Computers & Operations Research 51 (2014) 251–256

Contents lists available at ScienceDirect

Computers & Operations Research

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

A best possible on-line algorithm for two-machine flow shop scheduling to minimize makespan ☆

Peihai Liu *, Xiwen Lu

Department of Mathematics, School of Science, East China University of Science and Technology, Shanghai 200237, People's Republic of China

ARTICLE INFO

Available online 21 June 2014

Keywords:
Scheduling
Flow shop
Online
Competitive ratio

ABSTRACT

We address a two-machine flow shop on-line scheduling problem. Jobs arrive over time. Each job becomes available for processing at its release time after which it must be processed without preemption on the first machine and then on the second machine. The objective is to minimize the makespan. We provide a best possible deterministic on-line algorithm with a competitive ratio of $(\sqrt{5}+1)/2$. Computational experiments on randomly generated problem instances are performed and the results show that the online algorithm is very useful to obtain near-optimal solutions.

1. Introduction

We consider the following problem. There are two machines M_A and M_B, and a set of jobs $J=\{J_1,J_2,\ldots,J_n\}$, where n is not known in advance. Each job $J_j \in J$ consists of two successive operations O_A^j and O_B^j, where O_A^j is processed first for a_j time units on M_A, and then O_B^j is processed for b_j time units on M_B. For convenience, we call a_j A-processing time and b_j B-processing time. Jobs arrive over time. Each job $J_j \in J$ becomes available for processing at its nonnegative release time r_j. At any time, each machine can handle only one job and each job can be processed on only one machine. The objective is to minimize the makespan, i.e., the maximum completion time of jobs on M_B. Preemption is not allowed. Using the problem classification of Lawler et al. [5], our problem is written as $F2|r_j, \text{on}-\text{line}|C_{max}$.

The quality of an on-line algorithm is typically assessed by its competitive ratio: the nearer the ratio approaches 1, the better the algorithm is. We say that an algorithm is ρ-competitive if for any input instance, it always returns a feasible solution with an objective value not greater than ρ times the optimal (off-line) solution.

The two-machine flow shop problem with the objective of minimizing the makespan has been discussed in the operation research literature for several decades. In the special case when all r_j are equal, the well-known algorithm of Johnson [3] solves the problem $F2||C_{max}$ in $O(n \log n)$ time. In the general case when the release times may be different, the problem $F2|r_j|C_{max}$ is shown to be NP-hard by Lenstra et al. [6] in the strong sense.

Four heuristics are described by Potts [8] for $F2|r_j|C_{max}$. He shows that for three of them, the worst-case ratio is 2, and each has a running time of $O(n \log n)$. The fourth heuristic, which is based on the iterative use of the third one, has a better worst-case ratio of 5/3, and a time requirement of $O(n^3 \log n)$. Additionally, a polynomial time approximation scheme (PTAS) is designed by Hall [2], i.e., a family of algorithms that, given an arbitrary small but fixed $\varepsilon > 0$, finds a schedule in polynomial time and has a worst-case ratio of at most $1+\epsilon$. Kashyrskikh et al. [4] give an 1.5-approximation algorithm based on the fourth heuristic of Potts [8].

For the on-line open shop scheduling on two machines, Chen et al. [1] show that the greedy algorithm achieves the competitive ratio of 3/2 and this is optimal for scheduling without preemptions. When preemption is allowed, a 5/4 competitive algorithm exists and this is optimal [1]. For the on-line flow shop scheduling on two machines, Vestjens [10] shows that any deterministic on-line algorithm must have a competitive ratio of at least $(1+\sqrt{5})/2$.

In this paper, we study on-line version of the two-machine flow shop scheduling. We present a deterministic on-line algorithm and show that it is best possible.

2. Preliminaries

Throughout the paper we use the following notations:

- $a(\mathcal{J})$, the total A-processing time of all jobs in $\mathcal{J}$;
- $b(\mathcal{J})$, the total B-processing time of all jobs in $\mathcal{J}$;
- $S_j^o(s)$, $o \in \{A,B\}$, the starting time of operation O_o^j, $o \in \{A,B\}$ in schedule s;

☆This work was supported by the National Nature Science Foundation of China (11101147 and 11371137) and the Fundamental Research Funds for the Central Universities.

* Corresponding author.
E-mail addresses: pliu@ecust.edu.cn (P. Liu), xwlu@ecust.edu.cn (X. Lu).

http://dx.doi.org/10.1016/j.cor.2014.06.014
0305-0548/

- $C_j^o(s)$, $o \in \{A,B\}$, the completion time of operation O_o^j, $o \in \{A,B\}$ in schedule s;
- $S(\mathcal{J},s)$, the minimum starting time of jobs in $\mathcal{J}$ in schedule s;
- $C_{max}(s)$, the makespan, i.e., the objective value of schedule s.

Without causing any confusion, we also write for short $S_j(s)$ instead of $S_j^A(s)$ as the starting time of J_j on M_A in schedule s and write for short $C_j(s)$ instead of $C_j^B(s)$ as the completion time of J_j on M_B in schedule s.

Some lower bounds on the offline optimal makespan, which are needed in the subsequent analysis, are introduced. Let π be an offline optimal schedule. Given a job set $\mathcal{J}$, an obvious lower bound based on the total A-processing times is

$$C_{max}(\pi) \geq S(\mathcal{J},\pi)+a(\mathcal{J}) \tag{1}$$

It is well known that there exists an optimal schedule solution that is a permutation schedule in which the same job order is used on both machines. Hence we only consider schedules in which the same job order is used on both machines.

In general, each job J_i has a set of predecessors which are jobs that are known to be sequenced before J_i in an optimal schedule and a set of successors which are jobs that are known to be sequenced after J_i in an optimal schedule. Let $\mathcal{J}_1$ be the set of all predecessors in $\mathcal{J}$ of J_i and $\mathcal{J}_2$ be the set of all successors in $\mathcal{J}$ of J_i. Thus all operations O_A^j, $J_j \in \mathcal{J}_1$ are sequenced before O_A^i and all operations O_B^j, $J_j \in \mathcal{J}_2$ are sequenced after O_B^i. Therefore, a more complicated lower bound is

$$C_{max}(\pi) \geq S(\mathcal{J},\pi)+a(\mathcal{J}_1)+a_i+b_i+b(\mathcal{J}_2) \tag{2}$$

Here we refer J_i as a partition job since J_i partitions $\mathcal{J}\backslash\{J_i\}$ into two subsets $\mathcal{J}_1$ and $\mathcal{J}_2$ where all jobs in $\mathcal{J}_1$ are represented as O_A^j and all jobs in $\mathcal{J}_2$ are represented as O_B^j in the lower bound on the optimal makespan $C_{max}(\pi)$.

3. An on-line algorithm

In this section, we deal with the on-line two-machine flow shop scheduling problem. We first present a deterministic on-line algorithm and then show the algorithm is $(1+\alpha)$-competitive, where $\alpha=(\sqrt{5}-1)/2\approx 0.618$. For convenience, we shall call the jobs with $a_j \leq (1+\alpha)b_j$ A-small jobs and call the jobs with $a_j > (1+\alpha)b_j$ B-small jobs.

An online scheduling rule chooses a job to schedule at time t, or may choose to leave the machine idle, without using any information from jobs j with $r_j > t$. If an algorithm wants to guarantee a better performance bound, then it needs both a waiting strategy and a selection strategy. A waiting strategy sometimes allows the algorithm to wait for more information. A selection strategy will tell the algorithm how to select a job among the available jobs. The basic idea behind the algorithm is that, if one of the available jobs is an A-small job, then we schedule the job with the smallest A-processing time among the available A-small jobs; otherwise, we decide whether to schedule the job with the largest A-processing time, a job other than the job with the largest A-processing time, or no job at all.

In the following heuristic, the following notations are used.

- $U(t)$ denotes the set containing all jobs that have arrived at or before time t and that have not been started by time t;
- $Asmall(t)$ indicates the set of A-small jobs in $U(t)$;
- $Bsmall(t)$ indicates the set of B-small jobs in $U(t)$;
- $p(t)$ denotes the index of the job with the largest A-processing time in $U(t)$;
- $|\mathcal{J}|$ denotes the number of the jobs in the corresponding set $\mathcal{J}$.

The set $U(t)$ can be interpreted as the set of jobs that still need to be entirely processed at time t.

Algorithm *H*. *Processing strategy for machine A*:

Step 0. If M_A is idle and a job is available at time t, determine $Asmall(t)$ and $Bsmall(t)$. Otherwise, wait until the machine is idle and a job is available.

Step 1. If $Asmall(t) \neq \emptyset$, select a job with the smallest A-processing time in $Asmall(t)$ and schedule it on M_A.

Step 2. If $Asmall(t) = \emptyset$ and $|Bsmall(t)| > 1$, then

Step 2A. Schedule $J_{p(t)}$ on M_A if $t+a_j > r_{p(t)}+\alpha a_{p(t)}+(1+\alpha)b_j$ for all jobs J_j other than $J_{p(t)}$ in $Bsmall(t)$.

Step 2B. Schedule J_j on M_A if $t+a_j \leq r_{p(t)}+\alpha a_{p(t)}+(1+\alpha)b_j$ for some job J_j other than $J_{p(t)}$ in $Bsmall(t)$.

Step 3. If $Asmall(t) = \emptyset$ and $|Bsmall(t)| = 1$, then

Step 3A. Schedule $J_{p(t)}$ if $t \geq r_{p(t)}+\alpha a_{p(t)}$;

Step 3B. Wait to the next arrival or the time $r_{p(t)}+\alpha a_{p(t)}$ if $t < r_{p(t)}+\alpha a_{p(t)}$.

Step 4. Goto Step 0.

Processing strategy for machine B:

Start jobs on M_B in order of their nondecreasing completion time on M_A whenever M_B is idle.

According to the algorithm, we know all A-small jobs are assigned by Step 1 and all B-small jobs are scheduled by Steps 2 and 3. Note that if only one B-small job J_j is available and there is no available A-small jobs, then the algorithm schedules no job and waits either until time $r_j+\alpha p_j$ or until a new job arrives, whichever happens first. This means that at time t, if M_A is not busy, then there is at most one available job and the job is a B-small job if it exists. For the job J_j scheduled by Step 2B at time t, we can find that it starts before $r_{p(t)}+\alpha a_{p(t)}$ and its remaining A-processing time at time $r_{p(t)}+\alpha a_{p(t)}$ is not greater than $(1+\alpha)b_j$.

Let σ be the schedule generated by the Algorithm H and π be an optimal schedule. Without loss of generality, we index jobs with $J_j(1 \leq j \leq n)$ such that $S_1(\sigma) \leq \cdots \leq S_n(\sigma)$. Similar to Potts [8], the corresponding maximum completion time of σ can be written as (shown in Fig. 1)

$$C_{max}(\sigma) = S_u(\sigma)+\sum_{i=u}^{v} a_i+\sum_{i=v}^{n} b_i, \tag{3}$$

for some $u,v \in \{1,2,\ldots,n\}$, where $u \leq v$ and u is chosen as small as possible. Then, J_u is the first starting job at time 0 or starts exactly after an idle interval on M_A in σ. It is clear that the completion time of operation O_A^v coincides with the starting time of operation O_B^v. Similar to Kashyrskikh et al. [4], we refer to J_v as a transition job in schedule σ, since the critical path passes via the operations of this job from M_A to M_B.

By the algorithm, we know that M_A can be idle at time t only when no available jobs or there is only one available job and the job is a B-small job at t. Thus we have the following claim.

Claim 1. *Among jobs in* $\{J_j | u \leq j \leq n\}$, *there can be at most one job which is released before* $S_u(\sigma)$.

Claim 2. *If* J_j *is an A-small job and* $S_j(\sigma) \geq S_u(\sigma)$, *then* $r_j \geq S_u(\sigma)$.

Proof. From the definition of u, we know that $S_u(\sigma)$ starts at time 0 or starts exactly after an idle interval on M_A in σ.

If J_u starts at time 0, i.e., $S_u(\sigma)=0$, then the conclusion obviously holds.