J. Wang and L. Wang, "An iterated greedy algorithm for distributed hybrid flowshop scheduling problem with total tardiness minimization," in *Proc. IEEE 15th Int. Conf. Autom. Sci. Eng. (CASE)*, Vancouver, BC, Canada, Aug. 2019, pp. 350–355.

[25] H. Öztop, M. Fatih Tasgetiren, D. T. Eliiyi, and Q.-K. Pan, "Metaheuristic algorithms for the hybrid flowshop scheduling problem," *Comput. Oper. Res.*, vol. 111, pp. 177–196, Nov. 2019.

[26] A. Priya and S. K. Sahana, "Multiprocessor scheduling based on evolutionary technique for solving permutation flow shop problem," *IEEE Access*, vol. 8, pp. 53151–53161, 2020.

QI WEI was born in Ningbo, Zhejiang, China, in 1980. He received the M.S. degree in operational research and cybernetics from Zhejiang University, Zhejiang, in 2006, and the Ph.D. degree in operational research and cybernetics from Shanghai University, Shanghai, China, in 2015.

From 2006 to 2016, he was a Lecturer with the Ningbo Institute of Technology, Zhejiang University. Since 2016, he has been an Associate Professor with the Ningbo University of Finance and Economics. He is the author of more than 20 articles. His research interests include computational complexity, algorithm designs, game theory, and scheduling.

Dr. Wei won the Honorary Title of Leading and Top Talent in Ningbo. He has presided over and participated in three national projects in China.

YONG WU was born in Jiujiang, Jiangxi, China, in 1983. He received the B.E. degree in measurement and control technology and instruments from the Anhui University of Science and Technology, Anhui, in 2003, and the M.S. degree in operational research and cybernetics and the Ph.D. degree in operational research and cybernetics from Zhejiang University, Hangzhou, China, in 2006 and 2009, respectively.

Since 2018, he has been a Professor with the Mathematics Department, Ningbo Institute of Technology, Zhejiang University. He is the author of more than ten articles. His research interests include algorithm design and analysis, scheduling problem, and scheduling game.