Available online at www.sciencedirect.com

ScienceDirect

Computers & Operations Research 35 (2008) 525–537

computers & operations research

www.elsevier.com/locate/cor

Polynomial time algorithms for the UET permutation flowshop problem with time delays

Alix Munier-Kordon[a,*], Djamal Rebaine[b]

[a]Laboratoire LIP6, Université Pierre et Marie Curie, 4 place Jussieu, 75 252 Paris, Cedex 05, France
[b]Dépt d'informatique et de mathématique, Université du Québec à Chicoutimi, Que., Canada G7H 2B1

Available online 23 June 2006

Abstract

This paper addresses the problem of scheduling a set of n unit execution time (UET) jobs on an m-permutation flowshop with arbitrary time delays, so as to minimize the makespan criterion. A polynomial time algorithm is exhibited for the three-machine and four-machine cases, respectively.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Complexity; Makespan; Permutation flowshop; Scheduling; Time delays

1. Introduction

The flowshop scheduling problem with time delays can be described as follows. Given are a set $J = \{1, \dots, n\}$ of jobs and a set of m machines. Any job $i \in J$ comprises m operations, $o_{i1}, \dots, o_{im}$, each of which has a unit execution time (UET). Each job is first processed on machine 1, then on machine 2, and so on until it completes its last operation on machine m. Moreover, for any $j \in \{1, \dots, m-1\}$, there is a time delay between operations o_{ij} and o_{ij+1}, denoted by d_i^j. If $t(o_{ij})$ denotes the starting time of o_{ij}, then a valid schedule is such that

$$\forall i \in \{1, \dots, n\}, \ \ \forall j \in \{1, \dots, m-1\}, \quad t(o_{ij}) + 1 + d_i^j \leqslant t(o_{ij+1}).$$

For each job $i \in \{1, \dots, n\}$, there is an associated delay vector $D_i = (d_i^1, \dots, d_i^{m-1})$.

In this paper, we focus on a subclass of flowshop scheduling, namely the permutation schedules. In this subclass, the order in which each machine processes the jobs is identical for all machines. In other words, job overpassing is not permitted. A permutation schedule is thus given by a bijection $\sigma : J \to \{1, \dots, n\}$ such that, for any job $i \in J$, $\sigma(i)$ denotes the position of job i in σ. We seek a permutation schedule, σ, which minimizes the overall completion time, called the *makespan*.

Motivation for the formulated problem is twofold: theoretical and practical. The first goal is to explore the borderline between easy and hard cases of the problem under study. Regarding the second goal, in the traditional flowshop scheduling, it is a common practice to assume that once a job has finished its execution on any machine, it becomes

* Corresponding author.
E-mail addresses: Alix.Munier@lip6.fr (A. Munier-Kordon), drebaine@uqac.ca (D. Rebaine).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.03.018

immediately available for further processing. However, in many practical applications, this assumption is not justified. Indeed, there is often a significant time delay between the completion of an operation and the beginning of the next operation of the same job. Time delays may be attributed, for example, to transportation times of the jobs on the machines or, in some other applications, to the different times needed by the drying processes of the jobs before they can be handled by another processing stage. In some applications, the processing times might even be negligible compared to the time delays: assuming in this case that the processing times of the jobs are unitary, and therefore have a small influence on the makespan of the schedule.

The restriction to permutation flowshops is also a common assumption as it simplifies the structure of the solutions and the proofs (see e.g. [1]). From a practical point of view, the schedule obtained has a simple structure and thus may be easily implemented. Moreover, the raw materials between the machines remain bounded and the intermediate storage capacities may be limited (even though it is not the primary objective). This leads to good practical solutions.

Let us first recall that, when the time delays are ignored, the permutation flowshop schedules are dominant for $m \leqslant 3$. Johnson [2] showed that the two-machine problem is solvable in $O(n \log n)$. The problem becomes $\mathcal{NP}$-hard in the strong sense for any fixed $m \geqslant 3$ [3]. When the time delays are considered, to the best of our knowledge, the complexity status of the UET problem we are considering in this paper is unknown, but several authors worked on closely related problems. Mitten [4] showed that the two-machine permutation flowshop, with time delays, is still solvable in $O(n \log n)$. It is interesting to note that, for the UET case, this problem can be easily solved in $O(n)$ since any permutation is optimal. If both minimal and maximal time delays are taken into account simultaneously, then Frondevelle et al. [5] proved that the two-machine permutation flowshop problem becomes strongly $\mathcal{NP}$-hard. Yu et al. [6] showed that minimizing the makespan in a UET two-machine flowshop problem with time delays is strongly $\mathcal{NP}$-hard. Let us observe that permutation flowshops with time delays are clearly not dominant, even for the $m = 2$ case and UET jobs. Indeed, Rebaine [7] showed that the best UET permutation schedule is worse than that of the best UET flowshop schedule by a factor of m. However, special cases exist where permutation schedules are still dominant (see e.g. [8]).

This paper is devoted to the permutation flowshop problem with UET jobs and arbitrary time delays. It is organized as follows. Section 2 presents some definitions and preliminary results. In Section 3, an $O(n \log n)$ algorithm is presented to solve the three-machine case. Section 4 discusses the four-machine case, and presents an $O(n^2)$ algorithm. Section 5 is our conclusion.

2. Preliminaries

We begin by introducing definitions and results necessary for the following sections. Before proceeding further, let us first present the following simple property of a permutation schedule.

Lemma 1. *Any given permutation schedule can be converted into another permutation schedule of the same length such that jobs of equal delay vectors are scheduled one after the other.*

Proof. The lemma can be easily proved by a repeated exchange argument. □

From the above lemma, we can therefore assume without loss of generality, in the remainder of this paper, that the delay vectors are all different: for any $(i, j) \in J \times J$ with $i \neq j$, we have $D_i \neq D_j$.

For the sake of clarity, we recall that, in what follows, a node of a graph refers to a job.

Definition 1. Given a precedence graph, if there exists a path from node x to node y, then x is a predecessor or an ancestor of y, and y is a successor or a descendant of x.

Definition 2. Given a precedence graph, a node is initial if it does not have any predecessor.

For a given machine $j \in \{2, \dots, m\}$ and a permutation schedule σ, we denote by $s_j(\sigma)$ the total number of idle slots on machine j before the last operation o_{nj}, Its value is given by the following result. Observe that the makespan of the permutation schedule σ is $C_{\max}(\sigma) = s_m(\sigma) + n$.