View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

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.

© 2012 Elsevier B.V. All rights reserved.

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:

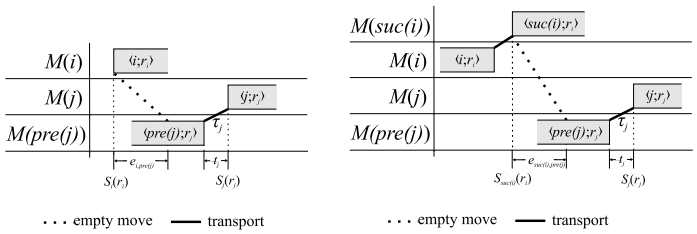
$$S_{\text{suc}(i)}(r) + e_{\text{suc}(i),\text{pre}(i)} + t_i \leq S_i(r+1), \tag{1.6}$$

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



(a) Constraints (1.8). (b) Constraints (1.9).

Fig. 1. Examples for constraints (1.8) and (1.9).

the job, transport it to $M(j)$ and load it onto that machine. Depending on the order of processing the r_i-th repetition of i and the r_j-th repetition of j, we have

$$\begin{aligned} &S_i(r_i) + e_{i,\mathrm{pre}(j)} + t_j \le S_j(r_j) \\ &\text{or } S_j(r_j) + e_{j,\mathrm{pre}(i)} + t_i \le S_i(r_i), \end{aligned} \tag{1.8}$$

for all $i, j \in \Omega^\star$ and $r_i, r_j \in \mathbb{Z}$ with $M(i) \neq M(j)$.

For two operations $i, j \in \Omega$ that have to be processed on the same machine $M(i) = M(j)$ we have to decide which job has to be processed first on the machine. Let us assume that i will be processed before j. Because of the blocking situation it follows, that after the processing of $J(i)$ has been finished the robot first has to transport $J(i)$ to the next machine before it can drive empty to $M(\mathrm{pre}(j))$ to transport $J(j)$ to $M(j)$. Otherwise, $M(i)$ would be blocked by $J(i)$ and we would have a deadlock situation. Therefore, one of the following constraints must hold:

$$\begin{aligned} &S_{\mathrm{suc}(i)}(r_i) + e_{\mathrm{suc}(i),\mathrm{pre}(j)} + t_j \le S_j(r_j) \\ &\text{or } S_{\mathrm{suc}(j)}(r_j) + e_{\mathrm{suc}(j),\mathrm{pre}(i)} + t_i \le S_i(r_i), \end{aligned} \tag{1.9}$$

for all $i, j \in \Omega$ and $r_i, r_j \in \mathbb{Z}$ with $i \neq j$; $M(i) = M(j)$. Fig. 1 shows some examples of how the last two constraints can be interpreted.

We briefly want to recall, why constraints (1.1)–(1.9) are not just necessary but also sufficient to define our problem. The question is, if a schedule, that fulfils these constraints is also a feasible one. First of all, the minimal processing and transportation times have to maintained. The processing times are trivially fulfilled by (1.4) and (1.5). The robot also has always sufficient time to drive to a machine, pick up a job and transport it to its next machine, and a job does not start before it has been transported to its machine. This is given by constraints (1.8) and (1.9). The latter one also ensures, that a machine is always free before another job will be loaded onto it, so no two jobs can overlap. Finally, it is not possible to miss out a repetition of a job, since (1.6) and (1.7) are satisfied.

In addition to the cycle time α, another important value of a cyclic job-shop problem is the *flow time* of a job. It is defined as the time it takes to process a specific instance of a job, from the start of its first operation until the end of its last operation. In general, the cycle time and the flow time are negatively correlated. That means, a smaller cycle time tends to lead to a larger flow time and vice versa. In practice, the flow time is a substantial measure. This is especially the case if there are deadlines on the costumers side where a specific instance of a job has to be finished by a certain due date. The variable which builds the connection between the flow time and the cycle time is the *height* h_j of job J_j. It is defined as the number of cycles that a specific instance of a job will stay in the system from start to finish. For a non-cyclic problem, in which minimising the cycle time is equivalent to minimising the makespan, the height for every job would be 1, since there only is one instance of each job that has to be processed. If a specific instance of job J_i has started its processing in the r-th cycle then it will be finished in cycle $r + h_j - 1$. The value $h_j - 1$ is also the number of overlapping operations job J_j has. For every overlapping operation the job instances enter another cycle. If the height is 1 then the job starts and finishes in the same cycle, which means it has no overlapping operation. At the same time, the height is the number of different repetitions of operations belonging to the same job J_j in one cycle. Since an overlapping operation starts in a cycle but finishes in the next one, the previous repetition of this operation must finish in the current cycle.

The maximum of all those heights $h = \max_{J_j \in J}\{h_j\}$ builds the overall height of the problem. Note that the height of a job is at least 1 and cannot be larger than the number of its operations, so $1 \le h_j \le n_j$ (cf. constraints (1.6), (1.7)).

To keep the production more flexible in practice, the height is limited to a maximum value. This limitation also has a strong influence on the number of possible cyclic schedules. There are three main types of models in the literature. They differ in the definition of when a new instance of a job is allowed to start its processing in the current cycle. One, that has been introduced in [3], is called the cyclic job-shop problem with job repetition. In this model it holds that the $(r + h)$-th repetition of a job can only start after the r-th occurrence of the last operation of the job has been finished. The second model is the called cyclic job-shop problem with machine repetition. It was introduced first by Hitz [12]. Here, the height is defined per machine. It holds that there cannot be more then h different repetitions of operations on each machine in a cycle. The model and a corresponding MIP formulation that we will present in the next chapter is derived from Hanen [10] and is also

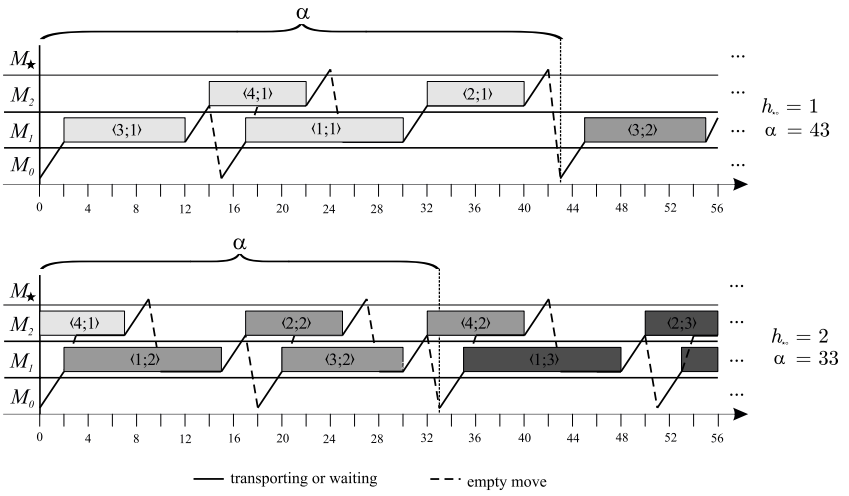


Fig. 2. Gantt-charts for Example 1.1.

used in [3]. Within this model, the height limits the number of different repetitions of all jobs in the same cycle. That means the r-th repetition of every job has to be finished before the $(r + h)$-th repetition of any other job can start. Therefore, it holds that

$$S_{\star j}(r) + e_{\star j,0} \leq S_0(r + h), \tag{1.10}$$

for all $j \in \{1, \ldots, N\}$. Note that the h value has to be given in advance.

The aim now is to minimise the cycle time α so that all constraints (1.1)–(1.10) are satisfied.

To communicate a better understanding of cyclic schedules and especially the influence of the height we briefly present a short example.

Example 1.1. Consider a cyclic job-shop problem with $N = 2$ jobs and $m = 2$ machines. Both jobs consist of two operations. The processing times and the machine allocations are given in the following table.

Job	J_1		J_2	
Operation	1	2	3	4
Processing time	13	8	10	8
Machine	M_1	M_2	M_1	M_2

Additionally, we have one transport robot. The transport time for each operation is equal to 2 and empty moving times between any two machines is 1. By setting the height $h_{\star\circ}$ to 1, the minimal cycle time is $\alpha = 43$. Increasing the height to 2, the cycle time decreases to $\alpha = 33$. Possible schedules for these solutions can be found in Fig. 2. In the second Gantt chart operation 4 is an example for an overlapping operation. The second repetition of it starts at time 32 in the first shown cycle and finishes at time 40 in the following cycle. As you can see the first repetition of operation 4 also overlaps and finishes in the first cycle but has started in the previous one that is not shown.

Cyclic scheduling problems have, in addition to mass production in flexible manufacturing systems, other applications, like compilation of loops for parallel computers, hoist routing in electroplating facilities, the design of embedded architectures or network scheduling. Different models have been proposed to handle those kinds of problem. Trouillet et al. [19] and Chrétienne [6] investigated the use of petri nets whereas Baccelli et al. [1] and Cohen et al. [7] applied Max-plus algebras to the problem. Probably the most common approach, and also the one we will apply in this paper, is modelling the problem using graphs (cf. [14,11]). A good overview on cyclic scheduling can be found in [17].

For the general cyclic job-shop problem with blocking Brucker and Kampmeyer [5] developed a tabu search algorithm and presented computational results. The same problem with an additional no-wait constraint is studied by Hall and Sriskandarajah [9]. Kamoun and Sriskandarajah [13] review algorithmic and complexity issues for this problem. In [18], an exact method for the special case with only one single job is presented. McCormick et al. [15] considered the problem with blocking constraints and limited buffers where blocking occurs when buffers are full. They propose heuristic approaches to this problem based on an equivalent maximum flow problem and critical path techniques.

This paper is structured as follows. In the next section, we will adapt a formulation from the literature to model the CJSPT and prove that it is equivalent to the definition provided in this section. After that, in Section 3, we characterise the layout of cyclic schedules and present a new more tailored integer programming formulation for the CJSPT based on overlapping

operations in a cycle. Both formulations are compared against each other to solve several different sized instances using CPLEX. The computational results are shown in Section 4. The aim of the paper is to not just to present a better linear programming formulation but to give a new idea of modelling cyclic scheduling problems.

2. A mixed integer programming model from the literature

Since integer programming software (such as CPLEX or Gurobi) is becoming more and more powerful to solve reasonably large problem instances it is useful to have integer programming formulations for a problem. The following formulation is based on the work of Hanen [10] and Brucker and Kampmeyer [4]. We used their ideas to model the cyclic disjunctive cyclic constraints and reformulated our problem definition from Section 1 with it.

Theorem 2.1. *By setting $S_i := S_i(0)$ problem (1.1)–(1.10) can be reformulated to the following mixed integer linear program.*

$$\min \alpha \tag{2.1}$$

s.t.

$$S_0 = 0 \tag{2.2}$$

$$S_1 = t_1 \tag{2.3}$$

$$S_i + p_i + t_{\mathrm{suc}(i)} = S_{\mathrm{suc}(i)} \quad i \in \Omega \tag{2.4}$$

$$p_i^{\min} \leq p_i \quad i \in \Omega \tag{2.5}$$

$$S_i + e_{i,\mathrm{pre}(j)} + t_j \leq S_j + \alpha HX_{ij} \quad i,j \in \Omega^\star;\ i \neq j;\ M(i) \neq M(j) \tag{2.6}$$

$$HX_{ij} + HX_{ji} = 1 \quad i,j \in \Omega^\star;\ i \neq j;\ M(i) \neq M(j) \tag{2.7}$$

$$S_i + p_i + t_{\mathrm{suc}(i)} + e_{\mathrm{suc}(i),\mathrm{pre}(j)} + t_j \leq S_j + \alpha HX_{\mathrm{suc}(i)j} \quad i,j \in \Omega;\ i \neq j;\ M(i) = M(j) \tag{2.8}$$

$$HX_{\mathrm{suc}(i)j} + HX_{\mathrm{suc}(j)i} = 1 \quad i,j \in \Omega;\ i \neq j;\ M(i) = M(j) \tag{2.9}$$

$$HX_{ij} \in \mathbb{Z} \quad i,j \in \Omega^\star;\ i \neq j \tag{2.10}$$

$$e_{i,\mathrm{pre}(i)} + t_i \leq \alpha \quad i \in \Omega^\star \setminus \Omega \tag{2.11}$$

$$p_i + t_{\mathrm{suc}(i)} + e_{\mathrm{suc}(i),\mathrm{pre}(i)} + t_i \leq \alpha \quad i \in \Omega \tag{2.12}$$

$$S_{*j} + e_{*j,0} \leq S_0 + \alpha h \quad j \in \{1,\ldots,N\}. \tag{2.13}$$

Proof. First of all, we substitute $S_i(r)$ for all $i \in \Omega^\star$ according to constraint (1.1) in (1.4)–(1.10). Therefore, (2.4) and (2.13) are equivalent to (1.4) and (1.10).

Constraints (2.6)–(2.9) are of the same structure and we are only going to prove that (2.6), (2.7) and (2.10) are equivalent to constraint (1.1) and (1.8). Applying the substitution described above to constraint (1.8) we get

$$S_i + \alpha r_i + e_{i,\mathrm{pre}(j)} + t_j \leq S_j + \alpha r_j$$
$$\text{or } S_j + \alpha r_j + e_{j,\mathrm{pre}(i)} + t_i \leq S_i + \alpha r_i,$$

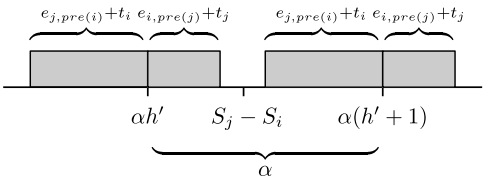
which is equivalent to

$$S_j - S_i \geq \alpha(r_i - r_j) + e_{i,\mathrm{pre}(j)} + t_j$$
$$\text{or } S_j - S_i \leq \alpha(r_i - r_j) - e_{j,\mathrm{pre}(i)} - t_i,$$

for $i,j \in \Omega^\star,\ i \neq j,\ M(i) \neq M(j)$ and $r_i, r_j \in \mathbb{Z}$. By setting $h' := r_i - r_j$ it follows that $S_j - S_i$ cannot be included in any of the intervals

$$]\alpha h' - e_{j,\mathrm{pre}(i)} - t_i,\ \alpha h' + e_{i,\mathrm{pre}(j)} + t_j[$$

for any $h' \in \mathbb{Z}$. The following graphic shows this relation.



Hence, there exists a $h^* \in \mathbb{Z}$ with

$$S_j - S_i \in [-\alpha h^* + e_{i,\mathrm{pre}(j)} + t_j, -\alpha(h^* - 1) - e_{j,\mathrm{pre}(i)} - t_i],$$

which means that

$$S_i + e_{i,\mathrm{pre}(j)} + t_j \leq S_j + \alpha h^*$$
$$\text{and } S_j + e_{j,\mathrm{pre}(i)} + t_i \leq S_i + \alpha(1 - h^*).$$

By setting $HX_{ij} = h^*$ and $HX_{ji} = 1 - h^*$ this is equivalent to (2.6), (2.7) and (2.10). Analogously, one can show that constraints (1.4) and (1.9) are equivalent to (2.4) and (2.8)–(2.10).

Finally, constraints (1.7) are, after substitution of $S_i(r)$ and $S_i(r+1)$, equivalent to (2.11) and (2.12), (2.4) are equivalent to (1.4)–(1.6). □

In this linear program the integer variables HX_{ij} restrict the job sequence on each machine and on the robot. In particular, $S_i(r)$ starts before $S_j(r)$ iff $HX_{ij} < HX_{ji}$. For more details about the interpretation of those values, we refer to Groenemeyer [8].

As a result of the linear program, we get the minimal cycle time α and feasible starting times $S_i(0)$ for every operation $\langle i, 0\rangle$ with $i \in \Omega^\star$. Note, that these values do not necessarily have to be included in the interval $[0, \alpha]$ since not all operations of a specific job instance have to start in the same cycle. However, constraint (2.13) ensures that all starting times are included in the interval $[0, \alpha h]$. Together with constraints (2.6)–(2.10) this also implies that $HX_{ij} \leq h$ for all $i, j \in \Omega^\star$. To get the starting times of each operation in the first considered cycle $[0, \alpha]$ we calculate the remainder of the division S_i/α for all $i \in \Omega^\star$ and thus, shift every operation in the first cycle. Note, that the repetition number of such an operation will change due to such a shift. An obvious question one can ask is, if an operation might clash with another operation on the same machine by shifting it in an earlier cycle and the same question could be asked for the robot moves. Or, in other words: Is the cycle length α large enough to process all operations once? (Note that neither (2.11) nor (2.12) are sufficient to bound the minimal cycle length.) Recalling the equivalence shown in Theorem 2.1 we know that a solution of the MIP model is also fulfilling constraints (1.1)–(1.10). The constraints guaranteeing that such a clash cannot happen are (1.1), (1.8) and (1.9). (The corresponding constraints in the MIP model are (2.6)–(2.10).) Each operation starts every α time units and therefore in every cycle at the same position. Constraints (1.9) define the order of any two operations processed on the same machine and ensure that they will not clash and (1.8) does the same for the robot moves.

After solving the linear program and calculating the starting time of each operation in a specific cycle, the repetition number for each starting time in the resulting schedule can be obtained as follows. For every job J_j, we start with its last operation $i = \star^j$ and assign it an arbitrary repetition number $r_i = r$. If the starting time of the predecessor $\mathrm{pre}(i)$ is smaller than the starting time of i ($S_{\mathrm{pre}(i)} \leq S_i$) then both operations must belong to the same instance of J_i and therefore get the same repetition number. If not, the actual predecessor of i must have started in the previous cycle. Hence, we set $r_{\mathrm{pre}(i)} = r_i + 1$. We continue until the first operation of J_j is assigned a repetition number. The procedure is then repeated for the remaining jobs.

By nature formulation (2.1)–(2.13) is not linear since constraints (2.6) and (2.8) are quadratic. However, it can be rewritten by dividing constraints (2.4), (2.6) and (2.8) by α. One can afterwards substitute $1/\alpha = \bar{\alpha}$, $S_i/\alpha = \bar{S}_i$ and $p_i/\alpha = \bar{p}_i$ and change the objective to maximise $\bar{\alpha}$.

3. A new mixed integer programming model

In this section, we will present a new formulation for the CJSPT that is more tailored to the problem compared to the model in the previous section. The major difference between cyclic and non-cyclic problems is that the precedence constraints between the operations are slightly relaxed. So, in a specific cycle operation i, does not have to be scheduled before its successor $\mathrm{suc}(i)$. In this case, the precedence constraints are, of course, not violated since both operations must belong to different repetitions of $J(i)$, but they provide a more flexible layout of the schedule. This property is due to the fact that operations are allowed to start in one cycle and finish in the next one. Those operations were called overlapping operations. Fig. 3 shows parts of two repetitions of a cycle in a cyclic schedule where i is an overlapping operation. To model overlapping operations, we introduce a set of binary variables γ_i with $i \in \Omega^\star$ which are defined as

$$\gamma_i = \begin{cases} 1, & \text{if } i \text{ is overlapping;} \\ 0, & \text{else.} \end{cases}$$

Remember that the number of overlapping operations of a specific job J_j is equal to $h_j - 1$, and this is limited by the overall height h. The number of overlapping operations per job has to be less than $h - 1$, which leads to

$$\sum_{i \in J_j} \gamma_i \leq h - 1, \tag{3.1}$$

for all $j = 1, \ldots, N$. The main idea behind this model is to specify the relations between operations in the same cycle rather than looking ahead into the next one. As before, we make the assumption that we start the cycle at machine M_0 with loading operation $i = 0$ at time 0. Furthermore, at the end of the cycle, the robot has to drive back to the input machine. Since we

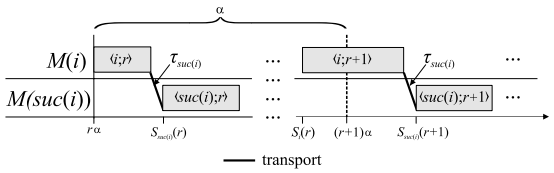


Fig. 3. Example of an overlapping operation i.

are only concentrating on all operations in one specific cycle, we can ignore the repetition numbers. Hence, the following constraints must hold:

$$S_0 = 0, \tag{3.2}$$

$$S_1 = t_1, \tag{3.3}$$

$$S_i + e_{i0} \leq \alpha, \tag{3.4}$$

for all $i \in \Omega^\star$. To ensure that the precedence constraints, (1.4) and (1.5) hold, we have to distinguish between overlapping and non-overlapping operations. For the non-overlapping case, it still holds that $S_i + p_i^{\min} + t_{\mathrm{suc}(i)} \leq S_{\mathrm{suc}(i)}$. For the overlapping case, we have $S_i + p_i^{\min} + t_{\mathrm{suc}(i)} \leq S_{\mathrm{suc}(i)} + \alpha$. Both cases can be combined in the following constraint:

$$S_i + p_i^{\min} + t_{\mathrm{suc}(i)} \leq S_{\mathrm{suc}(i)} + \alpha\gamma_i.$$

If i is not overlapping, then $\gamma_i = 0$ and $\alpha\gamma_i$ disappears. Since this is not a linear constraint we can split it up into the following two constraints:

$$S_i + p_i^{\min} + t_{\mathrm{suc}(i)} \leq S_{\mathrm{suc}(i)} + C\gamma_i, \tag{3.5}$$

$$S_i + p_i^{\min} + t_{\mathrm{suc}(i)} \leq S_{\mathrm{suc}(i)} + \alpha, \tag{3.6}$$

for all $i \in \Omega$ and where $C \in \mathbb{N}$ is a sufficiently large constant.

The constraints for the transportation of each job can be modelled in a very similar way to the ones in the previous model. Therefore, we introduce a set of binary variables θ_{ij} with $i, j \in \Omega^\star$ which are defined as

$$\theta_{ij} = \begin{cases} 1, & \text{if } i \text{ is transported after } j; \\ 0, & \text{else.} \end{cases}$$

The robot constraints (1.8) can then be formulated as

$$S_i + e_{i,\mathrm{pre}(j)} + t_j \leq S_j + C\theta_{ij}, \tag{3.7}$$

$$\theta_{ij} + \theta_{ji} = 1, \tag{3.8}$$

for all $i, j \in \Omega^\star$ and where $C \in \mathbb{N}$ again is a sufficiently large constant. Note that these constraints are logically identical to constraints (2.6) and (2.7). Another fact is that a transport move itself will never overlap. This is because we assume that the cycle starts with unloading a job from the input machine and that the cycle finishes with the robot arriving empty at the input machine.

Finally, we will formulate the blocking constraints for operations that have to go on the same machine. We again introduce a set of binary variables β_{ij} with $i, j \in \Omega^\star$ which define the processing order of the jobs on the same machine in the current cycle:

$$\beta_{ij} = \begin{cases} 0, & \text{if } j \text{ is processed after } i \text{ on } M(i) = M(j); \\ 1, & \text{else} \end{cases}$$

for all $i, j \in \Omega$ with $M(i) = M(j)$.

A first set of constraints is similar to constraints (2.6) and (2.7) and ensures that an operation cannot start its processing before the previous job on the same machine has been transported to its succeeding one:

$$S_{\mathrm{suc}(i)} + e_{\mathrm{suc}(i)\mathrm{pre}(j)} + t_j \leq S_j + C\beta_{ij}, \tag{3.9}$$

$$\beta_{ij} + \beta_{ji} = 1, \tag{3.10}$$

for all $i, j \in \Omega$ with $M(i) = M(j)$. Considering only one specific cycle, makes it more difficult to deal with overlapping operations. Before we start to tackle this problem, it is worth mentioning that there can be at most one overlapping operation on each machine and this operation always has to be loaded first off and last onto the machine in the cycle. Such an operation i has two processing periods in the cycle. One is at be very beginning, i.e. from time 0 to $S_{\mathrm{suc}(i)} - t_{\mathrm{suc}(i)}$. In addition, one is from S_i to α (cf. Fig. 3). Even if those two physically do not belong to the same repetition of the job, the sum of these processing times needs to be at least $p_i^{\min}$. Therefore in the following, we have to distinguish between an overlapping and a non-overlapping

operation on each machine. For the remaining constraints, consider i, j in Ω, with $i \neq j$ and $M(i) = M(j)$. In the case when i is overlapping, its successor has to start before any other operation in the cycle. This leads to the following constraints:

$$S_{\mathrm{suc}(i)} + e_{\mathrm{suc}(i)\mathrm{pre}(j)} + t_j \leq S_j + C(1 - \gamma_i). \tag{3.11}$$

Furthermore, all other operations must have finished their processing and been unloaded before the last operation on a machine can start in the current cycle:

$$S_{\mathrm{suc}(j)} \leq S_i + C(1 - \gamma_i), \tag{3.12}$$

where $M(i) \neq M(\mathrm{suc}(j)), j \neq i$ and $M(i) = M(j)$. Since there is at most one overlapping operation per machine, all other operations on this machine must have stayed (at least) for their minimal processing times, which leads to:

$$S_j + p_j^{\min} + t_{\mathrm{suc}(j)} \leq S_i + C(1 - \gamma_i), \tag{3.13}$$

where $M(i) \neq M(\mathrm{suc}(j)), j \neq i$ and $M(i) = M(j)$. Finally, constraints (1.6) and (1.7) have to hold as before, which gives the same constraints as in the model from the previous section:

$$e_{i,\mathrm{pre}(i)} + t_i \leq \alpha \quad \text{for } i \in \Omega^\star \setminus \Omega \quad \text{and} \quad e_{\mathrm{suc}(i),\mathrm{pre}(i)} + t_i \leq \alpha \quad \text{for } i \in \Omega. \tag{3.14}$$

Finally the model can be summarised by the following mixed integer program.

$$\min \alpha \tag{3.15}$$

s.t.

$$\sum_{i \in J_j} \gamma_i \leq h - 1 \quad j \in \{1, \ldots, N\} \tag{3.16}$$

$$S_0 = 0 \tag{3.17}$$

$$S_1 = t_1 \tag{3.18}$$

$$S_i + e_{i0} \leq \alpha \quad i \in \Omega^\star \tag{3.19}$$

$$S_i + p_i^{\min} + t_{\mathrm{suc}(i)} \leq S_{\mathrm{suc}(i)} + C\gamma_i \quad i \in \Omega \tag{3.20}$$

$$S_i + p_i^{\min} + t_{\mathrm{suc}(i)} \leq S_{\mathrm{suc}(i)} + \alpha \quad i \in \Omega \tag{3.21}$$

$$S_i + e_{i,\mathrm{pre}(j)} + t_j \leq S_j + C\theta_{ij} \quad i, j \in \Omega^\star \tag{3.22}$$

$$\theta_{ij} + \theta_{ji} = 1 \quad i, j \in \Omega^\star \tag{3.23}$$

$$S_{\mathrm{suc}(i)} + e_{\mathrm{suc}(i)\mathrm{pre}(j)} + t_j \leq S_j + C\beta_{ij} \quad i, j \in \Omega,\ i \neq j,\ M(i) = M(j) \tag{3.24}$$

$$\beta_{ij} + \beta_{ji} = 1 \quad i, j \in \Omega,\ i \neq j,\ M(i) = M(j) \tag{3.25}$$

$$S_{\mathrm{suc}(i)} + e_{\mathrm{suc}(i)\mathrm{pre}(j)} + t_j \leq S_j + C(1 - \gamma_i) \quad i, j \in \Omega,\ i \neq j,\ M(i) = M(j) \tag{3.26}$$

$$S_{\mathrm{suc}(j)} \leq S_i + C(1 - \gamma_i) \quad i, j \in \Omega,\ i \neq j,\ M(i) = M(j),\ M(i) \neq M(\mathrm{suc}(j)) \tag{3.27}$$

$$S_j + p_j^{\min} + t_{\mathrm{suc}(j)} \leq S_i + C(1 - \gamma_i) \quad i, j \in \Omega,\ i \neq j,\ M(i) = M(j) \tag{3.28}$$

$$e_{i,\mathrm{pre}(i)} + t_i \leq \alpha \quad i \in \Omega^\star \setminus \Omega \tag{3.29}$$

$$e_{\mathrm{suc}(i),\mathrm{pre}(i)} + t_i \leq \alpha \quad i \in \Omega, \tag{3.30}$$

$$\gamma_i, \theta_{ij}, \beta_{ij} \in \{0, 1\} \tag{3.31}$$

for all $i, j \in \Omega^\star$ and $C \in \mathbb{N}$ is a sufficiently large constant. Since modelling the overlapping operations on each machine is the key property of this model, we will refer to it as the *CJSPT–MIP–OO*. The repetition numbers for a resulting schedule can be obtained in the same way as in Section 2.

4. Computational results

In this section, we will compare the computational results for the models presented. Both models have been solved with CPLEX 12.2 on an Intel Xeon E5472 3.0 GHz computer with 16 GB memory, single threaded, running Linux 64 bit. We set the time limit for each problem instance to 3600 s. Since there are, to our knowledge, no standard benchmarks for the CJSPT available in the literature, we created our own data set generator (see Appendix for further details).

The computational results are given in Tables 4.1 and 4.2 which are structured as follows. The first column, 'instance', describes the problem instance in the format $N \times m - k$, where N is the number of jobs, m is the number of machines and k is just a counter for instances of the same size. We have tested the different models with three different instances of the same size and each instance with a maximum height of 1, 2 and 3. Every job has to visit every machine exactly once which means that the number of operations is $n = Nm$.

Table 4.1
Computational results 1.

Instance	Height	CJSPT–MIP–LIT					CJSPT–MIP–OO				
		Solution	LB	Gap	Time (s)	Memory (MB)	Solution	Bound	Gap	Time (s)	Memory (MB)
$5\times5-1$	1	519	519	0.0%	<1	–	519	519	0.0%	<1	–
	2	508	508	0.0%	<1	–	508	508	0.0%	<1	–
	3	508	508	0.0%	<1	–	508	508	0.0%	<1	–
$5\times5-2$	1	446	446	0.0%	<1	–	446	446	0.0%	<1	–
	2	432	432	0.0%	<1	–	432	432	0.0%	<1	–
	3	432	432	0.0%	<1	–	432	432	0.0%	<1	–
$5\times5-3$	1	519	519	0.0%	<1	–	519	519	0.0%	<1	–
	2	482	482	0.0%	<1	–	482	482	0.0%	<1	–
	3	482	482	0.0%	<1	–	482	482	0.0%	<1	–
$6\times6-1$	1	616	616	0.0%	1	–	616	616	0.0%	1	–
	2	575	575	0.0%	23	–	575	575	0.0%	43	–
	3	575	575	0.0%	201	–	575	575	0.0%	10	–
$6\times6-2$	1	605	605	0.0%	2	–	605	605	0.0%	<1	–
	2	548	548	0.0%	8	–	548	548	0.0%	14	–
	3	548	548	0.0%	50	–	548	548	0.0%	8	–
$6\times6-3$	1	559	559	0.0%	1	–	559	559	0.0%	<1	–
	2	532	532	0.0%	58	–	532	532	0.0%	14	–
	3	532	532	0.0%	531	–	532	532	0.0%	63	–
$7\times7-1$	1	683	683	0.0%	89	–	683	683	0.0%	7.3	–
	2	623	532	16.1%	3600	1242	**620**	620	0.0%	788	–
	3	623	412	33.9%	3600	414	**620**	620	0.0%	2621	–
$7\times7-2$	1	735	735	0.0%	81	–	735	735	0.0%	12	–
	2	644	612	5.0%	3600	339	**634**	634	0.0%	394	–
	3	634	457	27.9%	3600	3021	**634**	634	0.0%	930	–
$7\times7-3$	1	676	676	0.0%	129	–	676	676	0.0%	6	–
	2	621	539	13.2%	3600	1350	**618**	618	0.0%	1882	–
	3	621	384	38.2%	3600	3912	**618**	618	0.0%	1190	–
$8\times8-1$	1	948	948	0.0%	2623	–	948	948	0.0%	92	–
	2	874	571	34.7%	3600	3506	611	431	29.5%	3600	328
	3	893	516	42.2%	3600	3733	530	320	35.8%	3600	488
$8\times8-2$	1	887	792	10.7%	3600	404	**884**	884	0.0%	135	4
	2	811	634	21.8%	3600	1785	649	524	19.3%	3600	327
	3	879	634	27.9%	3600	3108	634	491	22.6%	3600	502
$8\times8-3$	1	852	802	5.9%	3600	1657	**852**	852	0.0%	92	–
	2	769	424	44.9%	3600	3196	797	488	38.8%	3600	456
	3	882	388	56.0%	3600	4648	972	405	58.3%	3600	489
$9\times9-1$	1	1175	722	38.6%	3600	3839	1050	1033	1.6%	3600	66
	2	inf	498	inf	3600	4387	inf	656	inf	3600	418
	3	inf	432	inf	3600	3428	inf	588	inf	3600	494
$9\times9-2$	1	1188	728	38.7%	3600	3867	871	714	18.0%	3600	572
	2	inf	501	inf	3600	4463	751	480	36.1%	3600	453
	3	1137	452	60.2%	3600	3851	inf	387	inf	3600	387
$9\times9-3$	1	1119	764	31.7%	3600	3808	**1097**	1097	0.0%	925	–
	2	1272	665	47.6%	3600	3582	inf	inf	inf	3600	408
	3	inf	665	inf	3600	4639	inf	inf	inf	3600	410

The first set of columns contains the results for the model from the literature (CJSPT–MIP–LIT) and the second set (CJSPT–MIP–OO) those from the newly developed model presented in this paper.

The first column contains the maximum given height. The minimal cycle time obtained by CPLEX ('Solution') is given in the next column, followed by the lower bound ('LB') and the corresponding gap ('GAP') which has been calculated by $\frac{\text{solution}-\text{LB}}{\text{solution}}$. The last two columns contain the time needed to solve the problem and the space for storing the tree if the problem could not be solved to optimality when the time limit had been reached. If no value for a specific column could be obtained (e.g. no solution has been found) we write 'inf'.

As we can see, our new model mostly outperforms the model from the literature. It always finds the optimum when the CJSPT–MIP–LIT model finds it and this is even the case for 20 more instances. Furthermore the differences in the memory needed for the branching tree are enormous. The average memory needed for the CJSPT–MIP–LIT model is 2931 MB, not taking into account instances solved to optimality, whereas the CJSPT–MIP–OO model only uses an average of 400 MB per unsolved instance. This new formulation especially benefits scenarios where there are computers with less memory or 32 bit operating systems.

Table 4.2
Computational results 2.

Instance	Height	CJSPT–MIP–LIT					CJSPT–MIP–OO				
		Solution	LB	Gap	Time (s)	Memory (MB)	Solution	Bound	Gap	Time (s)	Memory (MB)
$10\times10-1$	1	1403	803	42.8%	3600	5749	1389	803	42.2%	3600	93
	2	inf	446	inf	3600	2910	inf	346	inf	3600	270
	3	inf	228	inf	3600	3294	inf	311	inf	3600	277
$10\times10-2$	1	inf	822	inf	3600	5367	1444	800	44.6%	3600	54
	2	inf	362	inf	3600	967	inf	383	inf	3600	297
	3	inf	276	inf	3600	2611	inf	351	inf	3600	206
$10\times10-3$	1	inf	720	inf	3600	6572	1441	777	46.1%	3600	521
	2	inf	310	inf	3600	3863	inf	318	inf	3600	391
	3	inf	289	inf	3600	2821	inf	293	inf	3600	300
$10\times5-1$	1	819	819	0.0%	728	–	819	819	0.0%	28	–
	2	779	649	16.7%	3600	373	**779**	779	0.0%	704	–
	3	783	501	36.0%	3600	724	**779**	779	0.0%	1310	–
$10\times5-2$	1	818	818	0.0%	515	–	818	818	0.0%	147	–
	2	810	551	32.0%	3600	1350	**808**	808	0.0%	1150	–
	3	816	495	39.3%	3600	665	**808**	808	0.0%	3124	–
$10\times5-3$	1	844	844	0.0%	1798	–	844	844	0.0%	245	–
	2	798	572	28.3%	3600	1597	**798**	798	0.0%	1236	–
	3	822	463	43.7%	3600	625	**798**	798	0.0%	2661	–
$5\times10-1$	1	694	694	0.0%	<1	–	694	694	0.0%	7	–
	2	560	512	8.6%	3600	–	**558**	558	0.0%	1610	–
	3	617	417	32.4%	3600	–	**548**	548	0.0%	3455	–
$5\times10-2$	1	765	764	0.1%	3600	1	**765**	765	0.0%	6	–
	2	545	545	0.0%	<1	–	558	461	17.4%	3600	178
	3	545	509	6.6%	3600	1098	554	545	1.6%	3600	4
$5\times10-3$	1	696	696	0.0%	<1	–	696	696	0.0%	4	–
	2	524	504	3.8%	3600	665	**524**	524	0.0%	2818	–
	3	530	417	21.3%	3600	2848	**516**	516	0.0%	2276	–
$15\times15-1$	1	inf	953	inf	3600	2979	inf	927	inf	3600	85
	2	inf	722	inf	3600	3122	inf	289	inf	3600	201
	3	inf	621	inf	3600	4097	inf	316	inf	3600	289
$15\times15-2$	1	inf	942	inf	3600	3099	inf	959	inf	3600	89
	2	inf	483	inf	3600	4322	inf	330	inf	3600	209
	3	inf	380	inf	3600	76	inf	301	inf	3600	197
$15\times15-3$	1	inf	830	inf	3600	2255	inf	860	inf	3600	100
	2	inf	401	inf	3600	5204	inf	295	inf	3600	209
	3	inf	352	inf	3600	78	inf	297	inf	3600	193

5. Conclusions and future work

In this paper, we described the cyclic job-shop problem with one transport robot and blocking. We extended an existing model from the literature and showed how a mixed integer programming formulation can be applied to solve the problem. The main contribution of this paper is in Section 3, where we present a new and more tailored integer programming formulation for solving the CJSPT. The computational results show, that not only more problem instances could be solved or their solutions improved, but also, that the usage of the memory has been decreased enormously.

In future work, we will try to adjust the formulation, to solve different models from the literature, such as those mentioned in Section 2. We also aim to apply it to cyclic job-shop problems without transportation and/or no blocking.

Appendix. Generating problem instances

For the cyclic job-shop problem with one transport robot, there are, as far as we know no standard benchmarks available. Most authors have considered standard job-shop benchmarks and have added additional times for transportation. To be able to test any algorithm on different classes of problem instances (especially different ratios between processing times and transport times), we developed a problem generator for those instances [2].

The underlying pseudo random number generator (PRNG) is based on Park and Miller [16]. Thus, every problem instance can be reproduced if the program is run with the same parameters. The input parameters are the number of jobs N, the number of machines m (including an input and output machine) and a seed number for the PRNG. The generator randomly assigns a permutation of all machines to the jobs. Furthermore, one can set minimal and maximal values for the precessing, setup, empty moving and additional transport time. A reasonable assumption is that the transport time cannot be smaller

than the corresponding empty moving time between the same machines. Therefore, we add a random value between a minimal and maximal additional transport time to the corresponding empty moving time.

The processing times are calculated as follows. There is a lower bound for the smallest minimal processing time and an upper bound for the biggest maximal processing time. Additionally, a minimal distance between minimal and maximal processing time can be set (processing time window).

Since the triangle inequality has to hold, we determine the distances between the machines in the following way. We create a 2-dimensional quadratic area with diameter equal to the difference between minimum and maximum empty moving time. Then, we randomly place all machines on this area, calculate their euclidian distances to each other and add the minimal empty moving time. This guarantees a distance between the machines according to the given limits. We assume that every job starts at input machine 0 and finishes at the output machine $m - 1$. The minimal processing and setup times of the output machine are set to 0 and the maximum processing time to a big enough number.

Here is an example of a small problem instance with 2 jobs and 5 machines.

```
*SEED 212121
*MIN_PROCESSING_TIME 10
*MAX_PROCESSING_TIME 99
*MIN_PROCESSING_TIME_WINDOW 20
*MIN_TRANSPORT_TIME 1
*MAX_TRANSPORT_TIME 4
*MIN_EMPTY_MOVE_TIME 1
*MAX_EMPTY_MOVE_TIME 8
*MIN_SETUP_TIME 4
*MAX_SETUP_TIME 8
*The first line represents the numbers of jobs (2) followed by the numbers of machines (5).
*Each of the next 2 line(s) represent all operations of one job.
*Each operation has assigned 5 values which are in the following order:
*Machine | minProcessingTime | maxProcessingTime | transportTime | setupTime
*The other lines representing the time distance between the machines are of the form:
*machine-A | machine-B | distance
*We assume that the distance between the machines is symmetric
2 5                                                  // #jobs #machines
3 42 74 6 8 2 75 97 9 6 1 73 95 8 7 4 0 9999 5 0 // operations job 1
2 27 56 5 7 3 25 68 7 8 1 48 94 8 5 4 0 9999 5 0 // operations job 2
0 0 0                                                // empty moving times
0 1 5
0 2 4
0 3 5
0 4 5
1 1 0
1 2 6
1 3 5
1 4 7
2 2 0
2 3 3
2 4 2
3 3 0
3 4 4
4 4 0
```

References

[1] F.L. Baccelli, G. Cohen, G.J. Olsder, J.-P. Quadrat, Synchronization and Linearity: An Algebra for Discrete Event Systems, John Wiley & Sons, 1992.
[2] P. Brucker, E.K. Burke, S. Groenemeyer, Problem generator for job-shop problem with transportation, 2009. URL: http://www.cs.nott.ac.uk/~svg/problemgenerator.html.
[3] P. Brucker, T. Kampmeyer, Tabu search algorithms for cyclic machine scheduling problems, Journal of Scheduling 8 (2005) 303–322.
[4] P. Brucker, T. Kampmeyer, A general model for cyclic machine scheduling problems, Discrete Applied Mathematics 156 (2008) 2561–2572.
[5] P. Brucker, T. Kampmeyer, Cyclic job shop scheduling problems with blocking, Annals of Operations Research 159 (2008) 161–181.
[6] P. Chrétienne, Transient and limiting behavior of timed event graphs, RAIRO-TSI 4, 1985, pp. 127–142.
[7] G. Cohen, P. Moller, J.-P. Quadrat, M. Viot, Algebraic tools for the performance evaluation of discrete event systems, Proceedings of the IEEE 77 (1) (1989) 39–85.
[8] S. Groenemeyer, Novel approaches to cyclic job-shop problems with transportation, Ph.D. Thesis, University of Nottingham, School of Computer Science, 2012.
[9] N. Hall, C. Sriskandarajah, A survey of machine scheduling problems with blocking and no-wait in process, Operations Research 45 (1997) 510–525.
[10] C. Hanen, Study of a NP-hard cyclic scheduling problem: the recurrent job-shop, European Journal of Operational Research 72 (1) (1994) 82–101.
[11] C. Hanen, A. Munier, Cyclic scheduling on parallel processors: an overview, in: Scheduling Theory and its Applications, John Wiley & Sons, 1995, (Chapter).
[12] K. Hitz, Scheduling of flexible flow shops II, Tech. Rep., Laboratory for Information and Decision Systems, MIT, Cambridge, USA, 1980.

[13] H. Kamoun, C. Sriskandarajah, The complexity of scheduling jobs in repetitive manufacturing systems, European Journal of Operational Research 70 (1993) 350–364.
[14] V. Kats, E. Levner, A parametric critical path problem and an application for cyclic scheduling, Discrete Applied Mathematics 87 (1998) 149–158.
[15] S. McCormick, M.L. Pinedo, S. Shenker, B. Wolf, Sequencing in an assembly line with blocking to minimize cycle time, Operations Research 37 (6) (1989) 925–935.
[16] S.K. Park, K.W. Miller, Random number generators: Good ones are hard to find, Communications of the ACM 31 (10) (1988) 1192–1201.
[17] Y. Robert, F. Vivien, Introduction to Scheduling, Chapman and Hall, CRC Press, 2009.
[18] R. Roundy, Cyclic schedules for job-shops with identical jobs, Mathematics of Operations Research 17 (1992) 842–865.
[19] B. Trouillet, O. Korbaa, J.-C. Gentina, Formal approach of FMS cyclic scheduling, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37 (1) (2007) 126–137.