Computer Modeling in
Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2022.019754

ARTICLE

Two-Machine Hybrid Flow-Shop Problems in Shared Manufacturing

Qi Wei* **and Yong Wu**

Ningbo University of Finance & Economics, Ningbo, 315175, China

*Corresponding Author: Qi Wei. Email: weiqi@nbufe.edu.cn

Received: 13 October 2021 Accepted: 01 November 2021

ABSTRACT

In the "shared manufacturing" environment, based on fairness, shared manufacturing platforms often require manufacturing service enterprises to arrange production according to the principle of "order first, finish first" which leads to a series of scheduling problems with fixed processing sequences. In this paper, two two-machine hybrid flow-shop problems with fixed processing sequences are studied. Each job has two tasks. The first task is flexible, which can be processed on either of the two machines, and the second task must be processed on the second machine after the first task is completed. We consider two objective functions: to minimize the makespan and to minimize the total weighted completion time. First, we show the problem for any one of the two objectives is ordinary NP-hard by polynomial-time Turing Reduction. Then, using the Continuous Processing Module (CPM), we design a dynamic programming algorithm for each case and calculate the time complexity of each algorithm. Finally, numerical experiments are used to analyze the effect of dynamic programming algorithms in practical operations. Comparative experiments show that these dynamic programming algorithms have comprehensive advantages over the branch and bound algorithm (a classical exact algorithm) and the discrete harmony search algorithm (a high-performance heuristic algorithm).

KEYWORDS

Hybrid flow-shop; dynamic programming algorithm; computational complexity; numerical experiments; shared manufacturing

1 Introduction

1.1 Problem Statement

In this paper, we study two two-machine hybrid flow-shop scheduling problems with fixed processing sequences. In this problem, a set of n jobs $J = \{J_1, J_2, \cdots, J_n\}$ is processed in a two-machine two-stage flow-shop with 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, where task A_i is flexible which can be processed on either of the two machines (M_1 and M_2) for a_i time units, and task B_i is inflexible which must be processed on M_2 for b_i time units. Task B_i cannot be processed until task A_i is completed, and pre-emption is not allowed. The processing order of the jobs is given in advance. Without losing generality, we assume that the jobs are processed according to the subscript order, i.e., for any two tasks processed on the same machine, the task with a small subscript must be completed before the task

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

with a large subscript. The objectives are to minimize the makespan (the maximum completion time) and minimize the total weighted completion time.

Hybrid flow-shop problems are widely used in the traditional manufacturing industry [1], computer graphics processing [2], medical operation scheduling [3], and other fields. For the research on the hybrid flow-shop problem, please refer to the latest review [4]. A typical application scenario of the two-stage hybrid flow-shop scheduling is as follows. In integrated circuit manufacturing or mechanical manufacturing, product processing usually includes two stages: preliminary processing and finishing processing. The factories are equipped with two sets of equipment, one set of low-precision equipment for preliminary processing and the other set of high-precision equipment for finishing processing. However, when necessary, the high-precision equipment can also be degraded for preliminary processing. Reasonably arrange the processing sequence and processing machines can make all products complete as soon as possible.

The scheduling problems with fixed processing sequences refer to the scheduling problem in which the processing order of the jobs is determined in advance. This kind of problem is originally mainly applied to the stock control systems [5]. In recent years, with the integration of the sharing economy into the traditional manufacturing industry, there has been a shared manufacturing mode that shares manufacturing resources and demands on the network platform and matches supply and demand through advanced algorithms [6]. “Alibaba Taobao Factory” and “Aerospace Cooperative Manufacturing Network” in China, “Wanju-gun Local Food Processing Center” in Korea, and many other share manufacturing practices have been vigorously developed. Based on fairness, shared manufacturing platforms often require manufacturing service enterprises to arrange production according to the principle of “order first, finish first” which forms a constraint of fixed processing sequences. Suppose the two-stage hybrid flow-shop scheduling problem with two machines described above is considered in the shared manufacturing environment. It is equivalent to adding the constraint of fixed processing sequences in the above typical application scenario. Obviously, the mathematical model of this problem is the scheduling model described in the first paragraph of this section.

1.2 Related Problems and Results

Wei et al. [2] first considered this two-stage hybrid flow-shop scheduling problem without fixed processing sequences constraint (denoted by SHFS) in 2005. They proved that this problem is ordinary NP-hard and presented an approximation algorithm with polynomial-time computational complexity whose worst-case ratio is 2. For the same problem, Jiang et al. [7] designed an improved approximation algorithm whose worst-case ratio is 1.5. Recently, for the case where every job has a deadline and must be processed in a given order, Wei et al. [8] proposed efficient algorithms.

The research results of other hybrid flow-shop scheduling problems with two stages close to the problems studied in this paper are as follows. A special hybrid flow-shop scheduling problem with two machines was considered by Vairaktarakis et al. [1]. In their problem, the two tasks of each job can be processed on any one of the two machines, i.e., there are four processing methods in total. An excellent approximate algorithm was presented whose worst-case ratio is only 1.618. Zhang et al. [9] studied another similar problem where the later task of each job can be jointly processed by multiple machines, but the former task cannot. An approximate algorithm was presented in their paper whose worst-case ratio is only $2 - \varepsilon$. For the special case of the above problem raised by Zhang et al. [9], an improved approximation algorithm was presented by Peng et al. [10–12] respectively studied two hybrid flow-shop problems where all batching machines are at two stages. The objective function of the former problem considered by Wang et al. [11] is to minimize the total weighted completion time. Mixed-integer linear programming (MILP) was constructed, and an efficient heuristic algorithm was presented. The objective of the latter problem studied by Tan et al. [12] is to minimize the total weighted delay time. They proposed an iterative decomposition method to solve it. Some scholars have also studied the two-stage hybrid flow-shop where no buffer capacity exists between the two stages. For the latest research results on such problems, please refer to the research of Dong et al. [13–15]. For other latest studies on two-stage hybrid flow-shop under different parameter conditions and objective functions, please refer to Feng et al. [16–21], etc. However, these problems do not consider the constraint of fixed processing sequences.

The earliest scheduling problem with fixed processing sequences was presented by Shafransky et al. [22]. They considered the model with fixed processing sequences in an open-shop problem, showed this problem is NP-hard in the strong sense, and proposed a very accurate polynomial-time approximate algorithm whose worst-case ratio is only 1.25. Lin et al. [5] studied a two-stage differentiation flow-shop to minimize the total completion time and proposed a dynamic programming algorithm. Hwang et al. [23,24] firstly studied the general two-machine flow-shop problem with a given sequence and presented an optimal algorithm for each of the two special cases. Then, they studied the general flow-shop scheduling with batching machines and proposed optimal algorithms for several models. With the gradual emergence of the application value of the scheduling problem with fixed processing sequences, the related research has increased a lot in recent years. Lin et al. [25] considered a relocation scheduling problem with fixed processing sequences corresponding to resource-constrained scheduling on two parallel dedicated machines. They proposed polynomial-time optimal algorithms or approximate algorithms for several models with different objective functions. Halman et al. [26] further gave a fully polynomial-time approximation scheme for the above problem. Cheref et al. [27] studied a scheduling problem considering production and delivery with a given sequence. They showed that this problem is NP-hard and proposed a dynamic programming algorithm with good effect. A server scheduling problem on parallel dedicated machines with fixed processing sequences was considered by Cheng et al. [28,29]. For the two-machine case, a polynomial-time approximation algorithm was presented; for the case where each loading time is unit, two heuristic algorithms were proposed; and for the general case, a pseudo-polynomial algorithm was presented. However, the two-stage hybrid flow-shop scheduling with fixed processing sequences and the goal of minimizing makespan or total weighted completion time has not been considered yet.

1.3 Our Results

In the "shared manufacturing" environment, we introduce the constraint of fixed processing sequences into the two-machine two-stage hybrid flow-shop scheduling problem SHFS. We consider two objectives of this problem: to minimize the makespan and to minimize the total weighted completion time. For each problem, we analyze the computational complexity and present a dynamic programming algorithm. According to the three-field representation, these two problems can be expressed in the following form:

(1) $FS_2|FJS, Hybrid|C_{\max}$,

(2) $FS_2|FJS, Hybrid|\sum w_iC_i$,

which are denoted by SHFSFC and SHFSFW.

Firstly, we give the structural characteristics of one optimal solution and prove that these two problems are both ordinary NP-hard. Then we propose the dynamic programming algorithms and calculate their time complexity. Finally, we show the advantages of these algorithms over the existing exact algorithms and heuristic algorithms in practical efficiency and effect by numerical experiments.

The rest of this paper is arranged as follows: In Section 2, we first give the basic symbolic assumptions and the characteristics of one optimal solution, then analyze the computational complexity of the problems; In Sections 3 and 4, the dynamic programming algorithms for SHFSFC and SHFSFW are proposed, respectively; In Section 5, we compare the dynamic programming algorithms presented in this paper with the existing algorithms through numerical experiments to obtain the actual efficiency and effect of these algorithms. Finally, we conclude the paper and give the future research ideas in Section 6.

2 Symbolic Assumptions, Structure of One Optimal Schedule, and Complexity of the Problems

The basic symbols to be used in the following are given, properties of one optimal schedule of SHFSFC (SHFSFW) are analyzed, and the computational complexity of each problem is proved in this section.

2.1 Symbolic Hypothesis

The symbols and their meanings to be used below are as follows:

- a_i: The processing time of task A_i (the first task of job J_i) on machine M_1 or M_2;
- b_i: The processing time of task B_i (the second task of job J_i) on machine M_2;
- C_i: The completion time of job J_i;
- w_i: The weight of job J_i;
- V_1: The job set $\{J_i | A_i$ is processed on machine $M_1\}$;
- V_2: The job set $\{J_i | A_i$ is processed on machine $M_2\}$.

2.2 Structure of Optimal Scheduling

The two problems studied in this paper do not consider the buffer capacity between the two stages. That is, the buffer capacity is deemed to be infinite. So the tasks processed on machine M_1 can be processed as early as possible, i.e., there is no idle on M_1 until completion. If job $J_i \in V_1$, task B_i cannot be processed until task A_i is completed. So, there may be an idle time on machine M_2 before task B_i starts to be processed. But if job $J_i \in V_2$, task A_i and B_i can be processed immediately after task B_{i-1} on M_2 without any idle time.

From the above analysis, it is easy to find that the following proposition holds whether the objective is to minimize makespan or total weighted completion time.

Proposition 2.1 There exists one optimal schedule of SHFSFC (SHFSFW) that satisfies the following properties at the same time:

(1) Task A_1 is processed on M_2, and task A_n is processed on M_1;
(2) M_1 has no idle time until processing is complete;
(3) There is no idle time on M_2 between any task of the job in V_2 and its previous task;
(4) There is no idle time on M_2 between the second task B_i of job $J_i \in V_1$ and its previous task, or the starting processing time of B_i is exactly equal to the completion time of A_i.

Proof:

(1) Suppose ϕ_1 is an optimal schedule of SHFSFC (SHFSFW) where A_1 is processed on M_1 and A_n is processed on M_2. We construct a new schedule φ_1 based on ϕ_1 as follows: Change the processing mode of J_1 and J_n, i.e., A_1 and B_1 are all processed on M_2 and A_n is processed on M_1; Keep other task processing arrangements unchanged. Since B_1 cannot be processed on M_2 until A_1 is completed on M_1 in ϕ_1, there is an idle time of a_1 time units before task B_1 starts to be processed on M_2 in ϕ_1. Therefore A_1 can be processed on the above idle time on M_2 in φ_1 without affecting the completion time of B_1. And since J_n is the last job, no task is processed on M_1 after the starting processing time of A_n (denoted by $S(A_n)$) in ϕ_1. So A_n can be processed on M_1 from $S(A_n)$ to $S(A_n)+a_n$ in φ_1 without affecting the completion time of B_n on M_2. Obviously, φ_1 is feasible, and the makespan and the total weighted completion time in φ_1 are all equal to them in ϕ_1. So we have that φ_1 is also an optimal schedule which implies property (1) holds.

(2) Suppose ϕ_2 is an optimal schedule of SHFSFC (SHFSFW) satisfying property (1) where there exists an idle time between two successive processed tasks A_i and A_j on machine M_1. Next, we construct a new schedule φ_2 based on ϕ_2 as follows: Advance the start processing time of task A_j to the completion time of task A_i so that there is no idle time between them; Keep other task processing arrangements unchanged. Since the buffer capacity between two stages is not considered, the task A_j is processed in advance, and task B_j remains intact, the starting processing time of B_j is still larger than the completion time of A_j, i.e., φ_2 is still feasible. Considering that the processing schedules of all second tasks have not changed in φ_2, the completion time of each job is the same in ϕ_2 and φ_2. For ϕ_2 is an optimal schedule of SHFSFC (SHFSFW), φ_2 is also an optimal schedule. On M_1, by a similar method, all idle times before M_1 finishing processing can be eliminated, and the schedule is still optimal, which implies property (1) and (2) hold simultaneously.

(3) Suppose ϕ_3 is an optimal schedule of SHFSFC (SHFSFW) satisfying property (1) and (2) where there is an idle time on M_2 between the task A_i (or B_i) of job $J_i \in V_2$ and its previous task. Next, we construct a new schedule φ_3 based on ϕ_3 as follows: Advance the start processing time of task A_i (or B_i) to the completion time of its previous task on M_2, so that there is no idle time between them; Keep other task processing arrangements unchanged. Since the processing arrangements of the jobs in V_1 have not changed, φ_3 is feasible. And for the completion time of the job in φ_3 is equal to or less than that in ϕ_3, the makespan and the total weighted completion time in φ_3 are all less than or equal to them in ϕ_3. According that ϕ_3 is an optimal schedule, φ_3 is also an optimal schedule. Obviously, by a similar method, all idle times on M_2 between any task of the job in V_2 and its previous task can be eliminated, and the schedule is still optimal, which implies property (1), (2), and (3) hold at the same time.

(4) Suppose ϕ_4 is an optimal schedule of SHFSFC (SHFSFW) satisfying property (1), (2), and (3) where there is an idle time on M_2 between the second task B_i of the job $J_i \in V_1$ and its previous task, and the starting processing time of B_i is larger than the completion time of A_i. Next, we also construct a new schedule φ_4 based on ϕ_4 as follows: Advance the start processing time of task B_i to the larger value of the completion time of its previous task on M_2 and the one of A_i, so that there is no idle time between B_i and its previous task or the starting processing time of B_i is exactly equal to the completion time of A_i; Keep other task processing arrangements unchanged. Obviously, through a simple analysis

similar to the proof of (2) or (3), we can have that φ_4 is also an optimal schedule and then get property (1), (2), (3), and (4) hold at the same time.

To sum up, we can get Proposition 2.1 holds.

According to Proposition 2.1, it is easy to obtain there exists an optimal schedule of SHFSFC (or SHFSFW), which is shown in Fig. 1 below. That is, there exists an optimal schedule of SHFSFC (or SHFSFW) in which A_1 and A_n are processed on M_1 and M_2, respectively, and the continuous processing tasks are divided by some idle times before the second tasks of some jobs in V_1 on M_2. These continuous processing tasks are called **Continuous Processing Modules** (denoted by CPM) in the following study, which will help design dynamic programming algorithms in Sections 3 and 4.

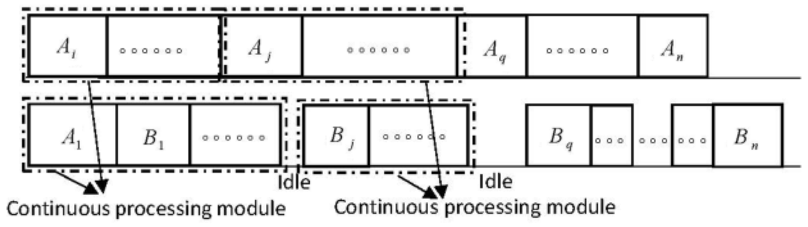

Figure 1: The structure of an optimal schedule

2.3 *Computational Complexity of the Problems*

Next, we prove that SHFSFC and SHFSFW are both NP-hard by Turing Reduction.

Theorem 2.2 Problem SHFSFC is NP-hard, even if the processing time of the second task of every job is 0, i.e., $b_i = 0$ for all i.

Proof: First, propose an Instance I of a well-known NP-hard problem Partition Problem: There is a set of n integers $E = \{e_1, e_2, \cdots, e_n\}$ and an integer $C = \frac{1}{2}\sum_{i=1}^{n} e_i$. Can set E be divided into two disjoint subsets E_1 and E_2, such that $E_1 \cap E_2 = \emptyset$, $E_1 \cup E_2 = E$, and $\sum_{e_i \in E_1} e_i = \sum_{e_i \in E_2} e_i = C$?

Then, based on Instance I, we construct an Instance II of SHFSFC: There is a job set of $n+2$ jobs $J = \{J_0, J_1, J_2, \cdots, J_n, J_{n+1}\}$, where

(1) J_0: $a_0 = C - \varepsilon$ and $b_0 = \varepsilon$;

(2) J_i: $a_i = e_i$ and $b_i = \varepsilon$ $(i = 1, 2, \cdots n)$;

(3) J_{n+1}: $a_{n+1} = C + n\varepsilon$ and $b_{n+1} = 0$ (ε is a number much smaller than C).

Is there a feasible schedule of Instance II that makes makespan $C_{\max} = 2C + n\varepsilon$?

Now, let's prove that the solutions of Instances I and II can be derived from each other. Suppose (E_1, E_2) is a solution of Instance I, i.e., E_1 and E_2 satisfy $E_1 \cap E_2 = \emptyset$, $E_1 \cup E_2 = E$, and $\sum_{e_i \in E_1} e_i = \sum_{e_i \in E_2} e_i = C$. Next, we construct the solution of Instance II as follows: Let $V_1 = \{J_i | e_i \in E_1\} \cup \{J_{n+1}\}$ and $V_2 = \{J_0\} \cup \{J_i | e_i \in E_2\}$. It is easy to have $C_{\max} = C + \sum_{J_i \in V_2/\{J_0\}} a_i + n\varepsilon = C + \sum_{e_i \in E_2} e_i + n\varepsilon = 2C + n\varepsilon$, which implies (V_1, V_2) is a solution of Instance II.

Suppose (V_1, V_2) is a solution of Instance II. Let's first prove that J_0 must belong to set V_2, i.e., the task A_0 must be processed on M_2. If A_0 is processed on M_1, M_2 has an idle time from time 0 to time a_0 $(= C - \varepsilon)$. So, after time $C - \varepsilon$, the total processing time of the rest tasks to be processed by two machines is $\sum_{i=0}^{n+1} (a_i + b_i) - (C - \varepsilon) = 3C + (2n + 1)\varepsilon$. We can easily get the makespan is at least $(C - \varepsilon) + 1/2\ (3C + (2n + 1)\varepsilon) = 5/2\ C + n\varepsilon - 1/2\ \varepsilon > 2C + n\varepsilon$, which is a contradiction with $C_{\max} = 2C + n\varepsilon$. Therefore, we have $J_0 \in V_2$.

Next, we prove that J_{n+1} must belong to set V_1, i.e., task A_{n+1} must be processed on M_1. If A_{n+1} is processed on M_2, no task can be processed on machine M_1 after the starting processing time of task A_{n+1}. So, before the starting processing time of task A_{n+1}, the total processing time of the tasks to be processed by two machines is $(4C + 2n\varepsilon) - (C + n\varepsilon) = 3C + n\varepsilon$. We can easily get the makespan is at least $1/2\,(3C + n\varepsilon) + a_{n+1} + b_{n+1} = 1/2\,(3C + n\varepsilon) + C + n\varepsilon \geq 5/2\,C + 3/2\,n\varepsilon > 2C + n\varepsilon$ which is a contradiction with $C_{\max} = 2C + n\varepsilon$. Therefore, we have $J_{n+1} \in V_1$.

Considering that $C_{\max} = 2C + n\varepsilon$ and the total processing time of all jobs is $4C + 2n\varepsilon$, it is easy to have that each machine's load is $2C + n\varepsilon$. And since $J_{n+1} \in V_1$, the total processing time of the tasks processed on M_1 except A_{n+1} is $2C + n\varepsilon - (C + n\varepsilon) = C$, i.e., we have $\sum_{J_i \in V_1/\{J_{n+1}\}} a_i = \sum_{J_i \in V_1/\{J_{n+1}\}} e_i = C$. Similarly, we can have that the total processing time of the tasks processed on M_2 except A_0 and the second tasks of all jobs is $2C + n\varepsilon - C - n\varepsilon = C$, i.e., we have $\sum_{J_i \in V_2/\{J_0\}} a_i = \sum_{J_i \in V_2/\{J_0\}} e_i = C$. Now we let $E_1 = \{e_i | J_i \in V_1/\{J_{n+1}\}\}$ and $E_2 = \{e_i | J_i \in V_2/\{J_0\}\}$. Obviously, we have $\sum_{e_i \in E_1} e_i = C$ and $\sum_{e_i \in E_2} e_i = C$, which implies (E_1, E_2) is a solution of Instance I.

It is known that Partition Problem is NP-hard, so it is easy to have that SHFSFC is also NP-hard.

Obviously, the above proof process still holds when $\varepsilon = 0$, so it can be obtained that Theorem 2.2 holds even if the processing time of the second task for every job is 0.

Theorem 2.3 SHFSFW is NP-hard.

Proof: Considering that the jobs are processed in subscript order, we have $C_{\max} = C_n$. So when $w_1 = w_2 = \cdots = w_{n-1} = 0$, $w_n = 1$, problem SHFSFC is a special case of problem SHFSFW. We have proved that SHFSFC is NP-hard, so SHFSFW is also NP-hard.

3 A Dynamic Programming Algorithm of Problem SHFSFC

According to Proposition 2.1, there is an optimal schedule of Problem SHFSFC, which is composed of several continuous processing modules and the idle times between continuous processing modules. Therefore, the following dynamic programming algorithm includes two stages: Construct the optimal continuous processing modules and connect the optimal continuous processing modules through idle times to form an optimal schedule. Firstly, the strict definition of continuous processing module for SHFSFC is given:

Definition 3.1 A sub-schedule consisting of a job subset $\{J_i, J_{i+1}, \cdots, J_j\}$ is called a continuous processing module of Problem SHFSFC if it satisfies the following conditions (see Fig. 2), denoted by four elements CPM(m, i, j, l).

(1) The first task of job $J_i A_i$ is processed on M_m $(m = 1$ or $2)$;
(2) No matter on M_1 or M_2, there is no idle time between any two continuously processed tasks;
(3) The difference between the complete time of subset $\{J_i, J_{i+1}, \cdots, J_j\}$ on M_1 and M_2 is equal to l.

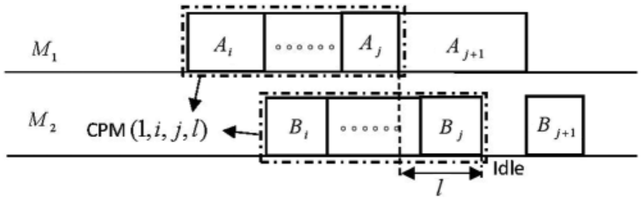

Figure 2: The continuous processing module(1, i, j, l)

It is easy to see that CPM(m, i, j, l) can be constructed by CPM$(m, i, j-1, l')$ using two different methods (as shown in Fig. 3). The first method to get CPM(m, i, j, l) is to add job $J_j \in V_1$ to CPM$(m, i, j-1, l')$ (see Fig. 3a). And the second method to get CPM(m, i, j, l) is to add job $J_j \in V_2$ to CPM$(m, i, j-1, l')$ (see Fig. 3b). We assume that $f(m, i, j, l)$ is the minimum load generated by the continuous processing module on machine M_1 composed of job subset $\{J_i, J_{i+1}, \cdots, J_j\}$.

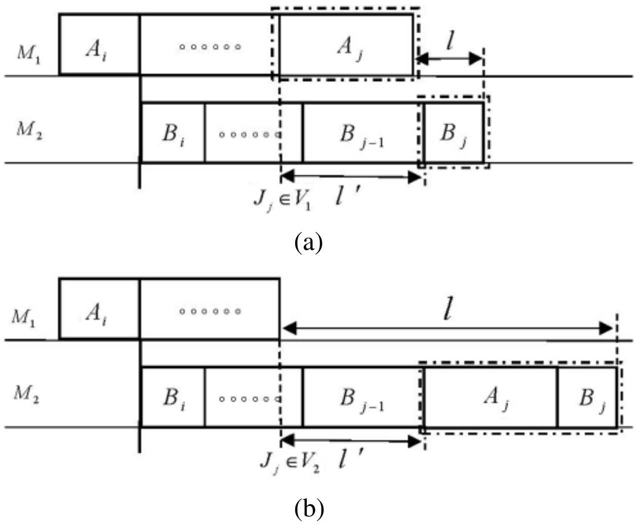

Figure 3: The two ways from CPM(1, i, $j-1$, l') to CPM(1, i, j, l) as shown in (a) and (b)

For the convenience of narration, a symbolic function is defined below:

Definition 3.2 Symbolic function $\sigma(m)$ is defined as follows:

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

When $i < j$, according to the definition of CPM(m, i, j, l), it is easy to get the following formulas hold:

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

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

$$l \geq b_j. \tag{3}$$

In the following discussion, we let

$$\underline{l} = \max\left\{b_j,\ a_i\sigma(m) + \sum_{t=i}^{j} b_t - \sum_{t=i+1}^{j} a_t\right\},$$

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

Obviously, in the sub-schedule composed of job subset $\{J_i, J_{i+1}, \cdots, J_j\}$, the difference in completion time between two machines l must be obtained in the interval $\left[\underline{l}\ \ \overline{l}\right]$. The maximum completion time of CPM(m,i,j,l) is $f(m,i,j,l)$+l. So, when l is fixed, minimizing the maximum completion time is equal to minimizing $f(m,i,j,l)$. Therefore the optimal continuous processing module is obtained when $f(m,i,j,l)$ is minimum. The dynamic programming algorithm for calculating the load $f(m,i,j,l)$ of the optimal CPM(m,i,j,l) on machine M_1 is given below:

Dynamic Programming Algorithm CPM(C):

Initial Conditions:

$$f(m,i,j,l) = \begin{cases} a_i, & \text{when } m = 1,\ i = j,\ l = b_i; \\ 0, & \text{when } m = 2,\ i = j,\ l = a_i + b_i; \\ +\infty & \text{otherwise.} \end{cases} \tag{4}$$

Recurrence Relation:

The parameters m,i,j,l respectively satisfy $m \in \{1,2\}$, $1 \leq i < j \leq n$, $\underline{l} \leq l \leq \overline{l}$.

(1) When A_j is processed on M_1 (see Fig. 3a),

$f_1 = f(m,i,j-1,l+a_j-b_j) + a_j$;

(2) When A_j is processed on M_2 (see Fig. 3b),

$$f_2 = \begin{cases} f(m,i,j-1,l-a_j-b_j) & \text{when } l \geq a_j + b_j; \\ +\infty & \text{otherwise.} \end{cases}$$

Recurrence Formula: $f(m,i,j,l) = \min\{f_1, f_2\}$.

The initial conditions and recurrence formula in dynamic programming algorithm CPM(C) are clearly valid. So we mainly analyze the recurrence relation below. Given a set of required parameters m,i,j,l, the load generated by the optimal CPM(m,i,j,l) on M_1 $f(m,i,j,l)$ can be obtained by two different methods. The first method is to put task A_j on M_1 and task B_j on M_2 (as shown in Fig. 3a). So we have $l' + b_j = l + a_j$ which implies $l' = l + a_j - b_j$, subject to $l' \geq a_j$, i.e., $l \geq b_j$. According to the value range of l, we have $l \geq b_j$ always holds. Therefore the relation (1) in Recurrence Relation holds. The other method is to put A_j and B_j both on M_2 (as shown in Fig. 3b). So we have $l = l' + a_j + b_j$ which implies $l' = l - a_j - b_j$, subject to $l' \geq 0$, i.e., $l \geq a_j + b_j$. Therefore the relation (2) in Recurrence Relation holds.

After the optimal continuous processing modules are constructed, the whole schedule can be constructed by recursively connecting these optimal continuous processing modules. It should be noted that every two continuous optimal continuous processing modules are separated by an idle time on M_2. First, we let the Partial Schedule (m, i) be a sub-schedule of the job set $\{J_i, J_{i+1}, \cdots, J_n\}$, denoted by PS$(m, i)$, where the first job $J_i \in V_m$, $m \in \{1, 2\}$. Then let $g(m, i)$ be the minimum makespan of PS(m, i). Next, we give a dynamic programming algorithm to calculate the makespan $g(m, i)$ of the optimal PS(m, i). For the convenience of narration, we set up a virtual job J_{n+1} with processing time $a_{n+1} = b_{n+1} = +\infty$.

Dynamic Programming Algorithm DP(C):

Initial Conditions:

$g(m, n+1) = 0, \ m = 1, 2;$

Recurrence Relation:

The parameters m, i, respectively, satisfy $m \in \{1, 2\}$, $1 \leq i \leq n$.

$$g = \begin{cases} f(m, i, j, l) + l \left\lfloor \frac{j}{n} \right\rfloor + g(1, j+1) & \text{when } \ l < a_{j+1}; \\ +\infty & \text{otherwise.} \end{cases} \tag{5}$$

Recurrence Formula: $g(m, i) = \min_{\substack{i \leq j \leq n \\ \underline{l} \leq l \leq \bar{l}}} \{g\}$.

Target Value: $\min C_{\max} = \min_{m \in \{1,2\}} \{g(m, 1)\}$.

The initial conditions, recurrence formula, and target value in dynamic programming algorithm DP(C) are clearly valid. So we also mainly analyze the recurrence relation below. When the optimal CPM(m, i, j, l) is given, PS(m, i) can be structured by the optimal PS$(m, j+1)$. Since there is an idle time between CPM(m, i, j, l) and PS$(m, j+1)$, according to Proposition 2.1, we can get m is equal to 1 in PS$(m, j+1)$, i.e., task A_{j+1} is processed on M_1 as shown in Fig. 4. It is easy to get $g(m, i) = \min f(m, i, j, l) + g(1, j+1)$ when $i < n$, and $g(m, i) = \min f(m, i, j, l) + g(1, j+1) + l$ when $i = n$. According that there is an idle time between job J_j and J_{j+1}, $l < a_{j+1}$ must be satisfied. So Eq. (5) holds.

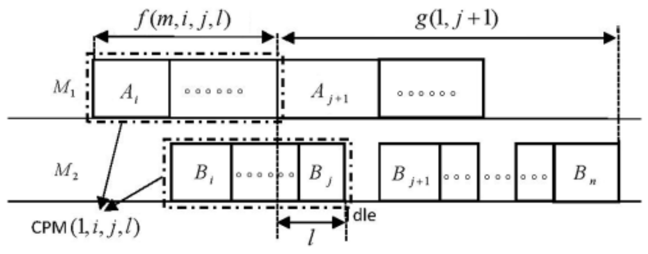

Figure 4: From CPM(m, i, j, l) and PS$(m, j+1)$ to PS (m, i)

Theorem 3.3 Dynamic programming algorithm DP(C) is a pseudo-polynomial time algorithm, and its time complexity is $O(n^2 \sum_{i=1}^{n} a_i)$.

Proof: Firstly, we consider the time complexity of solving the optimal continuous processing module. It takes $O(\sum_{i=1}^{n} a_i)$ time to search l in the interval $\left[\underline{l} \quad \overline{l}\right]$ and separately takes n time to search i,j in the interval $\left[1 \quad n\right]$. But m only has two cases. So solving the optimal continuous processing module need $O(n^2 \sum_{i=1}^{n} a_i)$ time by algorithm CB(C). In algorithm DP(C), the recursive formula needs $O(n \sum_{i=1}^{n} a_i)$ times of cyclic search, and each calculation of Eq. (5) requires $O(n)$ steps. Since algorithm DP(C) only uses the calculation results of algorithm CB(C) every time, but there is no nested loop, the time complexity of algorithm DP(C) is $O(n^2 \sum_{i=1}^{n} a_i)$. So Theorem 3.3 holds.

Since SHFSFC has a pseudo-polynomial time algorithm, SHFSFC is ordinary NP-hard. That is, the following theorem holds.

Theorem 3.4 Problem SHFSFC can be solved in $O(n^2 \sum_{i=1}^{n} a_i)$ time, which implies it is NP-hard in the ordinary sense.

4 A Dynamic Programming Algorithm of Problem SHFSFW

This section will construct a dynamic programming algorithm for problem SHFSFW using a method similar to Section 3. First, we construct the optimal continuous processing modules and calculate the objective value of every optimal continuous processing module by a dynamic programming algorithm. Then, we use optimal continuous processing modules to construct an optimal partial schedule. Finally, the objective of the optimal schedule is calculated by backward recursion. However, the goal of SHFSFW is to minimize the total weighted completion time, which is different from makespan. It has no direct relationship with the load of the jobs on machine M_1. So, we need to add a parameter h that represents the load of the jobs on M_1 for constructing the continuous processing module. So we use five elements m, i, j, h, l instead of four elements in Section 3 to structure the continuous processing module. The strict definition of the continuous processing module for SHFSFW is given below.

Definition 4.1 A sub-schedule consisting of a job subset $\{J_i, J_{i+1}, \cdots, J_j\}$ is called a continuous processing module of Problem SHFSFW if it satisfies the following conditions, denoted by five elements CPM(m, i, j, h, l).

(1) The first task of job J_iA_i is processed on M_m ($m = 1$ or 2);
(2) No matter on M_1 or M_2, there is no idle time between any two continuously processed jobs;
(3) The load generated by the subset $\{J_i, J_{i+1}, \cdots, J_j\}$ on M_1 is equal to h;
(4) The difference between the complete time of subset $\{J_i, J_{i+1}, \cdots, J_j\}$ on M_1 and M_2 is equal to l.

It is easy to see that CPM(m, i, j, h, l) can be structured by CPM($m, i, j-1, h', l'$) using two different methods (as shown in Fig. 5). The first method to get CPM(m, i, j, h, l) is to add job $J_j \in V_1$to CPM(m, i, $j-1$, h', l') (see Fig. 5a); The second method to get CPM(m, i, j, h, l) is to add job $J_j \in V_2$ to CPM(m, i, $j-1$, h', l') (see Fig. 5b). We assume that $f(m,i,j,h,l)$ is the minimum total weighted completion time of CPM(m, i, j, h, l) on machine M_1 composed of job subset $\{J_i, J_{i+1}, \cdots, J_j\}$. Using the discussion similar to Section 3, we can get the dynamic programming algorithm about $f(m,i,j,h,l)$ as follows.

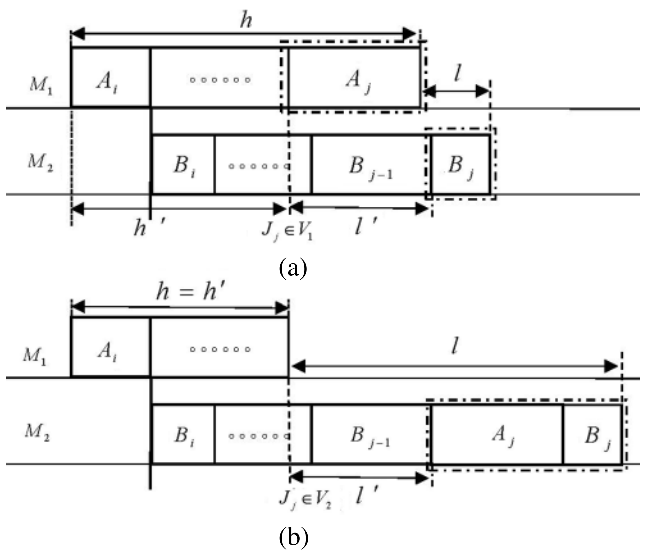

Figure 5: The two ways from CPM(1, $i, j-1, h', l'$) to CPM(1, i, j, h, l) as shown in (a) and (b)

Dynamic Programming Algorithm CPM(WC):

Initial Conditions:

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

Recurrence Relation:

The parameters m,i,j,h,l respectively satisfy $m\in\{1,2\}$, $1\le i<j\le n$, $0\le h\le\sum_{t=i}^{j}a_t$, $\underline{l}\le l\le\overline{l}$.

(1) When A_j is processed on M_1 (see Fig. 5a),

$$f_1=f(m,i,j-1,h-a_j,l+a_j-b_j)+w_j(h+l);$$

(2) When A_j is processed on M_2 (see Fig. 5b),

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

Recurrence Formula: $f(m,i,j,h,l)=\min\{f_1,\ \ f_2\}$.

Next, we define the Partial Schedule (m,i) of SHFSFW. Let Partial Schedule (m,i) be a sub-schedule of the job set $\{J_i, J_{i+1}, \cdots, J_n\}$, denoted by PS$(m,i)$, where the first job $J_i\in V_m$, $m\in\{1,2\}$. Then let $g(m,i)$ be the minimum total weighted completion time of PS(m,i). Next, we give a dynamic programming algorithm to calculate $g(m,i)$. For the convenience of narration, we construct a virtual job J_{n+1} with processing time $a_{n+1}=+\infty$, $b_{n+1}=+\infty$.

Dynamic Programming Algorithm DP(WC):

Initial Conditions:

$g(m,n+1)=0,\ \ m=1,2;$

Recurrence Relation:

The parameters m,i respectively satisfy $m\in\{1,2\},\ 1\le i\le n$.

$$g=\begin{cases} f(m,i,j,h,l)+g(1,j+1)+(\sum\limits_{t=j+1}^{n} w_t)h & \text{when } l<a_{j+1}; \\ +\infty & \text{otherwise.}\end{cases} \tag{6}$$

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

Target Value: $\min\sum w_iC_i=\min_{m\in\{1,2\}}\{g(m,1)\}$.

The initial conditions, recurrence formula, and target value in dynamic programming algorithm DP(WC) are clearly valid. So we also mainly analyze the recurrence relation below. Once the optimal CPM(m,i,j,h,l) is given, PS(m,i) can be obtained through the optimal PS$(m,j+1)$. For there is an idle time between CPM(m,i,j,h,l) and PS$(m,j+1)$, according to Proposition 2.1, we have m is equal to 1 in PS$(m,j+1)$. We can easily have $g(m,i)=f(m,i,j,h,l)+g(1,j+1)+(\sum\limits_{t=j+1}^{n} w_t)h$. According that there is an idle time between J_j and J_{j+1}, $l<a_{j+1}$ must be satisfied. So Eq. (6) holds.

Through the analysis similar to Theorem 3.3 in Section 3, we can obtain the following theorem on the time complexity of dynamic programming algorithm DP(WC).

Theorem 4.2 Dynamic programming algorithm DP(WC) is a pseudo-polynomial time algorithm, and its time complexity is $O(n^2(\sum_{i=1}^{n}a_i)^2)$.

Similar to Theorem 3.4, we can have the following theorem.

Theorem 4.3 Problem SHFSFC can be solved in $O(n^2(\sum_{i=1}^{n}a_i)^2)$ time, which implies it is NP-hard in the ordinary sense.

5 Analysis of Algorithm Effect Based on Numerical Experiments

5.1 Time Complexity Analysis

We analyze the computational complexity of two problems, design dynamic programming algorithms for them, and calculate the time complexity of the algorithms in Sections 2–4. These results are summarized in Table 1.

Table 1: The detailed results of the two models

Objective	Complexity	Time complexity	Remark
$C_{\max}$	Ordinary NP-hard	$O(n^2(\sum_{i=1}^{n}a_i))$	Theorem 2.3
$\sum w_iC_i$	Ordinary NP-hard	$O(n^2(\sum_{i=1}^{n}a_i)^2)$	Theorem 3.2

We took SHFSFC as an example to test the operation efficiency of the dynamic programming algorithms designed in this paper through numerical experiments. The numerical experiments were carried out in Matlab R2017 on a laptop with built-in an Intel Core i5 8250u CPU, 8 GB LPDDR3 RAM, and Windows 10. The processing time of each flexible task a_i was obtained by a random partition of a given number $\sum_{i=1}^{n} a_i$ into n values. The processing time of b_i was produced as a uniformly distributed random number within [0,10]. The experiments were performed for $10 \le n \le 230$ with an interval of 20 and $100 \le \sum_{i=1}^{n} a_i \le 1000$ with an interval of 100. We conducted a total of $12 \times 10 = 120$ experiments with different combinations of n and $\sum_{i=1}^{n} a_i$. And we generated 30 random test instances for each experiment. The average running results of all experiments are listed in Table 2, where the top row represents $\sum_{i=1}^{n} a_i$, the leftmost column represents n, and the unit of the data in the table is in seconds.

It can be seen from Table 2 that even if n reaches 230 and the total processing time of all flexible tasks reaches 1000, the time used by algorithm DP(C) only needs less than 85 s. Therefore, we can conclude that the actual effect of the algorithm is entirely acceptable.

Table 2: The average running times (in seconds) for $10 \le n \le 230$ 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
10	0.03	0.06	0.09	0.13	0.15	0.18	0.30	0.34	0.47	0.70
30	0.16	0.37	0.58	0.81	1.03	1.26	1.56	1.77	2.11	2.52
50	0.36	0.86	1.37	1.90	2.43	2.97	3.57	4.05	4.72	5.43
70	0.64	1.53	2.44	3.39	4.35	5.33	6.34	7.19	8.32	9.44
90	0.99	2.37	3.81	5.28	6.79	8.34	9.87	11.18	12.89	14.53
110	1.41	3.40	5.46	7.58	9.75	11.99	14.16	16.02	18.44	20.72
130	1.91	4.61	7.40	10.28	13.24	16.28	19.20	21.71	24.98	28.00
150	2.48	6.00	9.64	13.39	17.25	21.22	24.99	28.25	32.49	36.38
170	3.13	7.57	12.16	16.90	21.78	26.80	31.54	35.64	40.99	45.84
190	3.85	9.32	14.98	20.82	26.83	33.02	38.85	43.89	50.46	56.40
210	4.64	11.25	18.08	25.14	32.40	39.89	46.92	52.99	60.91	68.05
230	5.51	13.36	21.47	29.86	38.50	47.41	55.74	62.94	72.35	80.79

Using the data in Table 2, we can discuss the relationship between the running time of the algorithm and the total processing time of all flexible tasks when there are fewer jobs ($n = 10$), a lot of jobs ($n = 110$), and a large number of jobs ($n = 210$) (as shown in Fig. 6). It can be got from Fig. 6 that: (1) The running time of algorithm DP(C) increases with the increase of the total processing time of all flexible tasks $\sum_{i=1}^{n} a_i$; (2) The more the number of jobs, the faster the running time of the algorithm increases with the increase of $\sum_{i=1}^{n} a_i$; (3) When n is given, the growth rate of the algorithm processing time is similar, i.e., the running time of the algorithm is nearly linear with $\sum_{i=1}^{n} a_i$.

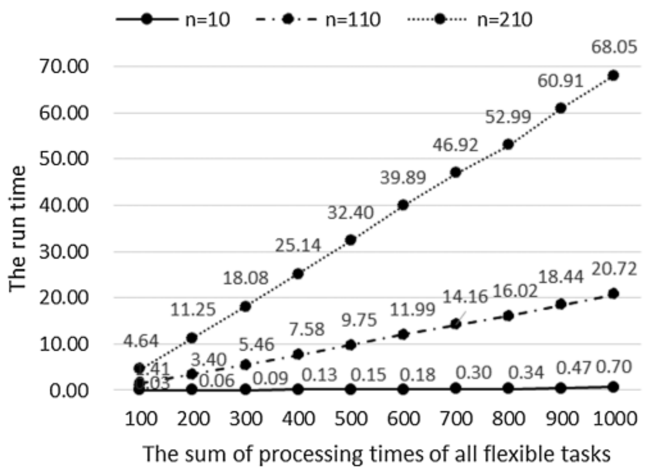

Figure 6: When $n = 10$, 110, 210, the running time changes with $\sum_{i=1}^{n} a_i$

Similarly, the data in Table 2 is used to analyze the relationship between the running time of the algorithm and the number of jobs when the total processing time of all flexible tasks is small ($\sum_{i=1}^{n} a_i = 100$), large ($\sum_{i=1}^{n} a_i = 500$), and huge ($\sum_{i=1}^{n} a_i = 1000$) (as shown in Fig. 7). It can be got from Fig. 7 that: (1) The running time of the algorithm increases with the increase of the number of all jobs; (2) The growth rate of the algorithm running time becomes faster with the increase of the number of all jobs; (3) The larger the total processing time of all flexible tasks, the faster the algorithm processing time increases with the increase of the number of all jobs.

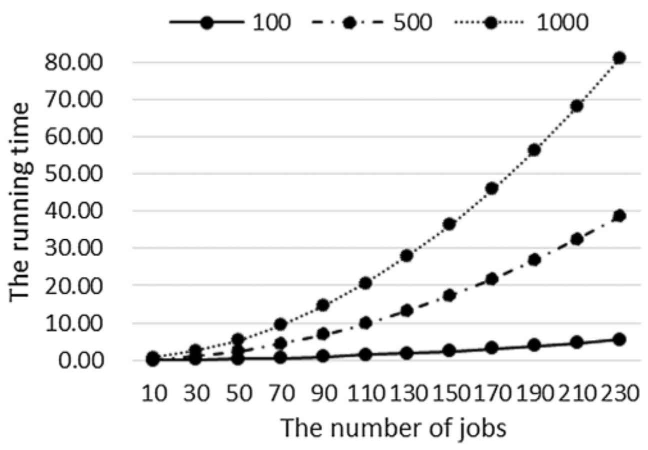

Figure 7: When $\sum_{i=1}^{n} a_i$=100, 500, 1000, the running time changes with n

5.2 *Comparison with Other Commonly Used Algorithms*

Since there is no other algorithm for the two-stage hybrid flow-shop problem studied in this paper, we compare the high-quality algorithms for a similar problem with the dynamic programming algorithms given in this paper to analyze the effect of our algorithms.

We usually design polynomial-time optimal algorithms to solve P problems [30]. But there exist two kinds of algorithms for solving NP-hard problems. The first one is the exact algorithm, which gives the optimal solution, but the time complexity is exponential, including the enumeration algorithm, branch and bound algorithm [31]. The second one is the heuristic algorithm, which usually gives the approximate solution of the problem, but the time complexity is polynomial, including the genetic algorithm [20,32], (Iterated) green algorithm [33], fireworks algorithm [34], artistic neural network algorithm [35], (discrete) harmony search algorithm [36], etc. These algorithms are widely used in solving various scientific and engineering problems. From the analysis of the existing literature, it is found that the branch and bound algorithm is the most commonly used and relatively practical algorithm in the accurate solution of hybrid flow-shop problems [31,37]. Regarding the heuristic algorithm, the research in literature [36] shows that a discrete harmony search algorithm has advantages compared with a genetic algorithm, greedy algorithm, and harmony search algorithm, both in running time and the average relative percentage deviation, in solving hybrid flow-shop problems. The average Relative Percentage Deviation (denoted by RPD) is usually used to measure the approximation of heuristic algorithms [38], and its specific definition is as follows: $RPD = \frac{C_A - C^*}{C^*} \times 100\%$, in which C_A is the objective value calculated by Algorithm A and C^* is the optimal objective value which can be got by the exact algorithm.

We still took problem SHFSFC as an example to compare dynamic programming algorithm DP(C) (the algorithm given by us), the branch and bound algorithm (given by Moursli et al. [31], denoted by B&B), and the discrete harmony search algorithm (given by Zini et al. [36], denoted by DHS) regarding the running time and the average relative percentage deviation.

The numerical experiments were carried out in the same software and hardware environment as in Section 5.1. The time complexity of B&B and DHS is independent of $\sum_{i=1}^{n} a_i$ but only related to the number of jobs. So, we only conducted grouping experiments according to n to unify the comparison standards. We considered the efficiency (running time) and performance (RPD) of the three algorithms in the small-scale-jobs case ($5 \le n \le 50$) and the large-scale-jobs case ($100 \le n \le 300$). The numerical experiments were carried out for $5 \le n \le 50$ with an interval of 5 in the small-scale-jobs case and $100 \le n \le 300$ with an interval of 50 in the large-scale-jobs case. The processing time of each task a_i (or b_i) was produced as a uniformly distributed random number within [0,10]. Twenty random instances for every experiment were produced. The average values of running time and RPDs were selected as the experimental results. The average running time of each algorithm under the two experimental conditions is shown in Table 3.

Table 3: The average running time (in seconds) of DP(C), B&B, and DHS for two kinds of experiments

	Small-scale-jobs experiments										Large-scale-jobs experiments				
n	5	10	15	20	25	30	35	40	45	50	100	150	200	250	300
DP(C)	0.09	0.24	0.43	0.66	0.92	1.22	1.55	1.92	2.32	2.77	9.18	19.21	32.86	54.14	88.03
B&B	0.04	1.19	2.73	5.16	9.31	18	43	129	466^a(6)^b	980^a(14)^b	(20)^b	(20)^b	(20)^b	(20)^b	(20)^b
DHS	0.01	0.14	0.25	0.36	0.47	0.67	0.79	0.93	1.26	1.49	4.86	7.69	10.64	13.68	17.55

Notes: a: The average running time of the instances completed in 1200 s; b: The number of instances that cannot be completed in 1200 s.

As shown in Table 3, DP(C) and DHS can complete the operation in 100 s in both two cases. In particular, DHS can be completed in 20 s, showing its advantage in time complexity. However, for B&B, when the number of jobs reaches 45, some instances cannot be completed in 1200 s; when the number of jobs reaches 50%, 70% of the instances cannot be completed in 1200 s; and all large-scale-jobs experiments cannot be completed in 1200 s.

The relationship between running time and *n* of each algorithm in the small-scale-jobs case is shown in Fig. 8. For this case, we can get that: (1) DHS has the most advantage in time complexity, followed by DP(C), while B&B has obvious disadvantages; (2) The running time of each algorithm increases with the increase of the number of jobs and one of B&B increases much faster than the other two algorithms.

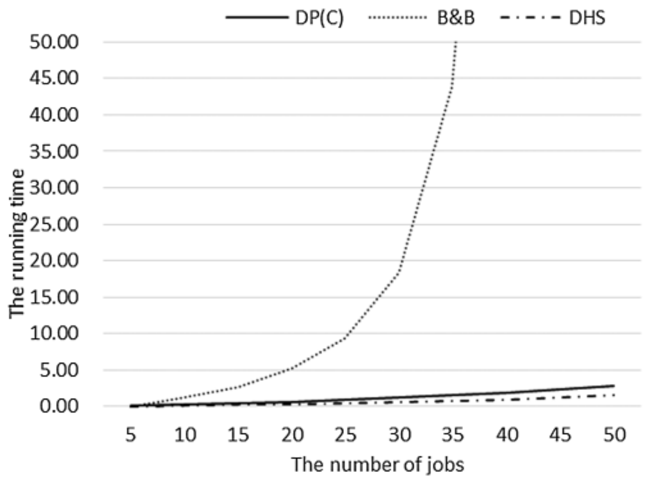

Figure 8: The average running time of DP(C), B&B, and DHS for small-scale-jobs experiments

Similarly, using the data in Table 3, the relationship between running time and *n* of each algorithm in the large-scale-jobs case is shown in Fig. 9. Since B&B cannot complete the operation within 1,200 s, only DHS and DP(C) are compared here. For this case, we can get that: (1) DHS has obvious advantages in time complexity, and the running time of each algorithm is within the acceptable range; (2) With the increase of *n*, the running time of each algorithm increases. The growth rate of the running time of DP(C) increases with the increase of *n*, while the growth rate of DHS is relatively flat.

Table 4 shows the results of the three algorithms on the relative percentage deviation under small-scale-jobs experiments and large-scale-jobs experiments.

Using the data in Table 4, we plot the relationship between the average RPD of algorithm DHS and the number of jobs in Fig. 10.

From Fig. 10, we can get that: (1) Since DP(C) and B&B are both exact algorithms, their average RPDs are 0. The average RPD of DHS is large in the small-scale-jobs case and decreases rapidly with the increase of the number of jobs, which means that the accuracy of DHS increases quickly with the rise of the number of jobs. However, in the small-scale-jobs case, the difference between the solution of DHS and the optimal solution is more than 25%, especially when the number of jobs is less than 15, the error is more than 50%; (2) In the large-scale-jobs case, with the increase of *n*, the reduction speed of the average RPD slows down. In all large-scale-jobs experiments, the average RPD is above 0.1. Even if n reaches 300, the gap between it and the optimal solution is still 12%.

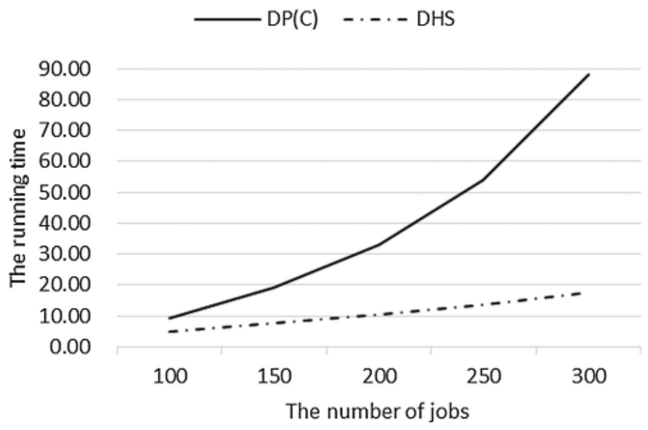

Figure 9: The average running time of DP(C) and DHS for large-scale-jobs experiments

Table 4: The average RPDs of DP(C), B&B, and DHS for two kinds of instances

	Small-scale-jobs experiments										Large-scale-jobs experiments				
n	5	10	15	20	25	30	35	40	45	50	100	150	200	250	300
DP(C)	0[a]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B&B	0[a]	0	0	0	0	0	0	0	0	0	/[b]	/	/	/	/
DHS	0.83	0.61	0.53	0.39	0.36	0.33	0.30	0.28	0.26	0.25	0.16	0.14	0.13	0.13	0.12

Notes: a: Since DP(C) and B&B are both exact algorithms, their RPDs are all 0; b: All instances cannot be completed in 1200 s by B&B.

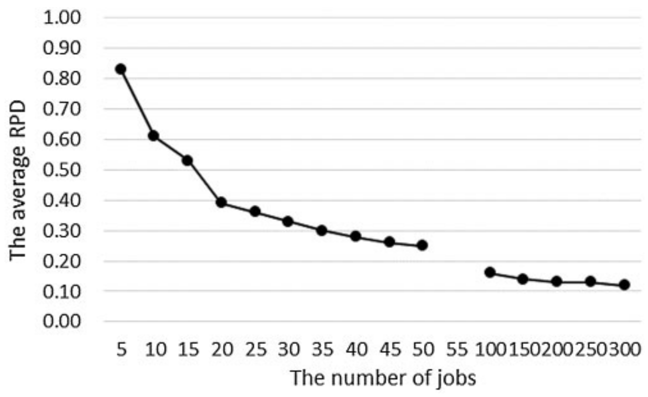

Figure 10: The average RPD of DHS changes with the number of jobs for two kinds of experiments

5.3 *Summary of Comparison*

According to the comparative analysis of numerical experimental results in Sections 5.1 and 5.2, we can get the following results:

(1) From the perspective of algorithm time complexity, DHS has advantages, followed by DP(C). The gap between the two can be ignored in the small-scale-jobs case. Although there is a certain gap in the large-scale-jobs case, DP(C) is still within the acceptable range. B&B has significant disadvantages in running time, and when the number of jobs is greater than 45, its running time becomes unacceptable;
(2) From the perspective of algorithm accuracy, DP(C) and B&B are both exact algorithms so that they can give the optimal solution. But B&B cannot provide the optimal solution within the time limit when the number of jobs is immense. DHS is a heuristic algorithm. In the small-scale-jobs case, the accuracy is low, and the gap with the optimal solution is more than 25%. In the large-scale-jobs case, the accuracy is improved, but the gap with the optimal solution is still more than 12%;
(3) The running time of DP(C) is small, and the accuracy is better than DHS 50% in the small-scale-jobs case. Although there is a certain gap between DP(C) and DHS, the running time of DP(C) is still in the acceptable range. And the algorithm accuracy is better than DHS 12% in the large-scale-jobs case.

6 Conclusions

This paper studies two two-stage hybrid flow-shop problems with two machines and fixed processing sequences, widely used in shared manufacturing, stock control systems, and other manufacturing areas. Two objectives of minimizing makespan and total weighted completion time are considered. For these two models, we first show they are both ordinary NP-hard, present a dynamic programming algorithm for each model, and analyze the time complexity of each algorithm. Then, the relationship between the running time and the combination of the total processing time of the flexible tasks and the number of jobs is obtained by numerical experiments. Finally, the advantages and disadvantages of the dynamic programming algorithms presented in this paper are compared with the exact algorithm B&B and the heuristic algorithm DHS, which have advantages in solving hybrid flow-shop problems.

In future research, we will study the polynomial-time approximation algorithm with a smaller worst-case ratio for this problem and extend the dynamic programming algorithm's design method given in this paper to other scheduling problems with fixed processing sequences.

Acknowledgement: This work was partially supported by the Zhejiang Provincial Philosophy and Social Science Program of China (Grant No. 19NDJC093YB), the National Social Science Foundation of China (Grant No. 19BGL001), the Natural Science Foundation of Zhejiang Province of China (Grant No. LY19A010002) and the Natural Science Foundation of Ningbo of China (The design of algorithms and cost-sharing rules for scheduling problems in shared manufacturing, Acceptance No. 20211JCGY010241).

Funding Statement: This study was funded by the Zhejiang Provincial Philosophy and Social Science Program of China (Grant No. 19NDJC093YB), Recipient: Qi Wei. And the National Social Science Foundation of China (Grant No. 19BGL001), Recipient: Qi Wei. And the Natural Science Foundation of Ningbo of China (Acceptance No. 20211JCGY010241), Recipient: Qi Wei.

And the Natural Science Foundation of Zhejiang Province of China (Grant No. LY19A010002), Recipient: Yong Wu.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Vairaktarakis, G., Lee, C. Y. (2004). Analysis of algorithms for two-stage flowshops with multi-processor task flexibility. *Naval Research Logistics, 51,* 44–59. DOI 10.1002/nav.10104.
2. Wei, Q., He, Y. (2005). A two-stage semi-hybrid flowshop problem in graphics processing. *Applied Mathematics: A Journal of Chinese Universities B, 20,* 393–400. DOI 10.1007/s11766-005-0016-6.
3. Zhong, W. Y., Shi, Y. (2018). Two-stage no-wait hybrid flowshop scheduling with inter-stage flexibility. *Journal of Combinatorial Optimization, 35(1),* 108–125. DOI 10.1007/s10878-017-0155-8.
4. Li, Y., Li, X., Gao, L. (2020). Review on hybrid flow shop scheduling problems. *China Mechanical Engineering, 31(23),* 2798. DOI 10.3969/j.issn.1004-132X.2020.23.004.
5. 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, 2011*, *111(5),* 208–212. DOI 10.1016/j.ipl.2010.11.021.
6. Yu, C., Xu, X., Yu, S., Sang, Z., Yang, C. et al. (2020). Shared manufacturing in the sharing economy: Concept, definition and service operations. *Computers & Industrial Engineering, 146,* 106602. DOI 10.1016/j.cie.2020.106602.
7. Jiang, Y. W., Wei, Q. (2011). An improved algorithm for a hybrid flow-shop problem in graphics processing. *Acta Automatica Sinica, 37(11),* 1381–1386. DOI 10.3724/SP.J.1004.2011.01381.
8. Wei, Q., Wu, Y. (2020). Dynamic programming algorithms for two-machine hybrid flow-shop scheduling with a given job sequence and deadline. *IEEE Access, 8,* 89964–89975. DOI 10.1109/ACCESS.2020.2993857.
9. Zhang, M., Lan, Y., Han, X. (2020). Approximation algorithms for two-stage flexible flow shop scheduling. *Journal of Combinatorial Optimization, 39(1),* 1–14. DOI 10.1007/s10878-019-00449-3.
10. Peng, A. Z., Liu, L. C., Lin, W. F. (2021). Improved approximation algorithms for two-stage flexible flow shop scheduling. *Journal of Combinatorial Optimization, 41(1),* 28–42. DOI 10.1007/s10878-020-00657-2.
11. Wang, S., Kurz, M., Mason, S. J., Rashidi, E. (2019). Two-stage hybrid flow shop batching and lot streaming with variable sublots and sequence-dependent setups. *International Journal of Production Research, 57(22),* 6893–6907. DOI 10.1080/00207543.2019.1571251.
12. Tan, Y., Monch, L., Fowler, J. W. (2017). A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. *Journal of Scheduling, 21(2),* 209–226. DOI 10.1007/s10951-017-0530-4.
13. Dong, J., Pan, H., Ye, C., Tong, W., Hu, J. (2020). No-wait two-stage flowshop problem with multi-task flexibility of the first machine. *Information Sciences, 544,* 25–38. DOI 10.1016/j.ins.2020.06.052.
14. Wang, S., Wang, X., Yu, L. (2020). Two-stage no-wait hybrid flow-shop scheduling with sequence-dependent setup times. *International Journal of Systems Science: Operations & Logistics, 7(3),* 291–307. DOI 10.1080/23302674.2019.1575997.
15. Shao, W., Shao, Z., Pi, D. (2021). Effective constructive heuristics for distributed no-wait flexible flow shop scheduling problem. *Computers & Operations Research, 136,* 105482. DOI 10.1016/j.cor.2021.105482.
16. Feng, X., Zheng, F., Xu, Y. (2016). Robust scheduling of a two-stage hybrid flow shop with uncertain interval processing times. *International Journal of Production Research, 54(12),* 1–12. DOI 10.1080/00207543.2016.1162341.
17. Ahonen, H., Alvarenga, A. G. (2017). Scheduling flexible flow shop with recirculation and machine sequence-dependent processing times: Formulation and solution procedures. *The International Journal of Advanced Manufacturing Technology, 89(1–4),* 765–777. DOI 10.1007/s00170-016-9093-3.
18. Lei, D., Wang, T. (2020). Solving distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memeplex grouping. *Engineering Optimization, 52(9),* 1461–1474. DOI 10.1080/0305215X.2019.1674295.

19. Cai, J., Zhou, R., Lei, D. (2020). Fuzzy distributed two-stage hybrid flow shop scheduling problem with setup time: Collaborative variable search. *Journal of Intelligent & Fuzzy Systems, 38(3),* 3189–3199. DOI 10.3233/JIFS-191175.
20. Rolf, B., Reggelin, T., Nahhas, A., Müller, M., Lang, S. (2020). Scheduling jobs in a two-stage hybrid flow shop with a simulation-based genetic algorithm and standard dispatching rules. *Winter Simulation Conference*, pp. 1584–1595. Florida, USA.
21. Wang, S., Wang, X., Chu, F., Yu, J. B. (2020). An energy-efficient two-stage hybrid flow shop scheduling problem in a glass production. *International Journal of Production Research, 58(8),* 2283–2314. DOI 10.1080/00207543.2019.1624857.
22. 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(7),* 705–731. DOI 10.1002/(SICI)1520-6750(199810)45:73.0.CO;2-F.
23. Hwang, F. J., Kovalyov, M. Y., Lin, B. M. T. (2012). Total completion time minimization in two-machine flow shop scheduling problems with a fixed job sequence. *Discrete Optimization, 9(1),* 29–39. DOI 10.1016/j.disopt.2011.11.001.
24. Hwang, F. J., Kovalyov, M. Y., Lin, B. M. T. (2014). Scheduling for fabrication and assembly in a two-machine flowshop with a fixed job sequence. *Annals of Operations Research, 217(1),* 263–279. DOI 10.1007/s10479-014-1531-8.
25. Lin, B. M. T., Hwang, F. J., Kononov, A. V. (2016). Relocation scheduling subject to fixed processing sequences. *Journal of Scheduling, 19(2),* 153–163. DOI 10.1007/s10951-015-0455-8.
26. Halman, N., Vinetz, U. (2021). An FPTAS for two performance measures for the relocation scheduling problem subject to fixed processing sequences. *Optimization Letters, 2021,* 1–16. DOI 10.1007/s11590-021-01772-7.
27. Cheref, A., Agnetis, A., Artigues, C., Billaut, J. C. (2017). Complexity results for an integrated single machine scheduling and outbound delivery problem with fixed sequence. *Journal of Scheduling, 20(6),* 681–693. DOI 10.1007/s10951-017-0540-2.
28. Cheng, T. C. E., Kravchenko, S. A., Lin, B. M. T. (2019). Server scheduling on parallel dedicated machines with fixed job sequences. *Naval Research Logistics, 66(4),* 321–332. DOI 10.1002/nav.21846.
29. Cheng, T. C. E., Kravchenko, S. A., Lin, B. M. T. (2020). Complexity of server scheduling on parallel dedicated machines subject to fixed job sequences. *Journal of the Operational Research Society, 2020,* 1–4. DOI 10.1080/01605682.2020.1779625.
30. Ji, M., Zhang, W. Y., Liao, L. J., Cheng, T. C. E., Tan, Y. Y. (2019). Multitasking parallel-machine scheduling with machine-dependent slack due-window assignment. *International Journal of Production Research, 57(6),* 1667–1684. DOI 10.1080/00207543.2018.1497312.
31. Moursli, O., Pochet, Y. (2000). A branch-and-bound algorithm for the hybrid flowshop. *International Journal of Production Economics, 64,* 113–125. DOI 10.1016/S0925-5273(99)00051-1.
32. Nirmal Kumar, S. J., Ravimaran, S., Alam, M. M. (2020). An effective non-commutative encryption approach with optimized genetic algorithm for ensuring data protection in cloud computing. *Computer Modeling in Engineering & Sciences, 125(2),* 671–697. DOI 10.32604/cmes.2020.09361.
33. Shao, W., Shao, Z., Pi, D. (2020). Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. *Knowledge-Based Systems, 194,* 105527. DOI 10.1016/j.knosys.2020.105527.
34. Pang, X., Xue, H., Tseng, M. L., Lim, M. K., Liu, K. (2020). Hybrid flow shop scheduling problems using improved fireworks algorithm for permutation. *Applied Sciences, 10(3),* 1174. DOI 10.3390/app10031174.
35. Karaci, A., Yaprak, H., Ozkaraca, O., Demir, I., Simsek, O. (2019). Estimating the properties of ground-waste-brick mortars using DNN and ANN. *Computer Modeling in Engineering & Sciences, 118(1),* 207–228. DOI 10.31614/cmes.2019.04216.

36. Zini, H., Elbernoussi, S. (2017). Minimizing makespan in hybrid flow shop scheduling with multiprocessor task problems using a discrete harmony search. *IEEE International Conference on Computational Intelligence & Virtual Environments for Measurement Systems & Applications*, pp. 177–180. Annecy, France.

37. Wang, S., Liu, M., Chu, C. (2015). A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling. *International Journal of Production Research, 53(4),* 1143–1167. DOI 10.1080/00207543.2014.949363.

38. Priya, A., Sahana, S. K. (2020). Multiprocessor scheduling based on evolutionary technique for solving permutation flow shop problem. *IEEE Access, 8,* 53151–53161. DOI 10.1109/ACCESS.2020.2973575.