Theoretical Computer Science 734 (2018) 24–31

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

An approximation scheme for minimizing the makespan of the parallel identical multi-stage flow-shops ☆

Weitian Tong [a,b], Eiji Miyano [c], Randy Goebel [b], Guohui Lin [b,*]

[a] Department of Computer Sciences, Georgia Southern University, Statesboro, GA 30460, USA
[b] Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada
[c] Department of Systems Design and Informatics, Kyushu Institute of Technology, Iizuka, Fukuoka 820-8502, Japan

ARTICLE INFO

Article history:
Received 5 August 2016
Received in revised form 7 July 2017
Accepted 21 September 2017
Available online 28 September 2017

Keywords:
Multiprocessor scheduling
Flow-shop scheduling
Makespan
Linear program
Polynomial-time approximation scheme

ABSTRACT

In the *parallel k-stage flow-shops* problem, we are given m identical k-stage flow-shops and a set of jobs. Each job can be processed by any one of the flow-shops but switching between flow-shops is not allowed. The objective is to minimize the makespan, which is the finishing time of the last job. This problem generalizes the classical parallel identical machine scheduling (where $k = 1$) and the classical flow-shop scheduling (where $m = 1$) problems, and thus it is NP-hard. We present a polynomial-time approximation scheme (PTAS) for the problem, when m and k are fixed constants. The key technique is to partition the jobs into *big* jobs and *small* jobs, enumerate over all feasible schedules for the big jobs, and handle the small jobs by solving a linear program and employing a "sliding" method. Such a technique has been used in the design of PTAS for several flow-shop scheduling variants. Our main contributions are the non-trivial application of this technique and a valid answer to the open question in the literature.

1. Introduction

In the *parallel k-stage flow-shop* problem, we are given m parallel identical k-stage flow-shops $F_1, F_2, \ldots, F_m$ and a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. These k-stage flow-shops are the *classic* flow-shops, each contains exactly one machine at every stage, *i.e.*, k sequential machines. Every job has k operations, and it can be assigned to exactly one of the m flow-shops for processing; once it is assigned to the flow-shop, its k operations are then respectively processed on the k sequential machines in the flow-shop. The goal is to minimize the makespan, which is the completion time of the last job. We denote the problem for simplicity as (m, k)-PFS. Let $M_{\ell,1}, M_{\ell,2}, \ldots, M_{\ell,k}$ denote the k sequential machines in the flow-shop F_ℓ, for every ℓ. The job J_i is represented as a k-tuple $(p_{i,1}, p_{i,2}, \ldots, p_{i,k})$, where $p_{i,j}$ is the processing time for the j-th operation, that is, J_i needs to be processed non-preemptively on the j-th machine in the flow-shop to which the job is assigned. For all i and j, the processing time $p_{i,j}$ is a non-negative real number.

It is clear to see that, when $m = 1$, the (m, k)-PFS problem is the classic *flow-shop scheduling* [5] (a k-stage flow-shop); when $k = 1$, the (m, k)-PFS problem is the classic *multiprocessor scheduling* [5] (m parallel identical machines). When the two-stage flow-shops are involved, *i.e.*, $k = 2$, the $(m, 2)$-PFS problem has been previously studied in [13,25,28,4]. Here

☆ An extended abstract appears in the Proceedings of FAW 2016.

* Corresponding author.
E-mail addresses: wtong@georgiasouthern.edu (W. Tong), miyano@ces.kyutech.ac.jp (E. Miyano), rgoebel@ualberta.ca (R. Goebel), guohui@ualberta.ca (G. Lin).

https://doi.org/10.1016/j.tcs.2017.09.018
0304-3975/

we first review the complexity and the approximation algorithms for the flow-shop scheduling and the multiprocessor scheduling problems.

For the k-stage flow-shop problem, it is known that when $k = 2$ or 3, there exists an optimal schedule that is a *permutation schedule*, in which the jobs are processed on all the k machines in the same order; but when $k \geq 4$, it is shown [3] that there may exist no optimal schedule that is a permutation schedule. Johnson [18] presented an $O(n \log n)$-time algorithm for the two-stage flow-shop problem, where n is the number of jobs; the k-stage flow-shop problem becomes *strongly* NP-hard when $k \geq 3$ [6]. After several efforts [18,6,7,2], Hall [12] designed a polynomial-time approximation scheme (PTAS) for the k-stage flow-shop problem, for any fixed constant $k \geq 3$. Due to the strong NP-hardness, such a PTAS is the best possible unless P = NP. When k is a part of the input (*i.e.*, an arbitrary integer), Williamson et al. [27] showed that the flow-shop scheduling cannot be approximated within 1.25; nevertheless, it remains unknown whether this case is APX-complete, that is, whether the problem admits a constant ratio approximation algorithm.

Note that the m-parallel identical machine scheduling problem is NP-hard when $m \geq 2$ [5]. When m is a fixed integer, the problem admits a pseudo-polynomial time exact algorithm [5], and Sahni [23] showed that this exact algorithm can be used to construct a *fully* PTAS (FPTAS); when m is a part of the input, the problem becomes strongly NP-hard, but still admits a PTAS by Hochbaum and Shmoys [14].[1] The *list-scheduling* algorithm by Graham [8] is a $(2 - 1/m)$-approximation, for arbitrary m.

The APX-hardness of the classic k-stage flow-shop problem when k is a part of the input implies the APX-hardness of the (m, k)-PFS problem when k is a part of the input. When k is a fixed integer, the (m, k)-PFS problem could admit a PTAS; however, since the classic k-stage flow-shop problem is strongly NP-hard for a fixed $k \geq 3$, the (m, k)-PFS problem would not admit an FPTAS unless P = NP. In this paper, we present a PTAS for the (m, k)-PFS problem when both k and m are fixed integers, which is the best possible approximability result. On the other hand, the (in-)approximability of the (m, k)-PFS problem when m is a part of the input while k is a fixed integer is left open.

Besides the (m, k)-PFS problem, another generalization of the flow-shop scheduling and the multiprocessor scheduling is the so-called *hybrid k-stage flow-shop* problem [20,22]. A hybrid k-stage flow-shop is a *flexible* flow-shop, that contains $m_j \geq 1$ parallel identical machines in the j-th stage, for $j = 1, 2, \ldots, k$. The problem is abbreviated as $(m_1, m_2, \ldots, m_k)$-HFS. A job J_i is again represented as a k-tuple $(p_{i,1}, p_{i,2}, \ldots, p_{i,k})$, where $p_{i,j}$ is the processing time for the j-th operation, which can be processed non-preemptively on any one of the m_j machines in the j-th stage. The objective of the $(m_1, m_2, \ldots, m_k)$-HFS problem is also to minimize the makespan. One clearly sees that when $m_1 = m_2 = \ldots = m_k = 1$, the problem reduces to the classic k-stage flow-shop problem; when $k = 1$, the problem reduces to the classic m-parallel identical machine scheduling problem.

As a toy example, suppose there is a set of three jobs, $\mathcal{J} = \{J_1 = (p_{1,1}, p_{1,2}, p_{1,3}), J_2 = (p_{2,1}, p_{2,2}, p_{2,3}), J_3 = (p_{3,1}, p_{3,2}, p_{3,3})\}$, that need to be processed. When we are provided with a $(2, 3)$-PFS (that is, two parallel identical 3-stage flow-shops), we may assign J_1 to the second flow-shop; then J_1 will be processed on the first machine of the second flow-shop for $p_{1,1}$ units of time, then on the second machine of the second flow-shop for $p_{1,2}$ units of time, and lastly on the third machine of the second flow-shop for $p_{1,3}$ units of time. On the other hand, if we are provided with a $(2, 1, 3)$-HFS, then we may process J_1 on any one of the two first-stage machines for $p_{1,1}$ units of time, then on the (only) second-stage machine for $p_{1,2}$ units of time, and lastly on any one of the three third-stage machine for $p_{1,3}$ units of time.

The literature on the hybrid k-stage flow-shop problem $(m_1, m_2, \ldots, m_k)$-HFS is also rich [20,22], especially on the hybrid two-stage flow-shop problem (m_1, m_2)-HFS. First, $(1, 1)$-HFS is the classic two-stage flow-shop problem which can be optimally solved in $O(n \log n)$ time [18], where n is the number of jobs. When $\max\{m_1, m_2\} \geq 2$, Hoogeveen et al. [15] showed that the (m_1, m_2)-HFS problem is strongly NP-hard. The special cases $(m_1, 1)$-HFS and $(1, m_2)$-HFS have attracted many researchers' attention [9,11,1,10]; the interested reader might refer to [26] for a survey on the hybrid two-stage flow-shop problem with a single machine in one stage.

For the general hybrid k-stage flow-shop problem $(m_1, m_2, \ldots, m_k)$-HFS, when all the m_1, m_2, ..., m_k are fixed integers, Hall [12] claimed that the PTAS for the classic k-stage flow-shop problem can be extended to a PTAS for the $(m_1, m_2, \ldots, m_k)$-HFS problem. Later, Schuurman and Woeginger [24] presented a PTAS for the hybrid two-stage flow-shop problem (m_1, m_2)-HFS, even when the numbers of machines m_1 and m_2 in the two stages are a part of the input. Jansen and Sviridenko [17] generalized this result to the hybrid k-stage flow-shop problem $(m_1, m_2, \ldots, m_k)$-HFS, where k is a fixed integer while $m_1, m_2, \ldots, m_k$ can be a part of the input. Due to the inapproximability of the classic k-stage flow-shop problem, when k is arbitrary, the $(m_1, m_2, \ldots, m_k)$-HFS problem cannot be approximated within 1.25 unless P = NP [27]. Table 1 summarizes the results we reviewed thus far.[2] In addition, there are plenty of heuristic algorithms in the literature for the general hybrid k-stage flow-shop problem, and the interested readers can refer to the survey by Ruiz et al. [22].

Compared to the rich literature on the hybrid k-stage flow-shop problem, the parallel k-stage flow-shop problem is much less studied. In fact, the general (m, k)-PFS problem is almost untouched, except only the two-stage flow-shops are involved [13,25,28,4]. He et al. [13] first studied the m parallel identical two-stage flow-shop problem $(m, 2)$-PFS, motivated

[1] We note that there are sequences of work in developing faster PTASes, which are not the intended subject in this paper. The interested readers might refer to [16] for major references.

[2] We do not list the detailed running time of these algorithms. Again, we note that there are sequences of work in developing faster PTASes, which are not the intended subject in this paper. The interested readers might refer to [16] for major references.

Table 1
Known results for the hybrid k-stage flow-shop problem.

		m_j machines in stage j		
		$m_j = 1$	m_j fixed	m_j arbitrary
k stages	$k = 1$	polynomial time	FPTAS [23]	PTAS [14]
	$k = 2$	polynomial time [18]	PTAS [12]	PTAS [24]
	$k \geq 3$ fixed	PTAS [12]	PTAS [12]	PTAS [17]
	k arbitrary	not be approximated within 1.25 [27]		

by an application from the glass industry. In their work, the $(m, 2)$-PFS problem is formulated as a mixed-integer programming and an efficient heuristic is proposed [13]. Vairaktarakis and Elhafsi [25] also studied the $(m, 2)$-PFS problem, in order to investigate the hybrid k-stage flow-shop problem. Among other results, Vairaktarakis and Elhafsi [25] observed that the $(2, 2)$-PFS problem can be broken down into two subproblems, a job partition problem and a classic two-stage flow-shop problem. Note that the second subproblem can be solved optimally by Johnson's algorithm [18]. The NP-hardness of the first subproblem [5] implies the NP-hardness of $(2, 2)$-PFS, simply by setting all $p_{i,2}$'s to zeros. One of the major contributions in [25] is an $O(nP^3)$-time dynamic programming algorithm for solving the NP-hard $(2, 2)$-PFS problem optimally, where n is the number of jobs and P is the sum of all processing times. (Qi [21] later improved the running time to $O(nP^2)$.) That is, the $(2, 2)$-PFS problem can be solved exactly in pseudo-polynomial time.

The NP-hardness of $(2, 2)$-PFS implies that the general $(m, 2)$-PFS problem is NP-hard, when either m is a part of the input (arbitrary) or m is a fixed integer greater than one. Zhang et al. [28] studied on how to approximate the $(m, 2)$-PFS problem, more precisely only for the special case where $m = 2$ or 3. They designed a 3/2-approximation algorithm when $m = 2$ and a 12/7-approximation algorithm when $m = 3$ [28]. Both algorithms are variations of Johnson's algorithm and the main idea is first to sort all the jobs using Johnson's algorithm into a sequence, then to cut this sequence into two (three, respectively) parts for the two (three, respectively) two-stage flow-shops in order to minimize the makespan. Recently, Dong et al. [4] extended the dynamic programming algorithm for the $(2, 2)$-PFS problem to solve the $(m, 2)$-PFS problem, for any fixed $m \geq 2$, in $O(nmP^{2m+1})$-time and $O(P^{2m})$-space. They then designed an FPTAS for the $(m, 2)$-PFS problem out of this exact pseudo-polynomial time algorithm.

Here we present a PTAS for the (m, k)-PFS problem when m and k are fixed integers. Our PTAS borrows some design ideas from the PTAS for the classic k-stage flow-shop problem by Hall [12]. The key technique is to partition the jobs into *big* jobs and *small* jobs, enumerate over all feasible schedules for the big jobs, and handle the small jobs by solving a linear program and employing a "sliding" method. Such a technique has been used in the design of PTAS for several flow-shop scheduling variants. Our main contributions are the non-trivial application of this technique and a valid answer to the open question proposed in [4].

2. An approximation scheme for the (m, k)-PFS Problem

In the sequel, a *schedule* for an instance of the (m, k)-PFS problem is an assignment of non-negative starting times to all the operations of the given jobs, each on one of the m flow-shops, and a *feasible* schedule is one in which the assignment meets the processing restrictions: 1) each job can have at most one of its operations undergoing processing at any point in time, 2) each operation of a job must be processed on a machine non-preemptively for the specified length of time, and 3) each machine can process at most one operation at any point in time. We use π^* to denote an optimal schedule and its makespan is denoted by OPT.

For ease of presentation, we let $P_i = \sum_{j=1}^{k} p_{ij}$ denote the total processing time of the job J_i over all k machines, and assume without loss of generality that $P_1 \geq P_2 \geq \ldots \geq P_n$; we also let $Q_j = \sum_{i=1}^{n} p_{ij}$ denote the total processing time of all the jobs in the j-th stage machines. Define $P = \sum_{i=1}^{n} P_i = \sum_{j=1}^{k} Q_j$. The following lemma bounds OPT.

Lemma 1. *We have the following upper and lower bounds on OPT:*

$$\max\left\{\frac{P}{mk}, P_1\right\} \leq OPT \leq \frac{P}{m} + P_1.$$

Proof. Recall that in a k-stage flow-shop, a job is first processed on the first machine, and when it is finished it can start to be processed on the second machine, and so on. Therefore, the optimal makespan must be greater than or equal to P_1, the longest total processing time of a job, that is, $\text{OPT} \geq P_1$. On the other hand, $\text{OPT} \geq \frac{Q_j}{m}$ for every j, which is the average processing time on the m machines in the j-th stage. This gives $\text{OPT} \geq \frac{1}{k}\sum_{j=1}^{k}\frac{Q_j}{m} = \frac{P}{mk}$.

For the upper bound, we add an extra constraint on the job processing: that every flow-shop must complete a job (*i.e.*, finish processing on all the k machines) before starting processing another job. This essentially constructs an instance of the m parallel identical machine scheduling problem to minimize the makespan, where the job J_i has processing time P_i. The *list scheduling* algorithm by Graham [8] produces a schedule π with makespan $C^{\pi} \leq \frac{P}{m} + (1 - \frac{1}{m})P_1 \leq \frac{P}{m} + P_1$. Certainly $\text{OPT} \leq C^{\pi}$, and thus the upper bound is proved. □

We normalize the job processing time by dividing each p_{ij} by the quantity $2 \cdot \max\{P/m, P_1\}$, for all i, j. This way, we have

$$\frac{1}{2k} \leq \text{OPT} \leq 1. \tag{1}$$

Note that from the proof of Lemma 1 we also have $C^\pi \leq 1$, where π is the schedule produced by the list scheduling algorithm and C^π denotes its makespan. We aim to find a better schedule than π and therefore, in the sequel, we consider only those feasible schedules having a makespan less than or equal to 1.

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$, for every integer $n \geq 1$. For some real number $\gamma \in (0, 1)$, which will be determined later (in Eq. (4)), we partition the job set $\mathcal{J}$ into two subsets of *big* jobs and *small* jobs, as follows.

$$\mathcal{B} = \{J_i \mid \exists j \in [k],\ p_{ij} \geq \gamma\},\ \text{and}\ \mathcal{S} = \{J_i \mid \forall j \in [k],\ p_{ij} < \gamma\}. \tag{2}$$

The next lemma states that there are not too many big jobs.

Lemma 2. *There are at most $\frac{mk}{\gamma}$ big jobs.*

Proof. If there were more than mk/γ big jobs, then in any feasible schedule there would be at least one of the mk machines which processes strictly more than $1/\gamma$ big job, resulting in a makespan greater than 1. This contradicts Eq. (1). □

At the high-level, the basic idea in our PTAS is as follows. First we compute the *configuration* for each feasible schedule (having a makespan ≤ 1), and the feasible schedules are partitioned into groups by their configurations. Then for each group, we use its configuration to construct a feasible schedule such that its makespan is very close to the minimum makespan of the schedules in the group. Lastly, we return the constructed schedule with the minimum makespan over all the groups.

2.1. Configuration

Recall that π^* denotes an optimal schedule and its makespan is OPT, which is lower and upper bounded in Eq. (1). Recall also that the makespan of all the feasible schedules considered is at most 1. We will determine the parameter γ later (in Eq. (4)), which depends on the worst-case approximation ratio we want to achieve.

Let $\delta \in (0, 1)$ be a multiple of γ (again this multiple will be determined later, in Eq. (4)), such that $\mu = 1/\delta$ is an integer. We call an interval of length δ a δ-interval. (In our discussion, these intervals are half open.) The time interval $[0, 1)$ is partitioned into μ consecutive δ-intervals; and we let I_t denote the t-th δ-interval $[(t-1)\delta, t\delta)$, for each $t \in [\mu]$.

Given a feasible schedule π (with makespan ≤ 1), for each job J_i, we define its *assignment* as $X_i = (\ell, s_1, s_2, \ldots, s_k)$, where ℓ is the index of the flow-shop to which the job J_i is assigned in the schedule π, and s_j records the index of the δ-interval in which the j-th operation is started. That is, the machine $M_{\ell,j}$ starts processing the job J_i in the δ-interval $[(s_j - 1)\delta, s_j\delta)$. Let $X_{\mathcal{B}} = (X_i)_{J_i \in \mathcal{B}}$ and $X_{\mathcal{S}} = (X_i)_{J_i \in \mathcal{S}}$.

In the schedule π, for each δ-interval I_t, $t \in [\mu]$ and each machine $M_{\ell,j}$, $(\ell, j) \in [m] \times [k]$, we define $L_{t,\ell,j}$ to be the *workload* of small jobs, which is the total time inside the interval I_t the machine $M_{\ell,j}$ spends for processing small jobs. Furthermore, we always round $L_{t,\ell,j}$ up to the nearest multiple of γ. Let $L = (L_{t,\ell,j})_{(t,\ell,j) \in [\mu] \times [m] \times [k]}$.

Then $(X_{\mathcal{B}}, L)$ is defined as the *configuration* of the schedule π, or we say that the schedule π is *associated* with the configuration $(X_{\mathcal{B}}, L)$. It is important to note that the configuration does not have any information about the assignments of small jobs. Clearly, every feasible schedule is associated with exactly one configuration; the feasible schedules associated with the same configuration form a group. The following Lemma 3 states that there are not too many distinct configurations. Let $\mathcal{C}$ be the collection of all configurations.

Lemma 3. *There are at most $(m\mu^k)^{mk/\gamma}(\delta/\gamma + 1)^{mk\mu}$ distinct configurations in $\mathcal{C}$.*

Proof. From Lemma 2, the number of big jobs is at most mk/γ. Since every job can have at most $m\mu^k$ different assignments, the number of all possible assignments $X_{\mathcal{B}}$ for big jobs is no greater than $(m\mu^k)^{|\mathcal{B}|} \leq (m\mu^k)^{mk/\gamma}$.

For every $L_{t,\ell,j}$ that is a multiple of γ, its value is in $\{0, \gamma, 2\gamma, \ldots, \delta\}$. That is, there are $\delta/\gamma + 1$ different possible values. Therefore, the number of all possible L is no greater than $(\delta/\gamma + 1)^{mk\mu}$.

Putting these two upper bounds together, there are at most $(m\mu^k)^{mk/\gamma}(\delta/\gamma + 1)^{mk\mu}$ distinct configurations in $\mathcal{C}$. □

2.2. The PTAS

We want to construct a feasible schedule for every configuration in $\mathcal{C}$, such that the makespan of the constructed schedule is very close to the minimum makespan among all the feasible schedules associated with the same configuration. For simplicity, we fix a configuration and assume that the optimal schedule π^* is associated with this configuration. That is, among all the feasible schedules associated with this configuration, the minimum makespan is OPT.

Algorithm SLIDE-I:

Input: m parallel identical k-stage flow-shops, $\mathcal{J}$ with known assignments, γ, δ;
Output: A feasible schedule π (with makespan at most $\mathrm{OPT} + 2(k-1)\delta$).

Step 1. For each machine $M_{\ell,j}$ and each interval I_t:
1.1. let $\mathcal{O}_{t,\ell,j}$ be the operation set with starting time in I_t on $M_{\ell,j}$;
1.2. schedule the operations of $\mathcal{O}_{t,\ell,j}$ in non-decreasing processing time;
1.2. let $\pi_{t,\ell,j}$ denote the sub-schedule for $\mathcal{O}_{t,\ell,j}$;
Step 2. For each machine $M_{\ell,j}$:
2.1. concatenate $\pi_{t,\ell,j}$ in increasing t;
2.2. let $\pi_{\ell,j}$ denote the sub-schedule on $M_{\ell,j}$;
Step 3. For each machine $M_{\ell,j}$:
3.1. delay the sub-schedule $\pi_{\ell,j}$ by $2(j-1)\delta$ time;
Step 4. Return the final whole schedule denoted as π.

Fig. 1. A high-level description of SLIDE-I.

We describe an algorithm called SLIDE-I (see Fig. 1) that constructs a feasible schedule when the assignments of all the jobs of $\mathcal{J}$ are known, that is, more information than the configuration. Using the assignments, the algorithm first collects for each machine $M_{\ell,j}$ the set of operations it needs to start in the interval I_t; let $\mathcal{O}_{t,\ell,j}$ denote this set of operations, for every $(t,\ell,j) \in [\mu] \times [m] \times [k]$. Next, the machine $M_{\ell,j}$ processes the operations of $\mathcal{O}_{t,\ell,j}$ in a non-decreasing order of processing time (in fact, any order suffices as long as all operations can be started in the interval I_t), denoted as $\vec{\mathcal{O}}_{t,\ell,j}$ (in Lemma 4 we prove that all these operations can be started in the interval I_t, in particular in the non-decreasing order of processing time); thus the sub-schedule on $M_{\ell,j}$ is $\langle \vec{\mathcal{O}}_{1,\ell,j}, \vec{\mathcal{O}}_{2,\ell,j}, \ldots, \vec{\mathcal{O}}_{\mu,\ell,j} \rangle$. Lastly, the machine $M_{\ell,j}$ *delays* the processing by $2(j-1)\delta$ time.

Lemma 4. *If in the configuration the assignments for all the jobs of $\mathcal{J}$ are known, then the algorithm* SLIDE-I *produces a feasible schedule with makespan at most $\mathrm{OPT} + 2(k-1)\delta$.*

Proof. The algorithm SLIDE-I is a variant of the algorithm designed by Hall [12] for the classic k-stage flow-shop problem. We prove below that the schedule is feasible.

First, we claim that every operation of $\vec{\mathcal{O}}_{t,\ell,j}$ starts its processing in the interval $I_{t+2(j-1)}$. This is true because $\vec{\mathcal{O}}_{t,\ell,j}$ is an re-ordering of the operations of $\mathcal{O}_{t,\ell,j}$ with non-decreasing processing time, and therefore the last operation of $\vec{\mathcal{O}}_{t,\ell,j}$ still starts its processing in the interval I_t if the machine $M_{\ell,j}$ does not delay the processing by $2(j-1)\delta$ time.

Second, the delay in processing is sufficient to make the schedule feasible. Assume the job J_i is assigned to the ℓ-th flow-shop in the optimal schedule π^*, its $(j-1)$-st operation starts in the interval $I_{t'}$ and its j-th operation starts in the interval I_t, for some $t', t \in [\mu]$. Originally the j-th operation starts after the $(j-1)$-st operation finishes; but due to re-ordering the $(j-1)$-st operation could be delayed as much as δ time while the j-th operation could be started as much as δ time earlier. Therefore, delaying the job processing by 2δ time relative to the machine $M_{\ell,j-1}$, the machine $M_{\ell,j}$ can feasibly process all its jobs in order.

In summary, if the machines do not delay the processing, then the schedule, which could be infeasible, produced by the algorithm SLIDE-I has a makespan at most OPT. By *sliding* the processing on the machine $M_{\ell,j}$ by a $2(j-1)\delta$-length time window, the schedule becomes feasible with its makespan increased by $2(k-1)\delta$. This finishes the proof. □

Unfortunately, given a configuration, we do not have the assignment information about the small jobs, but only the small job workload for each machine inside each δ-interval. We next try to obtain from the configuration the assignment information of "most" small jobs. To this purpose, we construct a linear program (LP) with the decision variables $y_{i,X}$, each for a small job and an assignment. That is, $y_{i,X} = 1$ if and only if the small job J_i has an assignment X in the given configuration. Recall that we use $\mathcal{S}$ to denote the set of small jobs and that there are at most $m\mu^k$ different assignments for each job.

(LP)

$$\sum_{X} y_{i,X} = 1, \qquad \forall J_i \in \mathcal{S};$$

$$\sum_{J_i \in \mathcal{S}, X=(\ell,s_1,\ldots,s_j=t,\ldots,s_k)} p_{ij} y_{i,X} \le L_{t,\ell,j}, \quad \forall (t,\ell,j) \in [\mu] \times [m] \times [k];$$

$$y \ge 0.$$

In this LP, every small job J_i must have an assignment, and the workload of the small jobs on the machine $M_{\ell,j}$ inside the interval I_t must be less than or equal to $L_{t,\ell,j}$, due to rounding. Clearly, there are only $|\mathcal{S}| + km\mu$ constraints and

Algorithm SLIDE-II:

Input: m parallel identical k-stage flow-shops, $\mathcal{J}$, a configuration, γ, δ;
Output: A feasible schedule π.

Step 1. Construct a linear program using the configuration and solve it;
1.1. obtain the job subset $\mathcal{S}_1$ with known assignments, and $\mathcal{S}_2$;
Step 2. Run the algorithm SLIDE-I on the job subset $\mathcal{B} \cup \mathcal{S}_1$;
2.1. obtain a partial schedule π;
Step 3. Append the jobs of $\mathcal{S}_2$ to the end of the schedule π;
3.1. each flow-shop is assigned with $|\mathcal{S}_2|/m$ small jobs of $\mathcal{S}_2$;
Step 4. Return the final whole schedule still denoted as π.

Fig. 2. A high-level description of SLIDE-II.

therefore the number of variables $|\mathcal{S}|m\mu^k$ is considerably larger. It follows that a basic feasible solution to this LP has at most $|\mathcal{S}| + km\mu$ positive values. Note that for every small job J_i, if there is an X such that $y_{i,X}$ is a positive fractional value, then there must be another distinct X' such that $y_{i,X'}$ is a positive fractional value too. Suppose the total number of positive fractional values in the basic feasible solution is N. Let $\mathcal{S}_1$ denote the subset of small jobs for each of which there is an associated variable having value 1, that is, from the basic solution we know the assignment for each small job of $\mathcal{S}_1$; and let $\mathcal{S}_2 = \mathcal{S} - \mathcal{S}_1$ denote the subset of small jobs for each of which there are some (equivalently, at least two) associated variables having fractional values. It follows that $|\mathcal{S}_2| \le N/2$, and thus $|\mathcal{S}_1| \ge |\mathcal{S}| - N/2$. Therefore, the total number of positive values in the basic solution is at least $|\mathcal{S}| - N/2 + N = |\mathcal{S}| + N/2$. From $|\mathcal{S}| + N/2 \le |\mathcal{S}| + km\mu$ we have $N \le 2km\mu$, and thus we conclude that

$$|\mathcal{S}_2| \le \frac{N}{2} \le km\mu. \tag{3}$$

We summarize the above result from the LP in the following lemma.

Lemma 5. *Given a configuration where the assignments for all the jobs of $\mathcal{B}$ are known, the assignments for most, but no more than $km\mu$, of jobs of $\mathcal{S}$ can be obtained by solving the constructed LP.*

Now we are ready to describe the second algorithm called SLIDE-II (see Fig. 2). In the first step, the algorithm uses the given configuration to specify a linear program LP as stated in the above, and obtains a basic solution to the LP. In the second step, the algorithm retrieves the assignments for the small jobs of $\mathcal{S}_1$, and calls the algorithm SLIDE-I on the job set $\mathcal{B} \cup \mathcal{S}_1$ since it has the assignments for all the big jobs from the given configuration. Let π denote the achieved schedule. Lastly, the algorithm appends the small jobs of $\mathcal{S}_2$ to the end of the schedule π, arbitrarily but each of the m flow-shops is assigned $|\mathcal{S}_2|/m$ small jobs. (When $|\mathcal{S}_2|/m$ is not integral, some flow-shops are assigned $\lceil |\mathcal{S}_2|/m \rceil$ small jobs of $\mathcal{S}_2$, while the others are assigned $\lfloor |\mathcal{S}_2|/m \rfloor$ small jobs.)

Lemma 6. *Given the configuration, the algorithm* SLIDE-II *produces a feasible solution with makespan at most* $OPT + 2(k-1)(\delta + \gamma) + \mu\gamma + (k\mu + k - 1)\gamma$.

Proof. The feasibility of the schedule π follows from the correctness of the algorithm SLIDE-I, as summarized in Lemma 4. In the last step the algorithm only appends the small jobs of $\mathcal{S}_2$ to the end of the schedule π. Therefore, the final achieved schedule is still feasible.

From Lemma 4 we know that the makespan of the schedule π (by the algorithm SLIDE-I) is at most $\mathrm{OPT} + 2(k-1)(\delta + \gamma) + \mu\gamma$, where the additional terms γ in $(\delta + \gamma)$ and $\mu\gamma$ are caused by the rounding up the small job workload to the nearest multiple of γ.

Since for each small job of $\mathcal{S}_2$, the processing time of every operation is at most γ, assigning it to any flow-shop will increase the completion time by at most $k\gamma$. Clearly, assigning two small jobs to a flow-shop will increase the completion time by at most $(k+1)\gamma$, and so on. From Eq. (3), every flow-shop processes at most $k\mu$ jobs of $\mathcal{S}_2$, and thus its completion time increases by at most $(k\mu + k - 1)\gamma$. Therefore, the makespan of the final achieved schedule is at most $\mathrm{OPT} + 2(k-1)(\delta + \gamma) + \mu\gamma + (k\mu + k - 1)\gamma$. □

Our final algorithm, called SLIDE-III, for the (m, k)-PFS problem runs the algorithm SLIDE-II on every configuration to achieve a schedule, and returns the best schedule among them, *i.e.*, the one with the minimum makespan.

Theorem 1. *The algorithm* SLIDE-III *can be designed into a PTAS for the (m, k)-PFS problem.*

Proof. For any $\epsilon \in (0,1)$, we show how to set up the values for the parameters δ and γ such that the makespan of the schedule returned by the algorithm SLIDE-III is within $(1+\epsilon)$OPT. Recall that the job processing times have been normalized to ensure that Eq. (1) holds for OPT. Recall also that δ is a multiple of γ. For ease of presentation (to avoid the use of ceiling function) we assume $\epsilon = \frac{1}{T}$ for some positive integer T. Let

$$\delta = \frac{\epsilon}{8k(k-1)}, \text{ and } \gamma = \frac{\epsilon^2}{64(k+1)k^2(k-1)}. \tag{4}$$

From Lemma 2, the number of big jobs is at most mk/γ, which is polynomial in $m, k, \frac{1}{\epsilon}$. Moreover, when m and k are fixed constants and ϵ is given (and thus a constant as well), mk/γ is a constant. Similarly, from Lemma 3, the number of distinct configurations is at most $(m\mu^k)^{mk/\gamma}(\delta/\gamma+1)^{mk\mu}$, which is a constant when m, k, ϵ are fixed constants. That is, the algorithm SLIDE-III makes only a constant number of calls to the algorithm SLIDE-II.

Inside the execution of the algorithm SLIDE-II, the constructed linear program LP contains $|\mathcal{S}| + km\mu$ constraints and $|\mathcal{S}|m\mu^k$ variables. That is, the size of the LP is polynomial when m, k, ϵ are fixed. Since a linear program can be solved in polynomial time, for example by the interior point method [19], and the running time of the algorithm SLIDE-I is polynomial in the number of jobs which have known assignments, the running time of the algorithm SLIDE-II is polynomial in the number of jobs. In summary, the algorithm SLIDE-III is polynomial in n, the number of jobs, when m, k, ϵ are fixed constants.[3]

For the performance ratio, from Lemma 6 we only need to measure the additive error term against OPT. By $\mu = 1/\delta$ and Eq. (4), we have the following:

$$\begin{aligned}
&2(k-1)(\delta+\gamma) + \mu\gamma + (k\mu + k - 1)\gamma \\
&= 2(k-1)\delta + (3(k-1) + (k+1)\mu)\,\gamma \\
&= 2(k-1)\delta + \left(3(k-1) + \frac{k+1}{\delta}\right)\gamma \\
&= \frac{\epsilon}{4k} + \left(3(k-1) + \frac{8(k+1)k(k-1)}{\epsilon}\right)\frac{\epsilon^2}{64(k+1)k^2(k-1)} \\
&= \frac{\epsilon}{4k} + \frac{3\epsilon^2}{64(k+1)k^2} + \frac{\epsilon}{8k} \\
&= \left(\frac{3}{4} + \frac{3\epsilon}{32(k+1)k}\right)\frac{1}{2k}\epsilon \\
&< \frac{1}{2k}\epsilon.
\end{aligned}$$

It follows that the makespan of the schedule produced by the algorithm SLIDE-III is less than $\text{OPT} + \frac{1}{2k}\epsilon < (1+\epsilon)\text{OPT}$, by Eq. (1). This proves the theorem. □

3. Conclusions

We presented a polynomial-time approximation scheme (PTAS) for the (m,k)-PFS problem, in which there are m parallel identical k-stage flow-shops. Our PTAS requires that both m and k are fixed integers. Since the classic k-stage flow-shop problem is strongly NP-hard for a fixed $k \geq 3$, our PTAS seems the best possible unless P = NP. The APX-hardness of the classic k-stage flow-shop problem when k is a part of the input implies the APX-hardness of the (m,k)-PFS problem when k is a part of the input. An open problem is to investigate the (in-)approximability of the (m,k)-PFS problem when m is a part of the input while k is a constant.

Acknowledgements

The authors are grateful to several reviewers for their insightful comments and changes, on both the extended abstract and an earlier version of this paper, that improve the presentation greatly.

Tong was supported in part by funds from the Office of the Vice President for Research & Economic Development and the Georgia Southern University, and an Alberta Innovates Technology Futures (AITF) Graduate Student Scholarship. Miyano was partially supported by the Grants-in-Aid for Scientific Research of Japan (KAKENHI), Grant Number 26330017. Goebel was supported by the AITF and the Natural Sciences and Engineering Research Council of Canada (NSERC). Lin was partially supported by the NSERC and his work was mostly done during his sabbatical leave at the Kyushu Institute of Technology, Iizuka Campus.

[3] One may certainly apply any techniques to bring down the time complexity to the lowest possible. Here we only emphasize that the time is polynomial, while ignoring the exact order.

References

[1] B. Chen, Analysis of classes of heuristics for scheduling a two-stage flow shop with parallel machines at one stage, J. Oper. Res. Soc. 46 (1995) 234–244.
[2] B. Chen, C.A. Glass, C.N. Potts, V.A. Strusevich, A new heuristic for three-machine flow shop scheduling, Oper. Res. 44 (1996) 891–898.
[3] R.W. Conway, W.L. Maxwell, L.W. Miller, Theory of Scheduling, Addison–Wesley, Reading, Mass., 1967.
[4] J. Dong, W. Tong, T. Luo, X. Wang, J. Hu, Y. Xu, G. Lin, An FPTAS for the parallel two-machine flowshop problem, Theoret. Comput. Sci. 657 (2017) 64–72.
[5] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.
[6] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Math. Oper. Res. 1 (1976) 117–129.
[7] T. Gonzalez, S. Sahni, Flowshop and jobshop schedules: complexity and approximation, Oper. Res. 26 (1978) 36–52.
[8] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell Syst. Tech. J. 45 (1966) 1563–1581.
[9] J.N.D. Gupta, Two-stage, hybrid flowshop scheduling problem, J. Oper. Res. Soc. 39 (1988) 359–364.
[10] J.N.D. Gupta, A.M.A. Hariri, C.N. Potts, Scheduling a two-stage hybrid flow shop with parallel machines at the first stage, Ann. Oper. Res. 69 (1997) 171–191.
[11] J.N.D. Gupta, E.A. Tunc, Schedules for a two-stage hybrid flowshop with parallel machines at the second stage, Int. J. Prod. Res. 29 (1991) 1489–1502.
[12] L.A. Hall, Approximability of flow shop scheduling, Math. Program. 82 (1998) 175–190.
[13] D.W. He, A. Kusiak, A. Artiba, A scheduling problem in glass manufacturing, IIE Trans. 28 (1996) 129–139.
[14] D.S. Hochbaum, D.B. Shmoys, Using dual approximation algorithms for scheduling problems theoretical and practical results, J. ACM 34 (1987) 144–162.
[15] J.A. Hoogeveen, J.K. Lenstra, B. Veltman, Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard, European J. Oper. Res. 89 (1996) 172–175.
[16] K. Jansen, M. Klein, J. Verschae, Closing the gap for makespan scheduling via sparsification techniques, in: The 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, 2016, Article no. 72.
[17] K. Jansen, M.I. Sviridenko, Polynomial time approximation schemes for the multiprocessor open and flow shop scheduling problem, in: STACS, in: LNCS, vol. 1770, 2000, pp. 455–465.
[18] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Nav. Res. Logist. Q. 1 (1954) 61–68.
[19] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (1984) 373–395.
[20] C.-Y. Lee, G.L. Vairaktarakis, Minimizing makespan in hybrid flowshops, Oper. Res. Lett. 16 (1994) 149–158.
[21] X. Qi, New results for scheduling two flowlines, working paper of Hong Kong University of Science and Technology.
[22] R. Ruiz, J.A. Vázquez-Rodríguez, The hybrid flow shop scheduling problem, European J. Oper. Res. 205 (2010) 1–18.
[23] S.K. Sahni, Algorithms for scheduling independent tasks, J. ACM 23 (1976) 116–127.
[24] P. Schuurman, G.J. Woeginger, A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem, Theoret. Comput. Sci. 237 (2000) 105–122.
[25] G. Vairaktarakis, M. Elhafsi, The use of flowlines to simplify routing complexity in two-stage flowshops, IIE Trans. 32 (2000) 687–699.
[26] H. Wang, Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions, Expert Syst. 22 (2005) 78–85.
[27] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevastjanov, D.B. Shmoys, Short shop schedules, Oper. Res. 45 (1997) 288–294.
[28] X. Zhang, S. van de Velde, Approximation algorithms for the parallel flow shop problem, European J. Oper. Res. 216 (2012) 544–552.