Theoretical Computer Science 776 (2019) 117–124

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Scheduling two-stage jobs on multiple flowshops ☆

Guangwei Wu [a,b], Jianer Chen [c,d], Jianxin Wang [a,*]

[a] School of Computer Science and Engineering, Central South University, ChangSha 410083, Hunan, China
[b] College of Computer and Information Engineering, Central South University of Forestry and Technology, ChangSha 410004, Hunan, China
[c] School of Computer Science & Education Software, Guangzhou University, Guangzhou, Guangdong 510006, China
[d] Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA

ARTICLE INFO

Article history:
Received 11 September 2016
Received in revised form 9 January 2019
Accepted 17 January 2019
Available online 23 January 2019
Communicated by T. Erlebach

Keywords:
Scheduling
Two-stage flowshop
Algorithm
Cloud computing

ABSTRACT

Scheduling two-stage jobs on multiple two-stage flowshops is studied. A new formulation for configurations of the scheduling is proposed, leading directly to improvements on complexity of scheduling algorithms for the problem. Motivated by observations in practice, we present a deeper study on the structures of the problem that leads to a new approach that gives very significant improved scheduling algorithms for the problem when the costs of the two stages differ significantly.

1. Introduction

A (*two-stage*) *flowshop* consists of an *R-processor* and a *T-processor* that can run in parallel. A *two-stage job* J consists of an *R-operation* and a *T-operation* such that the T-operation of J cannot start on the T-processor of a two-stage flowshop M until the R-operation of J has been completed on the R-processor of *the same flowshop* M. The current paper studies scheduling algorithms for two-stage jobs on multiple two-stage flowshops.

The research was motivated by the current needs from data centers and cloud computing. Recently, a cloud paradigm, *TransCom*, was proposed [23], which takes not only application software and data but also system software such as operating systems as resources. As a consequence, client devices in the system can be very light and significantly diversified. Clients dynamically request resources via networks from the cloud (see [23] for details).

In such a system, a significant amount of resources requested by clients are codes of system/application software, which in general are large in size, thus, stored in secondary memory. When a server receives a request from a client for a resource, it will have to first read the resource from secondary memory into main memory, then send to the client via networks. As a result, the request consists of a disk-read operation and a network-transmission operation. It is also natural to require that the network-transmission do not start until the requested resource has been brought into main memory. Therefore, in such a system, resource requests become two-stage jobs, consisting of the disk-read and the network-transmission operations, while each server becomes a two-stage flowshop (note that the disk-read and network-transmission can run in parallel in the same server). We should remark that the times for disk-read and for network-transmission in a typical server are in

☆ This work is supported by the National Natural Science Foundation of China under grants 61420106009 and 61672536; 111 project (No. B18059); Hunan Provincial Science and Technology Program (2018wk4001).

* Corresponding author.
E-mail addresses: will99031827@hotmail.com (G. Wu), chen@cs.tamu.edu (J. Chen), jxwang@mail.csu.edu.cn (J. Wang).

https://doi.org/10.1016/j.tcs.2019.01.017
0304-3975/

general comparable, and, due to the impact of cache systems, they need not have a linear relation [22]. Therefore, neither can be simply ignored if we want to maintain good performance for the cloud system.

We give a brief review on the results in the scheduling literature that are related to the above problem. The classical MAKESPAN problem can be regarded as scheduling one-stage jobs on multiple one-stage machines [12], while the classical TWO-STAGE FLOWSHOP problem is scheduling two-stage jobs on a single two-stage flowshop [10]. Other scheduling models that deal with multiple-stage jobs include various "hybrid" shop scheduling problems [13], which allow multiple machines for each job stage, but have no specific bonding for the machines that execute the stages of the same job. See also [14] and references therein for further study on shop scheduling algorithms. A model that also deals with multiple-stage jobs is that of scheduling jobs with setup costs [1], where, however, a machine cannot run the setup stage for one job and the regular processing stage for another job in parallel.

Kovalyov [11] studied the problem of scheduling two-stage jobs on multiple two-stage flowshops. He et al. [8] proposed a heuristic algorithm. Vairaktarakis and Elhafsi [16] considered the problem in their study on the hybrid flowshops and proposed a pseudo-polynomial time algorithm for scheduling two-stage jobs on two flowshops. Zhang and van de Velde [21] developed approximation algorithms for scheduling two-stage jobs on two and three flowshops. Recently, Dong et al. [4] proposed a pseudo-polynomial time algorithm for scheduling two-stage jobs on a fixed number of flowshops, and developed a fully polynomial-time approximation scheme based on the pseudo-polynomial time algorithm, a result similar to that of [11] but developed independently (see [5] for details). Approximation algorithms for k-stage jobs on multiple k-stage flowshops for general k have also been studied [15].

Recently, the scheduling problem has been studied when the number of flowshops is a part of input. Wu et al. developed an 2.6-approximation algorithm for the problem [18]. They also considered two special cases of the problem where the jobs are inclined, and proposed two 11/6-approximation algorithms [19]. The relationship between the problem and the classical MAKESPAN problem has also been studied [20].

The current paper, worked out independently of [4], presents more efficient algorithms for scheduling two-stage jobs on a fixed number of flowshops. We propose a new formulation for configurations for schedules of two-stage jobs on multiple flowshops, which is very different from previous ones [4,16]. We show that dynamic programming based on our formulation leads to improvements on algorithm complexity for the scheduling problem.

Our further study on the problem was motivated by the observation that in many cases in practice, the execution times for the two stages can differ very significantly. We present a deeper study on the structures of the problem that leads to a more carefully designed algorithm. With more thorough analysis, we are able to show that the new approach will give a very significantly improved scheduling algorithm for the problem when the costs of the two stages are significantly different.

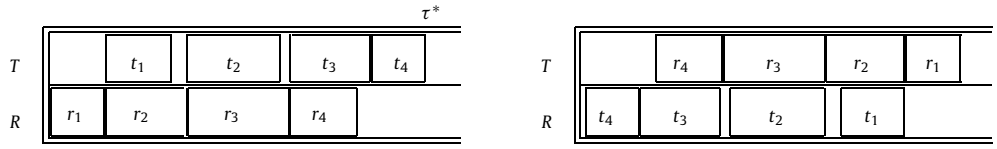
A fully polynomial-time approximation scheme (FPTAS) for the problem on a fixed number of two-stage flowshops is also developed, based on our new configurations of the schedules. Our FPTAS has improved time complexity compared with previously proposed FPTASs [11,4].

2. Single flowshop scheduling and dual scheduling

Consider n two-stage jobs $\{J_1, \ldots, J_n\}$ to be scheduled on m identical two-stage flowshops $\{M_1, \ldots, M_m\}$, where each job J_i is given by a pair (r_i, t_i) of integers, where the *R-time* r_i (resp. *T-time* t_i) is the time for processing the R-operation (resp. T-operation) of J_i by the R-processor (resp. T-processor) of a flowshop. A *schedule* $\mathcal{S}$ of a job set on m flowshops consists of an *assignment* of each job to a flowshop, and, for each flowshop, the execution orders of the R- and T-operations of the jobs assigned to that flowshop in its corresponding processors. The *completion time* of a flowshop M is the time when M finishes the execution of the last T-operation for the jobs assigned to M. The *makespan* $C_{\max}$ of a schedule $\mathcal{S}$ on multiple flowshops is the largest flowshop completion time under the schedule $\mathcal{S}$ over all flowshops. The objective of our scheduling is to construct a schedule that minimizes the makespan. Following the three-field notation $\alpha|\beta|\gamma$ suggested by Graham et al. [7], this scheduling model can be written as $P|2\text{FL}|C_{\max}$, or $P_m|2\text{FL}|C_{\max}$ if the number m of flowshops is a fixed constant.

For $m = 1$, the problem $P_1|2\text{FL}|C_{\max}$ becomes the classical TWO-STAGE FLOWSHOP problem. It is well-known [10] that if we are only concerned with the completion time of the flowshop, then we can consider only schedules that are given by ordered sequences of the jobs such that both executions of the R- and T-operations of the jobs strictly follow the given order. Under this condition, for each job J_i, the T-processor of the flowshop should start executing the T-operation of J_i as soon as the R-operation of J_i and the T-operation of J_{i-1} are both completed. Note that the R-operations of the jobs can be executed continuously. Thus, if our interests are in minimizing the makespan of schedules, then we can make the following assumptions.

Lemma 2.1. *Let $\mathcal{S} = \langle J_1, \ldots, J_t \rangle$ be a two-stage job schedule on a single flowshop, where $J_i = (r_i, t_i)$ for all i. Let $\bar{\rho}_h$ and $\bar{\tau}_h$, resp., be the times at which the R- and T-operations of job J_h are started. Then for $h \geq 1$, we can assume (with $\bar{\tau}_0 = t_0 = 0$) $\bar{\rho}_h = \sum_{i=1}^{h-1} r_i$ and $\bar{\tau}_h = \max\{\bar{\rho}_h + r_h, \bar{\tau}_{h-1} + t_{h-1}\}$.*



The schedule $\mathcal{S}$ on the original job set — The schedule $\mathcal{S}_1^d$ on the dual job set

Fig. 1. Schedules for a job set and its dual.

Thus, the configuration of a schedule of a job set on a flowshop can be given by a pair (ρ, τ), which gives the finish times of the R-operation and the T-operation of the last job assigned to the flowshop. The pair (ρ, τ), which will be called the *status* of the schedule, can be easily updated by Lemma 2.1.

Using the classical Johnson's algorithm, scheduling two-stage jobs on a single two-stage flowshop can be solved *optimally* in time $O(n \log n)$ [10]. Johnson's algorithm can be described as follows:

Johnson's Algorithm [10]. Divide the jobs into two groups G_1 and G_2, where G_1 contains jobs (r_h, t_h) with $r_h \le t_h$, and G_2 contains jobs (r_g, t_g) with $r_g > t_g$. Order the jobs in a sequence that starts with the jobs in G_1, sorted in nondecreasing order of R-times, followed by the jobs in G_2, sorted in nonincreasing order of T-times.

The job order described in Johnson's Algorithm is called *Johnson's order*. In scheduling two-stage jobs on multiple two-stage flowshops, once we determined how the jobs are assigned to the flowshops, Johnson's order of the jobs assigned to each flowshop will give an optimal execution order for the flowshop. As a result, what remains unsolved is how we determine the assignment of the jobs to the flowshops. Unfortunately, this task is intractable: when all jobs have R-time 0, the problem becomes the classical MAKESPAN problem, which is NP-hard even for two machines [2], and becomes strongly NP-hard when the number of machines is given as part of the input [6].

The *dual job* of a two-stage job $J_i = (r_i, t_i)$ is $J_i^d = (t_i, r_i)$. For a schedule $\mathcal{S} = \langle J_1, J_2, \ldots, J_n \rangle$ of two-stage jobs on a two-stage flowshop, the *dual schedule* of $\mathcal{S}$ on the dual jobs of $\mathcal{S}$ is $\mathcal{S}^d = \langle J_n^d, \ldots, J_2^d, J_1^d \rangle$, where J_i^d is the dual job of J_i for $1 \le i \le n$. It is easy to verify that if the schedule $\mathcal{S}$ follows Johnson's order, then the dual schedule $\mathcal{S}^d$ also follows Johnson's order. In fact, we have a more general result, as given in the following theorem.

Theorem 2.2. *On a single two-stage flowshop, optimal schedules of a job set $G = \{J_1, \ldots, J_n\}$ and optimal schedules of the dual job set $G^d = \{J_1^d, \ldots, J_n^d\}$ have the same completion time. Moreover, if a schedule $\mathcal{S}$ is optimal for the job set G then its dual schedule $\mathcal{S}^d$ is optimal for the dual job set G^d.*

Proof. The dual job of job $J_h = (r_h, t_h)$ is $J_h^d = (t_h, r_h)$. Let $\mathcal{S} = \langle J_1, \ldots, J_n \rangle$ be an optimal schedule for G with completion time τ^*. Consider the schedule $\mathcal{S}_1^d = \langle J_n^d, \ldots, J_1^d \rangle$ for the dual job set G^d, which will be called "flipping" of $\mathcal{S}$, where the R-operation (resp. T-operation) of J_i^d starts at time $\tau^* - t$ if the T-operation (resp. R-operation) of J_i ends at time t (see Fig. 1).

Obviously, $\mathcal{S}_1^d$ is a valid schedule for G^d and the schedules $\mathcal{S}$ and $\mathcal{S}_1^d$ have the same completion time. Thus, optimal schedules for G^d have completion time not larger than that of optimal schedules for G. On the other hand, flipping an optimal schedule for G^d gives a schedule for G with the same completion time, so optimal schedules for G^d have completion time not smaller than that of optimal schedules for G. This gives the first part of the lemma.

Observe that flipping the optimal schedule $\mathcal{S}$ for G gives an optimal schedule $\mathcal{S}_1^d$ for G^d, whose completion time is not smaller than that of the dual schedule $\mathcal{S}^d$ for G^d. Thus, $\mathcal{S}^d$ is also an optimal schedule for G^d. □

When we study scheduling two-stage jobs on multiple two-stage flowshops, Theorem 2.2 directly implies the following theorem.

Theorem 2.3. *On multiple two-stage flowshops, the optimal schedule of the job set G and the optimal schedule of the dual job set G^d have the same makespan. Moreover, an optimal schedule for the job set G can be easily obtained from an optimal schedule for the dual job set G^d.*

3. On $P_m|2FL|C_{max}$: general case

In this section, we study the problem $P_m|2FL|C_{max}$ for a fixed constant m. Given a set G of two-stage jobs, with a preprocessing, we can assume that $G = \langle J_1, \ldots, J_n \rangle$ is in Johnson's order, where $J_i = (r_i, t_i)$ for all i.

3.1. Pseudo-polynomial time algorithm

Let $R_0 = \sum_{i=1}^n r_i$, $T_0 = \sum_{i=1}^n t_i$, and for each k, let $G_k = \{J_1, J_2, \ldots, J_k\}$ be the set of the first k jobs in G in Johnson's order. Thus, if we follow the order of G and assign the jobs to the flowshops, then, the subsequence received by each

flowshop is also in Johnson's order, giving an optimal schedule of the jobs assigned to the flowshop. Therefore, the status of a flowshop M_h at any moment can be given by a pair (ρ_h, τ_h), where ρ_h and τ_h, respectively, are the completion times of the R- and T-processors of M_h. The status (ρ_h, τ_h) of M_h can be updated by Lemma 2.1 when a new job (r, t) is added to M_h: the new completion time of the R-processor is $\rho_h + r$, and the new completion time of the T-processor is $\max\{\rho_h + r, \tau_h\} + t$. For a schedule $\mathcal{S}_k$ for the job set G_k, the tuple $(k; \rho_1, \tau_1, \ldots, \rho_m, \tau_m)$ is called the *configuration* of $\mathcal{S}_k$ if under the schedule $\mathcal{S}_k$, for each h, the status of the flowshop M_h is (ρ_h, τ_h).

A simple but important observation is that for each $k > 0$, we have:

Fact A. The tuple $(k; \rho_1, \tau_1, \ldots, \rho_m, \tau_m)$ is a configuration of a schedule for the job subset G_k if and only if for some d, $1 \le d \le m$, the tuple $(k-1; \rho_1, \tau_1, \ldots, \rho_{d-1}, \tau_{d-1}, \rho'_d, \tau'_d, \rho_{d+1}, \tau_{d+1}, \ldots, \rho_m, \tau_m)$ is a configuration of a schedule for the job subset G_{k-1}, where ρ'_d and τ'_d satisfy $\rho_d = \rho'_d + r_k$ and $\tau_d = \max\{\rho'_d + r_k, \tau'_d\} + t_k$.

Fact A suggests that we can apply a dynamic programming algorithm [6] to construct an optimal schedule for the two-stage job set G. A straightforward implementation of the algorithm would run in time $O(nm^2 R_0^m (R_0 + T_0)^m)$. In the following, we study how to improve the algorithm complexity.

For the status (ρ_h, τ_h) of a flowshop M_h, by definition, $\rho_h \le \tau_h$. It is also easy to see that $\tau_h - \rho_h \le T_0$. This observation suggests that we can use the pair (ρ_h, δ_h) instead of the pair (ρ_h, τ_h), where $\delta_h = \tau_h - \rho_h$, and $0 \le \delta_h \le T_0$. Note that the pair (ρ_h, τ_h) can be easily obtained from the pair (ρ_h, δ_h). Therefore, the configuration $(k; \rho_1, \tau_1, \ldots, \rho_m, \tau_m)$ of a schedule $\mathcal{S}_k$ for G_k can be represented as the tuple $(k; \rho_1, \delta_1, \ldots, \rho_m, \delta_m)$, where for all h, $\delta_h = \tau_h - \rho_h$ with $0 \le \delta_h \le T_0$, which will be called the *s-configuration* of $\mathcal{S}_k$.

Remark. Our configurations and s-configurations defined above are very different from those proposed in the literature [16, 4]. We will see that our definitions of the configurations lead to faster algorithms.

Our next improvement is to reduce the dimension of the s-configurations. Let $\mathcal{S}_k = (k; \rho_1, \delta_1, \ldots, \rho_m, \delta_m)$ be an s-configuration for the job subset G_k. Let $R_0^k = \sum_{i=1}^k r_i$. By Lemma 2.1, the R-processors of the flowshops run continuously without idle time. Therefore, $\sum_{h=1}^m \rho_h = R_0^k$. This gives

Fact B. ρ_1 can be computed from ρ_2, …, ρ_m: $\rho_1 = R_0^k - \sum_{h=2}^m \rho_h$.

Let $\mathcal{S}_k = (k; \rho_1, \delta_1, \rho_2, \delta_2, \ldots, \rho_m, \delta_m)$ and $\mathcal{S}'_k = (k; \rho_1, \delta'_1, \rho_2, \delta_2, \ldots, \rho_m, \delta_m)$ be s-configurations for the job subset G_k that only differ in the third component, with $\delta_1 < \delta'_1$. It is easy to see that if we can assign the rest of the jobs J_{k+1}, …, J_n to $\mathcal{S}'_k$ to build a minimum makespan schedule for the entire job set G, then the same way of assigning the jobs J_{k+1}, …, J_n to $\mathcal{S}_k$ will also give a minimum makespan schedule of G. Therefore, when all other components are identical, we really only have to record the smallest completion time (thus the smallest value δ_1) for the T-processor of the flowshop M_1.

Thus, we can use a $(2m-1)$-dimensional array H to represent all "useful" s-configurations for G_k such that $H[k; \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (\delta_1, d)$ if by letting $\rho_1 = R_0^k - \sum_{h=2}^m \rho_h$, the value δ_1 is the smallest δ'_1 such that $(k; \rho_1, \delta'_1, \rho_2, \delta_2, \ldots, \rho_m, \delta_m)$ is a valid s-configuration for the job subset G_k (where d gives the flowshop to which the job J_k is assigned).

Our algorithm for $P_m|2\text{FL}|C_{\max}$ is given in Fig. 2, in which steps 3.4–3.7 add the job J_{k+1} to the d-th flowshop in the schedule for the job subset G_k with an s-configuration $\mathcal{S} = (k; \rho_1, \delta_1, \rho_2, \delta_2, \ldots, \rho_m, \delta_m)$. Thus, before adding J_{k+1}, the d-th flowshop has status $(\rho_d, \rho_d + \delta_d)$. By Lemma 2.1, after adding J_{k+1}, for the d-th flowshop, the completion time ρ'_d of the R-processor is $\rho_d + r_{k+1}$, and the completion time τ'_d of the T-processor is $\max\{\rho_d + r_{k+1}, \rho_d + \delta_d\} + t_{k+1}$. Therefore, by the definition, after adding the job J_{k+1}, we should have, as shown in step 3.5 of the algorithm:

$$\delta'_d = \tau'_d - \rho'_d = \max\{r_{k+1}, \delta_d\} + t_{k+1} - r_{k+1}.$$

The last row $H[n; *, \ldots, *]$ of the array H includes all s-configurations of the schedules for the job set $G = G_n$ on the m flowshops, and the value $\max_{1 \le h \le m}\{\rho_h + \delta_h\}$ for an element $H[n; \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (\delta_1, d)$ (where $\rho_1 = R_0 - \sum_{h=2}^m \rho_h$) gives the makespan of the schedule described by the element. Therefore, step 4 of the algorithm gives a schedule for the job set G on the m flowshops that has the minimum makespan. Also note that the value $H[k; \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (\delta_1, d)$ records that the last job J_k in G_k was added to the flowshop M_d to obtain the s-configuration corresponding to $H[k; \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (\delta_1, d)$. Thus, the actual schedule corresponding to the s-configuration can be re-constructed by back-tracking in the array H.

Since $0 \le k \le n$, and $0 \le \rho_h \le R_0$, $0 \le \delta_h \le T_0$, for all h, the array H has $O(nR_0^{m-1}T_0^{m-1})$ elements. Steps 3.2–3.7 of the algorithm, which is applied on each element of H, take time $O(m^2)$. This gives the following theorem.

Theorem 3.1. *Problem $P_m|2FL|C_{\max}$ is solvable in time $O(nm^2 R_0^{m-1} T_0^{m-1})$ and space $O(nR_0^{m-1}T_0^{m-1})$.*

The previously best algorithm for $P_m|2\text{FL}|C_{\max}$, based on a very different definition of schedule configurations, runs in time $O(nm^2(R_0 + T_0)^{2m-1})$ and space $O(m(R_0 + T_0)^{2m-2})$ [4]. For running time, our algorithm in Theorem 3.1 not only replaces the larger factor $R_0 + T_0$ by smaller factors R_0 and T_0, but also reduces the exponent from $2m-1$ to $2m-2$.

Algorithm DynProg-I
INPUT: a set $G = \{J_1, \ldots, J_n\}$ of two-stage jobs, in Johnson's order
OUTPUT: an optimal schedule of G on m two-stage flowshops
1. **for** all $0 \le k \le n$, $0 \le \rho_h \le R_0$, $0 \le \delta_h \le T_0$, $2 \le h \le m$ **do**
   $H[k; \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (+\infty, 0)$;
2. $H[0; 0, 0, \ldots, 0, 0] = (0, 0)$;
3. **for** $k = 0$ **to** $n-1$ **do**
3.1 **for** each $H[k, \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (\delta_1, d_k)$ with $\delta_1 \neq +\infty$ **do**
3.2 $\rho_1 = R_0^k - \sum_{h=2}^{m} \rho_h$;
3.3 **for** $d = 1$ **to** m **do**
3.4 **for** $(1 \le h \le m)$ & $(h \neq d)$ **do** { $\rho'_h = \rho_h$; $\delta'_h = \delta_h$; }
3.5 $\rho'_d = \rho_d + r_{k+1}$; $\delta'_d = \max\{r_{k+1}, \delta_d\} + t_{k+1} - r_{k+1}$;
3.6 **if** $H[k+1; \rho'_2, \delta'_2, \ldots, \rho'_m, \delta'_m] = (\delta_1, d_{k+1})$ with $\delta'_1 < \delta_1$
3.7 **then** $H[k+1; \rho'_2, \delta'_2, \ldots, \rho'_m, \delta'_m] = (\delta'_1, d)$;
4. return the $H[n; \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (\delta_1, d_n)$ with $\min\{\max_h\{\rho_h + \delta_h\}\}$.

Fig. 2. An improved algorithm for $P_m|2\text{FL}|C_{\max}$.

Algorithm Approx
INPUT: a set $\{J_1, \ldots, J_n\}$ of two-stage jobs, $\epsilon > 0$, where $\forall k$, $J_k = (r_k, t_k)$
OUTPUT: a schedule of G on m identical two-stage flowshops

1. let $T_{\max} = \max\{R_0, T_0\}$ and $K = \epsilon \cdot T_{\max}/(nm)$;
2. **for** $i = 1$ **to** n **do** { $r'_i = \lfloor r_i/K \rfloor$; $t'_i = \lfloor t_i/K \rfloor$ };

3. let $G' = \{J'_1, \ldots, J'_n\}$, where for each i, $J'_i = (r'_i, t'_i)$;
4. call algorithm **DynProg-I** to obtain an optimal schedule $\mathcal{S}'$ for G';
5. return $\mathcal{S}$ that is the schedule $\mathcal{S}'$ with each J'_i replaced with J_i.

Fig. 3. An approximation algorithm for $P_m|2\text{FL}|C_{\max}$.

For space complexity, our algorithm seems to use more space because in general $n > m$. However, a careful examination shows that the algorithm in [4] only returns the value of the makespan of an optimal schedule without giving the actual schedule. In order to return a schedule, the algorithm in [4] seems to have to increase its space complexity to at least $O(nm(R_0 + T_0)^{2m-2})$. If we are only interested in the value of the optimal makespan, we can easily modify our algorithm to run in space $O(R_0^{m-1} T_0^{m-1})$. Thus, our algorithm in Theorem 3.1 improves both time complexity and space complexity of the algorithm given in [4].

3.2. Approximation algorithms

There is a fairly standard procedure to develop polynomial-time approximation algorithms based on pseudo-polynomial time algorithms for a problem [9] (see also [3] for a formal description of the conditions on the applicability of the procedure). Using this method, we can develop a polynomial-time approximation algorithm for $P_m|2\text{FL}|C_{\max}$ based on the results in subsection 3.1. Essentially, the procedure first scales the problem instance so that values in the instance become bounded by a polynomial of the instance size, and applies the pseudo-polynomial time algorithm on the scaled instance, whose result then is converted into a solution to the original instance. The approximation algorithm for the $P_m|2\text{FL}|C_{\max}$ problem is given in Fig. 3.

Since the analysis for the approximation ratio for the algorithm **Approx** is very similar to that for the standard procedure [9,6], we give the conclusion directly in the following theorem, with the detailed verification omitted. Readers who are interested in the details are referred to [17].

Theorem 3.2. *There is an algorithm for $P_m|2FL|C_{\max}$ that on a set G of n two-stage jobs and for any real number $\epsilon > 0$, constructs a schedule for G on m two-stage flowshops with a makespan bounded by $Opt(G)(1+\epsilon)$. Moreover, the running time of the algorithm is $O(n^{2m-1}m^{2m}/\epsilon^{2m-2})$.*

Compared to the algorithm in [4] that produces schedules with makespan bounded by $Opt(G)(1+\epsilon)$ but runs in time $O(2^{2m-1}n^{2m}m^{2m+1}/\epsilon^{2m-1})$, our algorithm gives an obvious improvement on the running time.

4. On $P_m|2\text{FL}|C_{\max}$: when R_0 and T_0 differ significantly

In certain cases in practice, the values R_0 and T_0 can differ very significantly. Consider the situation in data centers as described in Section 1. In order to improve the process of data-read/network-transformation, severs in the center may keep certain commonly used software in the main memory so that the time-consuming data-read process can be avoided [22]. Thus, client requests for the code will become two-stage jobs $J_i = (r_i, t_i)$ with $r_i = 0$. As a consequence, the value

$R_0 = \sum_{i=1}^{n} r_i$ can be significantly smaller than the value $T_0 = \sum_{i=1}^{n} t_i$. On the other hand, certain data centers may consist of a large number of slow-speed servers (e.g., PC's) but are equipped with high-speed networks [23], which may make T_0 much smaller than R_0.

We study how to improve the complexity of scheduling algorithms when R_0 and T_0 differ significantly. We divide the study into two cases: (1) $T_0 \gg R_0$ and (2) $T_0 \ll R_0$. We first consider the case $T_0 \gg R_0$.

Since all flowshops are identical, we can re-order the flowshops. An s-configuration $(k; \rho_1, \delta_1, \ldots, \rho_m, \delta_m)$ is *canonical* if $\rho_1 \geq \cdots \geq \rho_m$. An s-configuration can be made canonical by re-ordering the flowshops.

Let $(k; \rho_1, \delta_1, \ldots, \rho_m, \delta_m)$ be a canonical s-configuration for a schedule $\mathcal{S}_k$ for the job subset G_k. By Lemma 2.1, the R-processors of the flowshops run continuously without idle time, so $\sum_{h=1}^{m} \rho_i \leq R_0$. This gives reduced upper bounds for the completion time of the R-processors of the flowshops:

Fact C. In a canonical s-configuration $(k; \rho_1, \delta_1, \ldots, \rho_m, \delta_m)$, $\forall h$, $\rho_h \leq R_0/h$.

If our objective is to minimize the makespan, then when all the values k, ρ_1, ρ_2, δ_2, …, ρ_m, δ_m are given, we only need to record the smallest δ_1' such that $(k; \rho_1, \delta_1', \rho_2, \delta_2, \ldots, \rho_m, \delta_m)$ corresponds to a valid schedule for G_k. This reduces the number of dimensions for the s-configurations by 1.

In contrast to Fact C, the values δ_h can be very large (recall $T_0 \gg R_0$). We now consider how to deal with the situations when the values δ_h are large.

Fix an h, and consider the h-th flowshop. By Fact C, the completion time of the R-processor of the flowshop can never be larger than $R_0/h \leq R_0$. If the completion time $\rho_h + \delta_h$ of the h-th flowshop is larger than or equal to R_0, then for any further job J_p assigned to the flowshop, the T-operation of J_p can always start immediately when the T-processor is available. Therefore, all further jobs assigned to the flowshop can have their T-operations executed consecutively with no execution gaps in the T-processor of the flowshop. Thus, the completion time of the flowshop will only depend on the T-operations of the further assigned jobs, while it is independent of the R-operations of these jobs. We can use a single value $\rho_h = R_0/h + 1$ to record this situation so that the pair $(R_0/h + 1, \delta_h)$ represents a real status (ρ_h', δ_h') of the flowshop where $\rho_h' + \delta_h' \geq R_0$, and $\delta_h = \rho_h' + \delta_h' - R_0$. Note that when a new job $J_p = (r_p, t_p)$ is added to the flowshop, the corresponding pair of the flowshop is simply changed to $(R_0/h + 1, \delta_h + t_p)$.

Thus, we can represent the status of the h-th flowshop by a pair (ρ_h, δ_h), where either $0 \leq \rho_h \leq R_0/h$ and $0 \leq \rho_h + \delta_h < R_0$ (which implies $0 \leq \delta_h < R_0$), or $\rho_h = R_0/h + 1$ and $0 \leq \delta_h \leq T_0$ (which implies that the completion time for the T-processor is $R_0 + \delta_h$). A pair is *valid* for the flowshop if it satisfies these conditions. The total number of valid pairs for a flowshop is bounded by $(R_0/h+1)R_0 + (T_0+1) = O(R_0^2/h + T_0)$. Note that all valid pairs can be given by a two-dimensional array (i.e., a matrix) with $R_0/h + 2$ rows in which each of the first $R_0/h + 1$ rows contains R_0 elements and the last row contains $T_0 + 1$ elements (if you like, you can also regard this matrix as an $(R_0/h + 1) \times R_0$ matrix plus a one-dimensional array of size $T_0 + 1$).

Summarizing the above discussions, we conclude that all "useful" canonical s-configurations for the job subset G_k, for all k, can be represented by a $(2m)$-dimensional array H' whose elements are $(m + 1)$-tuples, such that if $H'[k; \rho_1, \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (d_k, \delta_1', \rho_2', \ldots, \rho_m')$, where $0 \leq \rho_1 \leq R_0$, and for $2 \leq h \leq m$, (ρ_h, δ_h) is a valid pair for the h-th flowshop, then there is a canonical s-configuration $(k; \rho_1', \delta_1', \rho_2', \delta_2', \ldots, \rho_m', \delta_m')$ for a valid schedule for the job subset G_k, where $\rho_1' = \rho_1$ and δ_1' is the smallest when all other parameters satisfy their conditions, such that for each h, $2 \leq h \leq m$,

(1) if $\rho_h \leq R_0/h$, then $\rho_h + \delta_h < R_0$, $\rho_h' = \rho_h$ and $\delta_h' = \delta_h$, and
(2) if $\rho_h = R_0/h + 1$, then $\rho_h' + \delta_h' \geq R_0$, and $\delta_h = \rho_h' + \delta_h' - R_0$.

Finally the value d_k in the array element, $1 \leq d_k \leq m$, indicates that the last job J_k in the job subset G_k is assigned to the d_k-th flowshop.

Note that in the case $\rho_h = R_0/h + 1$, there can be many different values for ρ_h' that thus correspond to many different canonical s-configurations that satisfy the above conditions. As explained earlier, in this case, different choices of the values ρ_h' will not affect the makespan of the final schedule of the job set G. Thus, we can pick any valid values (not necessarily the smallest) for these ρ_h', as long as their sum plus ρ_1 is equal to $\sum_{i=1}^{k} r_i$.

Since the total number of valid pairs for the h-th flowshop is $O(R_0^2/h + T_0)$, and $k \leq n$, we conclude that the number of elements in the array H' is (note that m is a fixed constant):

$$O((n+1)(R_0+1)\prod_{h=2}^{m}(R_0^2/h + T_0)) = O(n(R_0^{2m-1} + R_0 T_0^{m-1})).$$

Finally, since each element of H' is an $(m+1)$-tuple, we conclude that the array H' takes space $O(n(R_0^{2m-1} + R_0 T_0^{m-1}))$.

We explain how to extend a schedule for G_k to a schedule for G_{k+1} when the job J_{k+1} is added. Let $\mathcal{S}_k = (k; \rho_1', \delta_1', \rho_2', \delta_2', \ldots, \rho_m', \delta_m')$ be a canonical s-configuration for G_k given by the element of the array H':

$$H'[k; \rho_1, \rho_2, \delta_2, \ldots, \rho_m, \delta_m] = (d_k, \delta_1', \rho_2', \ldots, \rho_m'),$$

as explained above. Note that the s-configuration $\mathcal{S}_k$ can be completely re-constructed when the corresponding element of H' is given:

(1) the status (ρ'_1, δ'_1) for the first flowshop is (ρ_1, δ'_1);
(2) for $2 \le h \le m$, (a) if $0 \le \rho_h \le R_0/h$, then the status (ρ'_h, δ'_h) of the h-th flowshop is (ρ_h, δ_h); and (b) if $\rho_h = R_0/h + 1$, then the status(ρ'_h, δ'_h) of the h-th flowshop is $(\rho'_h, R_0 + \delta_h - \rho'_h)$.

Note that in case (2b), what matters is that the completion time of the T-processor is equal to $\rho'_h + (R_0 + \delta_h - \rho'_h) = R_0 + \delta_h$, while the value ρ'_h may vary as long as it satisfies $\rho_1 + \sum_{h=2}^{m} \rho'_h = R_0$.

Now suppose we add the job $J_{k+1} = (r_{k+1}, t_{k+1})$ to the d-th flowshop in the s-configuration $\mathcal{S}_k$. Then the resulting configuration for G_{k+1} will become

$$(k+1; \rho''_1, \delta''_1, \rho''_2, \delta''_2, \ldots, \rho''_m, \delta''_m),$$

where $\rho''_d = \rho'_d + r_{k+1}$ and $\delta''_d = \max\{r_{k+1}, \delta'_d\} + t_{k+1} - r_{k+1}$ (see the explanation given for algorithm **DynProg-I** in the previous section), and for $h \ne d$, $\rho''_h = \rho'_h$ and $\tau''_h = \tau'_h$. This, after properly sorting the flowshops using the values of ρ''_h, becomes a canonical s-configuration

$$\mathcal{S}_{k+1} = (k+1; \bar{\rho}'_1, \bar{\delta}'_1, \bar{\rho}'_2, \bar{\delta}'_2, \ldots, \bar{\rho}'_m, \bar{\delta}'_m)$$

for the job subset G_{k+1}. Assume the d-th flowshop in $\mathcal{S}_k$ becomes the d_{k+1}-th flowshop in $\mathcal{S}_{k+1}$. Now let $\bar{\rho}_1 = \bar{\rho}'_1$, and for each h, $2 \le h \le m$, if $\bar{\rho}'_h + \bar{\delta}'_h < R_0$ then let $\bar{\rho}_h = \bar{\rho}'_h$ and $\bar{\delta}_h = \bar{\delta}'_h$, and if $\bar{\rho}'_h + \bar{\delta}'_h \ge R_0$ then let $\bar{\rho}_h = R_0/h + 1$ and $\bar{\delta}_h = \bar{\rho}'_h + \bar{\delta}'_h - R_0$. With these values, look at the array element

$$H'[k+1; \bar{\rho}_1, \bar{\rho}_2, \bar{\delta}_2, \ldots, \bar{\rho}_m, \bar{\delta}_m].$$

If the element has not been assigned a value, yet, then assign it the value $(d_{k+1}, \bar{\delta}'_1, \bar{\rho}'_2, \ldots, \bar{\rho}'_m)$. If the element already has a value $(d', \bar{\delta}''_1, \bar{\rho}''_2, \ldots, \bar{\rho}''_m)$ but $\bar{\delta}'_1 < \bar{\delta}''_1$, then change its value to $(d_{k+1}, \bar{\delta}'_1, \bar{\rho}'_2, \ldots, \bar{\rho}'_m)$. This completes the process of extending the canonical s-configuration given by the array element $H'[k, \rho_1, \rho_2, \delta_2, \ldots, \rho_m, \delta_m]$, when job J_{k+1} is added to the d-th flowshop, to an array element for a canonical s-configuration for the job subset G_{k+1}. It is easy to see that this process takes time $O(m)$.

Using the above description to replace the steps 3.1–3.7 in the algorithm **DynProg-I** gives the procedure of extending a canonical s-configuration for G_k to a canonical s-configuration for G_{k+1}. This, plus certain obvious modifications in other steps, gives a new algorithm **DynProg-II** for the $P_m|2\text{FL}|C_{\max}$ problem. Since the number of elements of the array H' is bounded by $O(n(R_0^{2m-1} + R_0 T_0^{m-1}))$, we conclude that the time complexity of the algorithm **DynProg-II** is $O(n(R_0^{2m-1} + R_0 T_0^{m-1}))$. Similarly as we explained for the algorithm **DynProg-I**, once we apply algorithm **DynProg-II** and find the array element of H' that gives a minimum makespan schedule of the job set G, we can use the array to construct the actual schedule by backtracking the array in the same amount of time.

Now we describe the job sets G with $T_0 \ll R_0$. Let G^d be the dual job set of G, and let R'_0 and T'_0 be the sums of the times of the R- and T-operations, respectively, of the jobs in G^d. By definition, $R'_0 = T_0$, $T'_0 = R_0$. Therefore, $T'_0 \gg R'_0$. Applying algorithm **DynProg-II** on G^d will construct an optimal schedule $\mathcal{S}^d$ for G^d in time $O(n((R'_0)^{2m-1} + R'_0 (T'_0)^{m-1})) = O(n(T_0^{2m-1} + T_0 R_0^{m-1}))$. By Theorem 2.3, an optimal schedule $\mathcal{S}$ for the job set G can be easily constructed from $\mathcal{S}^d$. In summary, we have

Theorem 4.1. *Optimal schedules for a two-stage job set $\{J_1, \ldots, J_n\}$ on m flowshops, where $J_k = (r_k, t_k)$, can be constructed in time $O(n(T_{\min}^{2m-1} + T_{\min} T_{\max}^{m-1}))$ and space $O(n(T_{\min}^{2m-1} + T_{\min} T_{\max}^{m-1}))$, where $T_{\min}$ and $T_{\max}$ are the smaller and the larger, respectively, of the values $\sum_{k=1}^{n} r_k$ and $\sum_{k=1}^{n} t_k$.*

When $T_{\max} \gg T_{\min}$, Theorem 4.1 provides significant improvements. For example, if $T_{\max} = T_{\min}^2$, then the algorithm given in Theorem 4.1 runs in time $O(nT_{\min}^{2m-1}) = O(nT_{\max}^{m-1/2})$, which almost matches the best pseudo-polynomial time algorithm for the MAKESPAN problem $P_m||C_{\max}$ [12], which can be regarded as a much simpler version of the $P_m|2\text{FL}|C_{\max}$ problem. On the other hand, the time complexity of algorithm **DynProg-I** given in the previous section is of the order $O(nT_{\min}^{3m-3}) = O(nT_{\min}^{m-1} T_{\max}^{m-1})$.

5. Conclusion and future research

Motivated by the research in data centers and cloud computing, we studied the scheduling problem of two-stage jobs on multiple two-stage flowshops, which in particular addresses the scheduling issues of data transmissions between clients and servers in data centers in the cloud framework based on the principle of transparent computing. The problem is NP-hard. Pseudo-polynomial time algorithms for the problem were presented for the case where the number of flowshops is a fixed constant. With thorough analysis, we show that for certain cases, much faster pseudo-polynomial time algorithms can be achieved. Approximation algorithms for the problem were also developed. Our algorithms improve previous known algorithms for the problem.

Needs and considerations in data centers and cloud computing suggest many research topics for future study of our scheduling model.

A cloud computing center may have many servers with different powers, ranging from large mainframe computers to small PC's. The disks connected to the servers and the network bandwidth available for the servers can also differ. Moreover, the disk-read on a server at some moment may not even be needed if the requested data is already in the server's main memory. This calls for the study of scheduling two-stage jobs on heterogeneous two-stage flowshops. Our scheduling model can be easily extended to include this situation: suppose that we need to schedule n jobs $J_1, \ldots, J_n$ on m flowshops $M_1, \ldots, M_m$ that may not be identical, then we can represent each two-stage job J_i by m pairs $\{(r_{i,j}, t_{i,j}) \mid 1 \leq j \leq m\}$, where $(r_{i,j}, t_{i,j})$ gives the R-time and the T-time, respectively, for the job J_i to be processed by the flowshop M_j. Of course, constructing optimal schedules and developing good approximation algorithms on this more general model become more challenging. We are currently working on this extended version of the scheduling model.

A data request from a client can be for a file consisting of data blocks stored in either secondary or main memory. The disk-read and network-transmission of the file can be executed in a pipeline manner in units of data blocks. Thus, once a data block of the file is read into the main memory, the block can be transmitted via networks to the client even if some other data blocks for the file have not been in the main memory, yet. In such a model, preemption of processing data transmissions from servers to clients becomes possible: after transmitting a few data blocks for a file F, a server may switch to processing a different task, and come back later to continue transmitting the remaining data blocks for the file F. In this case, a client request can be given by data size (i.e., the number of data blocks), and can be decomposed into a continuous sequence of two-stage jobs, each corresponding to the disk-read and network-transmission of a data block, with preemptions allowed. However, frequent preemptions should be avoided since restarting disk-read for a file will require new disk search, which is significantly more time-consuming compared to reading a data block. Therefore, when we study scheduling on this model, penalty on preemptions should be considered.

Scheduling with job precedences is very common in cloud computing practice. For example, a user who wants to run a Microsoft application on a transparent computing platform may need from the cloud both the code of the application as well as the code of Microsoft Windows software. However, the application cannot be installed until the Windows software is installed on the client device. As a consequence, there is a need to study the scheduling problems under our model in which job precedence is presented.

Of course, the more general problem $P|2\text{FL}|C_{\max}$, where the number of flowshops is given as part of the input, is also theoretically interesting and challenging, and practically meaningful.

References

[1] A. Allahverdi, C. Ng, T.E. Cheng, M.Y. Kovalyov, A survey of scheduling problems with setup times or costs, European J. Oper. Res. 187 (3) (2008) 985–1032.
[2] J. Bruno, E.G. Coffman Jr, R. Sethi, Scheduling independent tasks to reduce mean finishing time, Commun. ACM 17 (7) (1974) 382–387.
[3] J. Chen, X. Huang, I. Kanj, G. Xia, Polynomial time approximation schemes and parameterized complexity, Discrete Appl. Math. 155 (2007) 180–193.
[4] J. Dong, W. Tong, T. Luo, X. Wang, J. Hu, Y. Xu, G. Lin, An FPTAS for parallel two-stage flowshop problem, Theoret. Comput. Sci. 657 (2017) 64–72.
[5] J. Dong, J. Hu, M. Kovalyov, G. Lin, T. Luo, W. Tong, X. Wang, Y. Xu, Corrigendum to "An FPTAS for the parallel two-stage flowshop problem", Theoret. Comput. Sci. 687 (2017) 93–94.
[6] M.R. Gary, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, WH Freeman and Company, New York, 1979.
[7] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[8] D.W. He, A. Kusiak, A. Artiba, A scheduling problem in glass manufacturing, IIE Trans. 28 (2) (1996) 129–139.
[9] O.H. Ibarra, C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems, J. ACM 22 (4) (1975) 463–468.
[10] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Nav. Res. Logist. Q. 1 (1) (1954) 61–68.
[11] M.Y. Kovalyov, Efficient epsilon-approximation algorithm for minimizing the makespan in a parallel two-stage system, Vestsī Akad. Navuk Belarusī SSR, Ser. Fīz.-Mat. Navuk 3 (1985) 119 (in Russian).
[12] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, Springer Science, New York, 2016.
[13] R. Ruiz, J.A. Vázquez-Rodríguez, The hybrid flow shop scheduling problem, European J. Oper. Res. 205 (1) (2010) 1–18.
[14] D. Shmoys, C. Stein, J. Wein, Improved approximation algorithms for shop scheduling problems, SIAM J. Comput. 23 (3) (1994) 617–632.
[15] R.G.W. Tong, E. Miyano, G. Lin, A PTAS for the multiple parallel identical multi-stage flow-shops to minimize the makespan, in: Lecture Notes in Computer Science, vol. 9711, FAW 2016, 2016, pp. 227–237.
[16] G. Vairaktarakis, M. Elhafsi, The use of flowlines to simplify routing complexity in two-stage flowshops, IIE Trans. 32 (8) (2000) 687–699.
[17] G. Wu, J. Chen, J. Wang, On scheduling two-stage jobs on multiple two-stage flowshops, CoRR, arXiv:1801.09089, 2018.
[18] G. Wu, J. Chen, J. Wang, On scheduling multiple two-stage flowshops, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.04.017.
[19] G. Wu, J. Chen, J. Wang, On scheduling inclined jobs on multiple two-stage flowshops, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.04.005.
[20] G. Wu, J. Chen, Approximation algorithms on multiple two-stage flowshops, in: Lecture Notes in Computer Science, vol. 10976, COCOON 2018, 2018, pp. 713–725.
[21] X. Zhang, S. van de Velde, Approximation algorithms for the parallel flow shop problem, European J. Oper. Res. 216 (3) (2012) 544–552.
[22] Y. Zhang, Y. Zhou, Separating computation and storage with storage virtualization, Comput. Commun. 34 (13) (2011) 1539–1548.
[23] Y. Zhang, Y. Zhou, TransOS: a transparent computing-based operating system for the cloud, Int. J. Cloud Comput. 1 (4) (2012) 287–301.