Discrete Applied Mathematics 160 (2012) 1924–1935

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

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

A mixed integer programming model for the cyclic job-shop problem with transportation

Peter Brucker [a], Edmund K. Burke [b], Sven Groenemeyer [b,*]

[a] Universität Osnabrück, Fachbereich Mathematik/Informatik, 49069 Osnabrück, Germany

[b] University of Nottingham, School of Computer Science, Jubilee Campus, Nottingham, NG8 1BB, United Kingdom

ARTICLE INFO

Article history:
Received 26 July 2011
Received in revised form 31 October 2011
Accepted 2 April 2012
Available online 7 May 2012

Keywords:
Cyclic job-shop
Transport
Blocking
Integer programming
Minimal cycle time

ABSTRACT

This paper focuses on the study of cyclic job-shop problems with transportation and blocking. Within this domain, there are many real world problems like large scale productions, robotic cells, software pipelining or hoist scheduling. The aim in general is to find, for each machine, a feasible order of all the operations processed on this machine, so that an objective function is optimised. In this paper, we consider the problem of minimising the cycle time (maximising the throughput) in a job-shop environment, where the jobs are transported by a single robot between the machines. Additionally to the problem description, we will give some explanations and interpretation possibilities of the problem *height*, which is often omitted in the literature. As the main contribution, we will present a new integer programming formulation and show that it outperforms an existing model from the literature.

1. Introduction and problem definition

The classical job-shop problem is known as a standard problem in scheduling and has been widely investigated over the last few decades. However, in practice, scheduling problems often cannot be modelled as a classical job-shop problem since other real world constraints usually have a major influence. This is especially the case in industrial production environments where aspects such as material handling, storage space and machine setup times have to be taken into account.

The problem studied in this paper is a cyclic job-shop problem with one transport robot and blocking (CJSPT) and can be formulated as follows. We are given a set of N jobs $J_1, J_2, \ldots, J_N$. Each job J_j consists of a set of n_j operations which have to be processed in a prescribed order (precedence constraints). To simplify the notation, we consecutively number all operations in the form $i = 1, 2, \ldots, n$ with $n = \sum_{j=1}^{N} n_j$. This means, that the operations $1, \ldots, n_1$ belong to job J_1, $n_1 + 1, \ldots, n_1 + n_2$ belong to J_2, and so on. The set of all operations is denoted by Ω. Let $J(i)$ be the job operation $i \in \Omega$ belongs to and let $\mathrm{pre}(i) \in \Omega$ (respectively $\mathrm{suc}(i)$) be the preceding (respectively succeeding) operation of i according to the precedence constraints. Furthermore, we assume that preemption is not allowed. The predecessor of the first operation and the successor of the last operation of every job are always dummy operations with processing times equal to 0.

Every operation has to be processed on one specified machine out of m machines $M_1, \ldots, M_m$. The machine, operation $i \in \Omega$ will be processed on, is denoted by $M(i)$. If two succeeding operations of the same job are processed on the same machine, we simply combine those two operations into one. Also, each machine has no buffer and can only process one operation at a time. Furthermore, let $p_i^{\min} > 0$ be the minimum processing time of operation $i \in \Omega$ and p_i the "actual" time

* Corresponding author.
E-mail addresses: pbrucker@uni-osnabrueck.de (P. Brucker), ekb@cs.nott.ac.uk (E.K. Burke), svg@cs.nott.ac.uk (S. Groenemeyer).

0166-218X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2012.04.001

a job stays on a machine (e.g. because of waiting until the next machine is ready). Moreover, there is an input station M_0 that stores the unprocessed jobs and an output station $M_\star$ that stores the finished jobs. Both stations have infinite buffers.

In modern fully automated processing lines, a single transport robot is in charge of carrying the jobs between the machines. If operation pre(i) has finished its processing on a machine, then the robot has to unload the job, transport it to its next machine $M(i)$ and load the job onto that machine. This *transport move* is denoted by τ_i and the time needed to execute this task by t_i. For the last operation of any job J_j, we introduce a transport move $\tau_{\star j}$ that unloads the completed job off its last machine and transports it to the output station, $M_\star$. The union of all operations $i \in \Omega$ and those successor dummy operations of the last operations $\{\star^1, \ldots, \star^N\}$ is denoted by $\Omega^\star$. If the robot, after loading a job onto machine $M(i)$, is not waiting for this job, but moving empty to another machine $M(j)$ to unload a different job, then this *empty moving time* is denoted by $e_{M(i)M(j)}$ or simply e_{ij}. We assume that the triangle inequality $e_{ij} + e_{jk} \geq e_{ik}$ holds for the empty moving times between any three machines $M(i)$, $M(j)$ and $M(k)$. The empty moving time between the same machine is $e_{ij} = 0$ for $M(i) = M(j)$. Note that a transport move and an empty move between the same two machines do not need to have the same duration. Since a transport time t_i also includes the loading and unloading process of the job it usually holds that $t_i > e_{\text{pre}(i),i}$.

In a cyclic scheduling problem, all jobs are processed indefinitely often. Since the output of the finished jobs should usually be spread evenly we only plan a minimal part set (MPS) of all jobs and repeat this production schedule all the time. The time difference between the *starting times* S_i of two succeeding repetitions of the same operation i is called the *cycle time*. We call a schedule *cyclic*, with cycle time $\alpha \geq 0$, if for each operation i the following holds. For every α time units, after a repetition of i has been started, the next repetition of i starts its processing. This means that the time between two consecutive processings of the same operation is α. To distinguish between the different repetitions of each operation we denote the starting time of the r-th repetition $\langle i, r\rangle$ of operation $i \in \Omega^\star$ by $S_i(r)$ where $r \in \mathbb{Z}$ is called the *repetition number*. A schedule S can be represented by a vector $S = (S_i(r_i))$ including the starting times of each operation and their repetition numbers in an arbitrary time interval of length α (*cycle*). That means every operation starts and finishes exactly once in each cycle. The difference between cyclic schedules and non-cyclic ones is that an operation that starts in one cycle can finish in the next cycle. We call such an operation *overlapping* since it overlaps into the next cycle. The final schedule usually becomes more compact compared to the non-cyclic solution (cf. a more detailed exemplification in Example 1.1).

Formalising the previous description, a schedule is called *cyclic* with *cycle time* $\alpha \geq 0$ if

$$S_i(r) = S_i(0) + \alpha r, \tag{1.1}$$

for all $i \in \Omega^\star$, $r \in \mathbb{Z}$. Note that there are indefinitely many schedules that have the same cycle time, but different starting times for the operations. For a given cyclic schedule S, another schedule S' can be obtained by setting $S'_i = S_i + \varepsilon \mod \alpha$ for all $i \in \Omega^\star$ and $\varepsilon \in \mathbb{R}$. Hence, we assume without loss of generality that a cycle starts with the dummy start operation at time 0:

$$S_0(0) = 0 \tag{1.2}$$

$$S_1(0) = t_1. \tag{1.3}$$

Note that this constraint also implies that at the end of the cycle, the last move the robot executes is driving empty to the input station M_0 to be ready for unloading another instance of J_1. Additionally, we assume that a job immediately starts its processing after it has been loaded onto a machine. This convention is called the *no-wait constraint* and can be formulated as

$$S_i(r) + p_i + t_{\text{suc}(i)} = S_{\text{suc}(i)}(r), \tag{1.4}$$

for all $i \in \Omega;\ r \in \mathbb{Z}$. For the actual processing time p_i, which corresponds to the duration a job stays at a machine it holds that

$$p_i^{\min} \leq p_i, \tag{1.5}$$

for all $i \in \Omega$. Constraints (1.4) and (1.5) also ensure that an operation has to be processed at least for its minimal processing time before its succeeding operation can start.

We also postulate that the $(r+1)$-th repetition of operation $i \in \Omega$ cannot start before the r-th repetition of it has been finished and transported to the next machine $M(\text{suc}(i))$. After that, the robot has to move to $M(\text{pre}(i))$ and repeat the transport move τ_i. Therefore, we get the following constraint:

$$S_{\text{suc}(i)}(r) + e_{\text{suc}(i),\text{pre}(i)} + t_i \leq S_i(r+1), \tag{1.6}$$

for all $i \in \Omega$, $r \in \mathbb{Z}$. For $i \in \{\star^1, \ldots, \star^N\}$ constraint (1.6) changes to

$$S_i(r) + e_{i,\text{pre}(i)} + t_i \leq S_i(r+1), \tag{1.7}$$

for all $r \in \mathbb{Z}$.

Now consider two operations i, j with $M(i) \neq M(j)$. As there exists no storage at the machines, operation j can only start its processing after $J(j)$ has been loaded onto the machine. If i starts its processing immediately before j, the robot has to finish the loading of job $J(i)$, drive empty to the machine on which the predecessor pre(j) of operation j is processed, unload