Chapter 6
Using Graphs for Modeling and Solving Cyclic Flow Shop with Waiting Time Constraints

Czeslaw Smutnicki and Wojciech Bożejko

Abstract The chapter deals with the permutation flow-shop scheduling problem with time couplings in the form of an upper/lower bound on waiting-time between operations in the job and the minimal cycle time criterion. We propose certain decomposition of the problem leading to two particular sub-problems, namely "to find the minimal cycle time for a fixed job processing order" and "to find the best job processing order". A variety of graphs have been used to solve the former sub-case in the sequential and parallel computing environments. For the latter sub-case we recommend a meta-heuristic with some special properties derived from these graphs.

Keywords Scheduling · Cyclic flow-shop · Waiting times · Graphs · Algorithm · Time couplings

6.1 Introduction

A deterministic scheduling theory is frequently used for modeling and optimization of workflow in production systems with robots, see for example review in Dolgui et al. [9]. These models perform well for human-less systems, since robots usually works in almost deterministic way. The existing taxonomy of problems in the scheduling field is set to cover, as this is possible, practical instances and takes into account, among others, architecture of production/service units, classes of the schedules, additional constraints and optimization criteria. The goal of this classification is to fit for each practical case to the appropriate model and to recommend a powerful solution method, see for example Bocewicz et al. [1, 2]. Among the schedule

C. Smutnicki (✉)
Faculty of Electronics, Department of Computer Engineering, Wroclaw University of Science and Technology, Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
e-mail: czeslaw.smutnicki@pwr.edu.pl

W. Bożejko
Faculty of Electronics, Department of Control Systems and Mechatronics, Wroclaw University of Science and Technology, Wroclaw, Poland

S. Zawiślak and J. Rysiński (eds.), *Graph-Based Modelling in Science, Technology and Art*, Mechanisms and Machine Science 107,
https://doi.org/10.1007/978-3-030-76787-7_6

classes appeared in the scheduling theory, we use only two of them in this chapter: (1) non-delay schedule, which means the single execution of the job set, see e.g. definition in Pinedo [21] and (2) cyclic schedule, which means repeatable execution of the job set, see e.g. Brucker and Kampmeyer [5, 6]. The former one is treated as auxiliary for the latter. The permutation flow-shop scheduling problem appears in the literature quite frequently, see for example the survey in Ruiz and Maroto [23]. A few real implementations were presented till now in the literature, e.g. in the chemical industry, Rajendran [22], food industry, Hall and Sriskandarajah [15] and construction automation, Grabowski and Pempera [12].

We consider the cyclic flow-shop scheduling problem in which an assortment of products (a certain fixed mixture of products) is delivered to the output by repeating identical, but mixed in the composition production series, each of which called a Minimal Part Set (MPS). The problem has also included "limited waiting time" constraints between successive operations of a job, which is a generalization of the case known in the literature as "no-wait flow-shop". Both mentioned technologies are used to eliminate or reduce the queue before a stage. Gilmore and Gomory [11] analysed a non-delay schedule for the flow-shop problem with "no-wait" constraints for two machines. This result can be easily extended to cyclic schedule in case of two machines. For more than two machines, problem has a relation to the Traveling Salesman Problem (TSP), thus algorithms for asymmetric TSP can be recommended. Other variants of the flow-shop with "no-wait" constraints have been considered in Grabowski and Pempera [14]. We also refer to algorithms dedicated to flow-shop [16] and hybrid flow-shop [8, 24, 28].

The remainder of the chapter is as follows. Section 6.2 gives a mathematical model with reference to a linear programming task to find the minimum cycle time for a given job processing order. Alternative solutions algorithms with the use of graphs are presented in the Sect. 6.3. Section 6.4 describes methods of parallel determination of the minimum cycle time (Table 6.1).

6.2 Mathematical Model

The conventional flow-shop scheduling problem refers to the machine park consisting of m service stages defined by the set $M = \{1, \ldots, m\}$, which are used to perform n jobs from the set $N = \{1, 2, \ldots, n\}$. Each job flows through machines $1, 2, \ldots, m$ in the identical technological order, however the order of loading the jobs into the manufacturing system is changeable. In the paper we consider two cases: (a) n jobs are loaded to the system once, in some order, providing so called non-delay schedule considered next in the context of regular scheduling criteria; (b) n jobs are performed in the repetitive way, duplicating the loading processing order repeatedly, providing so called cyclic schedule, which is actually an irregular scheduling criteria. Processing time of a job $j \in N$ on machine $i \in M$ is called an operation (i, j) and has the duration $p_{i,j} > 0$. Time event constraint means that the time of waiting between processing of successive operation of a job has pre-defined lower $(a_{i,j})$ and upper $(b_{i,j})$

Table 6.1 List of symbols

data	meaning
$M = \{1, 2, \ldots, m\}$	set of machines
$N = \{1, 2, \ldots, n\}$	set of jobs
$O = M \times N$	set of operations
p_{ij}	processing time of operation (i, j)
a_{ij}	lower bound on waiting time $(i, j) \rightarrow (i + 1, j)$
b_{ij}	upper bound on waiting time $(i, j) \rightarrow (i + 1, j)$
variables	**meaning**
$\pi = (\pi(1), .., \pi(n))$	processing order; permutation on N
$S = [S_{ij}]_{m \times n}$	schedule (matrix)
T	cycle time
criteria	**meaning**
$C_{\max}(\pi)$	makespan for processing order π
$T(\pi)$	minimal cycle time for processing order π
auxiliary	**meaning**
$x = 1, 2, \ldots$	index of cycles (MPSes)
i, j	machine index; job index
$(i, j) \in O$	operation i of job j
$o = m \cdot n$	number of operations
$G(\pi)$	planar graph for π
$H(\pi)$	wrapped-around-cylinder graph for π
$G^*(\pi)$	chain of $m + 1$ graphs $G(\pi)$
i^x	operation (pair) in cycle x in $G^*(\pi)$
$U_{i^x, j^y} = (u_0, .., u_v)$	path $i^x \longrightarrow j^y$
$L(i^x, j^y)$	path length
$B_{a:b}$	block
$B_{a+1:b-1}$	internal block

bounds. Processing jobs on the machines cannot be suspended. Each machine can process at most one job at a time, and each job can be processed on only one machine at a time.

Jobs enter the system in the order established by a permutation $\pi = (\pi(1), \ldots, \pi(n))$ on the set N, which will be called next the "loading sequence" on the system entry. Each π implies the schedule $S = [S_{i,j}]_{m \times n}$ (defined as starting times of all operations). Considering case (a) we refer to the most popular criterion called the makespan written as

$$C_{\max}(\pi) = \max_{1\le i\le m}\max_{1\le j\le n}(S_{i,j} + p_{i,j}). \tag{6.1}$$

Note that values of other regular criteria can be calculated from a non-delay schedule S. For the case (b) we refer to the minimal cycle time $T(\pi)$, which is calculated in more complex manner, described in the sequel. The overall aim is to find permutation π^* with minimal criterion value

$$K(\pi^*) = \min_{\pi\in\Pi} K(\pi), \tag{6.2}$$

where Π is the set of permutations and $K(\pi)$ is either $C_{\max}(\pi)$ or $T(\pi)$. Let us consider the problem of finding S in cases (a) and (b) for the given π in detail.

The manufacturing system repeats processing of jobs from N in the form of successive MPSes indexed by $x = 1, 2, \ldots$. Case (a) corresponds to $x = 1$, case (b) to infinite sequence $x = 1, 2, \ldots$. We assume in case (b) that the job processing order π in each cycle remains the same. The schedule in successive cycles is periodical so can be obtained by shifting on the time axis the schedule S by a certain number of periods T. For the given π, the minimal period of time, for which technological and sequential constraints inside the cycle and between cycles have been satisfied is $T(\pi)$. We are looking in case (b) for the synchronous periodical schedule with the property

$$S^x_{i,j} = S_{i,j} + (x-1)T, \quad i = 1,\ldots,m,\ j = 1,\ldots,n,\ x = 1,\ldots. \tag{6.3}$$

where $S^x_{i,j}$ is the start time of job $j \in N$ on machine i in xth cycle and $T \ge T(\pi)$. From (6.3) it is enough to set single start time S, since all the remain S^x for $x = 1, 2, 3, \ldots$ follow immediately from this S. Hence, we define cyclic schedule S as this satisfying the following constraints

$$S_{i,j} \ge 0, \quad i = 1,\ldots,m,\ j = 1,\ldots,n, \tag{6.4}$$

$$S_{i,\pi(j)} \ge S_{i,\pi(j-1)} + p_{i,\pi(j-1)}, \quad i = 1,\ldots,m,\ j = 2,\ldots,n, \tag{6.5}$$

$$a_{i,j} \le S_{i+1,j} - (S_{i,j} + p_{i,j}) \le b_{i,j}, \quad i = 2,\ldots,m,\ j = 1,\ldots,n, \tag{6.6}$$

$$S_{i,\pi(n)} + p_{i,\pi(n)} \le S_{i,\pi(1)} + T \quad i = 1,\ldots,m \tag{6.7}$$

Constraint (6.4) forces the schedule to be on the positive part of the time axis. Constraint (6.5) means that starting of successive operation on a machine is possible upon completion of the preceding operation on this machine. The last constraint fixes bounds on the waiting time expected between successive technological operations. Notice, assuming $a_{i,j} = 0 = b_{i,j}$ we get the well-known flow-shop scheduling problem with "no-wait" constraints. Value $a_{i,j}$ can be also perceived as the transport time between successive operations of a job. Minimal cycle time $T(\pi)$ for fixed job processing order π can be found using a few methods. We discuss them in detail. The first approach follows immediately from the linear programming (LP) task, namely

$$T(\pi) = \min_{S} T \tag{6.8}$$

under constraints (6.4) – (6.7). The task has $mn + 1$ non-negative variables and $2mn - n$ constraints. Although this task is a polynomial-time method, it seems to be numerically too expensive, since usually it is called repeatedly. Among other approaches, which can be adopted to find the cyclic schedule, we have found: mixed integer linear programming, [6]; network flows, [18]; cycles in graphs, [17]. Each of these technologies provides a non-polynomial algorithm to find $T(\pi)$, has various running time and requires specific modelling fashion. In this chapter we provide a few polynomial-time methods to find $T(\pi)$, based on specific graphs defined for the problem, see [20] for other implementations of this idea.

6.3 Graph Models

In this section we will introduce three various graphs (networks of operations) applied further to the formulation of special properties of the problem. These graphs refer to a few common notions defined in the sequel. Each graph is defined as the operational network presented in the AoN (Activity-on-Node) style. Node $(i, j) \in O = M \times N$ in the graph corresponds to the operation of job $j \in N$ performed on machine $i \in M$. The node (i, j) has weight $p_{i,j}$ and has assigned label $S_{i,j}$ which means the event "start time of the operation (i, j)". Any constraint between time events $S_{i,j}$ and $S_{k,l}$, for some (i, j) and (k, l), written generally as

$$S_{i,j} + p_{i,j} + c_{i,j,k,l} \leq S_{k,l} \tag{6.9}$$

is represented by the arc $(i, j) \to (k, l)$ with the weight $c_{i,j,k,l}$.

In order to build suitable graphs, we need to assign inequalities (6.4)–(6.7) to particular arcs in the network. Actually, inequality (6.5) re-written in the form of

$$S_{i,\pi(j-1)} + p_{i,\pi(j-1)} \leq S_{i,\pi(j)}, \quad i = 1, \ldots, m, \; j = 2, \ldots, n \tag{6.10}$$

can be expressed by the arc

$$(i, \pi(j-1)) \to (i, \pi(j)) \tag{6.11}$$

with the weight $c_{i,\pi(j-1),i,\pi(j)} = 0$. Equation (6.6) can be transformed into two independent inequalities (6.12) and (6.13), namely

$$S_{i+1,j} - S_{i,j} - p_{i,j} \geq a_{i,j}, \quad i = 2, \ldots, m, \; j = 1, \ldots, n, \tag{6.12}$$

$$S_{i+1,j} - S_{i,j} - p_{i,j} \leq b_{i,j}, \quad i = 2, \dots, m, \ j = 1, \dots, n, \tag{6.13}$$

To identify arcs following from (6.12) and (6.13) we have to convert them to the form of (6.9). After a few simple mathematical transformations of (6.12) we have

$$S_{i,j} + p_{i,j} + a_{i,j} \leq S_{i+1,j}, \quad i = 2, \dots, m, \ j = 1, \dots, n, \tag{6.14}$$

which corresponds to the arc

$$(i, j) \rightarrow (i + 1, j) \tag{6.15}$$

with the weight $c_{i,j,i+1,j} = a_{i,j}$. Next, from (6.13) we obtain

$$S_{i+1,j} + p_{i+1,j} + (-p_{i,j} - p_{i+1,j} - b_{i,j}) \leq S_{i,j}, \quad i = 2, \dots, m, \quad j = 1, \dots, n, \tag{6.16}$$

which corresponds to the arc

$$(i + 1, j) \rightarrow (i, j) \tag{6.17}$$

with the weight $c_{i+1,j,i,j} = -p_{i,j} - p_{i+1,j} - b_{i,j}$. Inequality (6.7) converted to

$$S_{i,\pi(n)} + p_{i,\pi(n)} - T \leq S_{i,\pi(1)}, \quad i = 1, \dots, m. \tag{6.18}$$

generates the arc

$$(i, \pi(n)) \rightarrow (i, \pi(1)) \tag{6.19}$$

with the weight $c_{i,\pi(n),i,\pi(1)} = -T$.

6.3.1 Non-delay Schedule

This planar graph is recommended for processing single job set N, see Fig. 6.5 (left) for the given job processing order π. It allows us to find a non-delay schedule S satisfying constraints (6.4)–(6.6), but not (6.7). It is suitable for finding the makespan $C_{\max}(\pi)$ or other regular criterion value for the given processing order π. The graph has the form

$$G(\pi) = (O, E(\pi)), \tag{6.20}$$

where $O = M \times N$ is the set of nodes and $E(\pi) \subset O \times O$ is the set of arcs

$$E(\pi) = P \cup R \cup Q(\pi). \tag{6.21}$$

Arcs from P refer to inequality (6.14), have the form of (6.15), represent "technological" constraints associated with lower bounds on "waiting-time"

$$P = \bigcup_{i=1}^{m-1} \bigcup_{j=1}^{n} \{(i, j), (i+1.j)\}. \tag{6.22}$$

The arc $((i, j), (i+1, j)) \in P$ has weight $a_{i,j}$. Arcs from R refer to inequality (6.16), have the form of (6.17), can be also considered as “technological” and express upper bounds on ”waiting-time”

$$R = \bigcup_{i=1}^{m-1} \bigcup_{j=1}^{n} \{(i+1, j), (i.j)\}. \tag{6.23}$$

The arc $((i+1, j), (i, j)) \in R$ has weight $-p_{i,j} - p_{i+1,j} - b_{i,j}$. Arcs from $Q(\pi)$ refer to inequality (6.10), have the form of (6.11), express “processing order” constraints and have the weight zero,

$$Q(\pi) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n-1} \{(i, \pi(j-1)), (i, \pi(j))\}. \tag{6.24}$$

We call schedule S feasible for this π if labels $S_{i,j}$ assigned to nodes $(i, j) \in O$ satisfy appropriate constraints for all $((i, j), (k, l)) \in E(\pi)$. Note that although $G(\pi)$ has graph cycles however it does not contain cycles of the length greater than zero. This means that the feasible S always exists. To find S we use the labeling algorithm, called next LA, applied previously in the paper [25] for a directed graph with graph cycles, see Fig. 6.1. LA checks for each arc $((i, j), (k, l)) \in E(\pi)$ whether the inequality $S_{k.l} \geq S_{i,j} + p_{i,j} + c_{i,j,k,l}$ is satisfied.

If the answer is “not satisfied”, LA adjusts $S_{k,l}$ to fulfil the constraint. Variable *tag* is used to detect the early end of the adjustment process. LA can be perceived as

```
Parameter: $\pi$
   for $(i,j) \in O$ do $S_{i,j} = 0$;
   for $v = 1$ to $m \cdot n - 1$ do
      $tag = 0$;
      for $((i,j),(k,l)) \in E(\pi)$ do
         if $S_{k,l} < S_{i,j} + p_{i,j} + c_{i,j,k,l}$ then
            { $S_{k,l} = S_{i,j} + p_{i,j} + c_{i,j,k,l}$; $tag = 1$; }
         end if;
      end for $((i,j),(k,l))$;
      if not $tag$ then return;
   end for $v$;
```

Fig. 6.1 Algorithm LA

```
Parameter:  $\pi$
   $k = \pi(1); S_{1,k} = 0$;
   for $i = 1, \ldots, m-1$ do $S_{i+1,k} = S_{i,k} + p_{i,k} + a_{i,k}$;
   for $j = 2 \ldots, n$ do
      $k = \pi(j)$; $l = \pi(j-1)$; $S_{1,k} = S_{1,l} + p_{1,l}$;
      for $i = 2, \ldots, m$ do
         $S_{i,k} = \max\{S_{i,l} + p_{i,l}, S_{i-1,k} + p_{i-1,k} + a_{i-1,k}\}$
      end for $i$;
      for $i = m, \ldots, 2$ do
         $S_{i-1,k} = \max\{S_{i-1,k}, S_{i,k} - p_{i-1,k} - b_{i-1,k}\}$
      end for $i$;
   end for $j$;
```

Fig. 6.2 Algorithm LA$^+$

a far analogy to the well-known Bellman-Ford algorithm dedicated for the shortest paths from the single source.

Property 1 *For the given π algorithm LA provides non-delay schedule S in the time $\Omega(m \cdot n)$ and $O(m^2 \cdot n^2)$.*

Apart LA, we propose more advanced labelling algorithm LA$^+$ with computational complexity considerably smaller than LA. Observe that $G(\pi)$ can be decomposed into a sequence of "strips" unscrambled along job processing order π, see Fig. 6.5 (left). Strip $\pi(i)$ has sequential predecessors in strips $\pi(1), \pi(2), \ldots, \pi(i-1)$ and technological predecessors only inside strip $\pi(i)$. This allows us to calculate S strip-by-strip, down-and-up, according to the constraints (6.10), (6.14), (6.16), see the pseudo-code of LA$^+$ in Fig. 6.2. Indeed, the first operation of the job $k = \pi(j)$, has no technological predecessors, so the substitution $S_{1,k} = S_{1,l} + p_{1,l}$ is clear. Next, formula $S_{i,k} = \max\{S_{i,l} + p_{i,l}, S_{i-1,k} + p_{i-1,k} + a_{i-1,k}\}$ in the former loop for $i = 2, \ldots, m$ ensures (6.10), (6.14) but not necessarily (6.16). Checking (6.16) and adjusting $S_{i-1,k}$ is performed in the later loop for $i = m, \ldots, 2$. It can be verified that LA$^+$ ensures the feasibility of (6.10), (6.16), (6.14).

Property 2 *For the given π algorithm LA$^+$ provides non-delay schedule S in the time $\Theta(m \cdot n)$.*

Besides the theoretical analysis of the computational complexity, one expects also experimental comparisons between LA and LA$^+$. Using 80 flow-shop instances with $m \cdot n \leq 1,000$ from the benchmark set of [27] we find that the ratio of the running time of LA to the running time of LA$^+$ is on average $0.6 \ldots 0.7 \cdot m \cdot n$. It means, among others, that already for medium size instances $n = 50$, $m = 10$, LA$^+$ is approximately 300 times faster than LA. It also means that the complexity of LA is closer to $O(m^2 \cdot n^2)$ than to $\Omega(m \cdot n)$.

Properties 1 and 2 allow us to find efficiently the goal function value for π for the majority of regular scheduling criteria, for example the makespan, mean flow time. Prospective benefits for the minimal cycle time criterion derive from the following property, the proof is obvious.

Property 3 *$T(\pi) \leq C_{\max}(\pi)$, where $C_{\max}(\pi)$ is the makespan found for $G(\pi)$.*

6.3.2 Cyclic Schedule

We introduce graph $H(\pi)$ to find the cyclic schedule S (for the given π and T) satisfying constraints (6.4)–(6.7) converted to the equivalent form (6.10), (6.14), (6.16), (6.18), if only such schedule exists. It vicariously provides an alternative method of finding $T(\pi)$, presented at the end of this subsection. The graph has the form of

$$H(\pi) = (O, E(\pi) \cup F(\pi)), \tag{6.25}$$

where O and $E(\pi)$ have been already defined for $G(\pi)$ (graph $G(\pi)$ fulfills constraints (6.10), (6.14), (6.16)), whereas set of arcs

$$F(\pi) = \bigcup_{i=1}^{m} \{((i, \pi(n)), (i, \pi(1))), \tag{6.26}$$

represents constraint (6.18). Each arc has the form (6.19) and the negative weight $-T$. $H(\pi)$ is a non-planar graph, can be drawn as an extension of $G(\pi)$ wrapped around a cylinder, see Fig. 6.5 (right), and can be perceived as a graph with unknown value of the parameter T. The best way is to set $T = T(\pi)$, but actually we do not know $T(\pi)$. Let us assign labels $S_{i,j}$ to nodes in $H(\pi)$ interpreted as the start time of an operation $(i, j) \in O$, and discuss the relation between T and S.

From (6.8) it follows that $T(\pi)$ is the minimal value of T for which the feasible schedule S under constraints (6.4) – (6.7) still exists. It means that: (a) if $T \geq T(\pi)$ then the feasible S can be found, (b) if $T < T(\pi)$ no feasible S exists. Concerning the graph in case (b) we conclude that $H(\pi)$ contains a graph cycle with the length greater then zero because of too small value of T.

Considering the broader context of using graphs, $H(\pi)$ has at least two possible applications. The former is obvious and consists in finding the schedule S for $T = T(\pi)$ with known in advance $T(\pi)$. This needs a single run of, for example LA, applied to $H(\pi)$ and this T. The latter is to find $T(\pi)$ through the identification of the border value between regions of feasible and unfeasible S. This needs multiple runs of a modified LA for variable values of T. We present hereafter more details about the latter idea, including an extension of LA.

At first, we have equipped LA with graph cycle detector used in case (b), see extended version of LA+E in Fig. 6.3. After the "end for v" loop LA+E checks additionally whether labels S have been set in the feasible manner. If not, ERR indicates the lack of feasibility. Based on LA+E we have designed the following bi-partition method. Let us assume that the minimal cycle time $T(\pi)$ is located in an interval, it means $T(\pi) \in [T_*, T^*]$, where T_* is a lower bound, T^* is an upper bound of the interval. One expects that S found for T^* is feasible, but for T_* is unfeasible. Next we run LA+E for trial value $T = (T_* + T^*)/2$. If for this T algorithm LA+E returns ERR then $T(\pi) \in [T, T^*]$; otherwise, if returns OK then $T(\pi) \in [T_*, T]$. The partition is repeated until a stopping criteria will be met, for example $T^* - T_* \leq \epsilon$, for certain small value ϵ. From the formulation of LP task it follows that the num-

```
Parameter: $\pi$
   for $(i,j) \in O$ do $S_{i,j} = 0$;
   for $v = 1$ to $m \cdot n - 1$ do
      $tag = 0$;
      for $(((i,j),(k,l)) \in E(\pi) \cup F(\pi))$ do
         if $S_{k,l} < S_{i,j} + p_{i,j} + c_{i,j,k,l}$ then
            { $S_{k,l} = S_{i,j} + p_{i,j} + c_{i,j,k,l}$; $tag = 1$; }
         end if;
      end for $((i,j),(k,l))$;
      if not $tag$ then return OK;
   end for $v$;
   for $((i,j),(k,l)) \in E(\pi) \cup F(\pi)$ do
      if $S_{k,l} < S_{i,j} + p_{i,j} + c_{i,j,k,l}$ then return ERR;
   return OK;
```

Fig. 6.3 Algorithm LA+E

ber of LA+E calls in the bi-partition method until stop is unknown since $T(\pi)$ does not need to be integer. Notice, if ERR occurs, then the computational complexity of single run of LA+E is $\Theta(m^2 \cdot n^2)$, otherwise is $O(m^2 \cdot n^2)$ and $\Omega(m \cdot n)$. Therefore, finding $T(\pi)$ by a bi-partition method has the computational complexity $O(n^2 \cdot m^2 \cdot s)$, where s is the number of LA+E calls. Therefore, the bi-partition method with LA+E provides the algorithm of finding $T(\pi)$ with the pseudo-polynomial computational complexity. The method although theoretically interesting, might not be computationally attractive comparing to those presented in the next subsection.

One can ask whether algorithm LA$^+$ significantly faster than LA in the non-delay scheduling case has the similar dominance also for $H(\pi)$. The improvement from LA+E to LA$^+$+E remains true, however the potential profits are lesser comparing to LA+E because of the structure of paths in $H(\pi)$. Indeed, a graph cycle in $H(\pi)$ can go around the cylinder at most m times, which means that analyse of strips has to be repeated m times. The appropriate pseudo-code of LA$^+$+E is shown in Fig. 6.4. To check feasibility in LA$^+$+E we use the symbol $V \leftarrow E$, where V is a variable and E is an expression: if $V = E$ than do nothing, otherwise if $V < E$ than set $V = E$ and $tag = 1$.

The computational complexity of single run of LA$^+$+E is $\Omega(m \cdot n)$ and $O(m^2 \cdot n)$. To complete considerations of this subsection we provide the method of setting initial values T_* and T^* for the bi-partition method. From Property 3 we obtain immediately $T^* = C_{\max}(\pi)$. On the other hand, from (6.7) we have for $i = 1, \ldots, m$

$$T \geq S_{i,\pi(n)} + p_{i,\pi(n)} - S_{i,\pi(1)} \geq d_{i,\pi(1),i,\pi(n)} \tag{6.27}$$

```
Parameter: $\pi$
  for $i = 1 \ldots m$ do for $j = 1 \ldots n$ do $S_{i,j} = 0$;
  for $v = 1, \ldots, m$ do
    $tag = 0$;
    $k = \pi(1)$; $l = \pi(n)$; $S_{1,k} \leftarrow \max\{S_{1,k}, S_{1,l} + p_{1,l} - T\}$;
    for $i = 2, \ldots, m$ do
      $S_{i,k} \leftarrow \max\{S_{i,k}, S_{i-1,k} + p_{i-1,k} + a_{i-1,k}, S_{i,l} + p_{i,l} - T\}$
    end for $i$;
    for $i = m, \ldots, 2$ do
      $S_{i-1,k} \leftarrow \max\{S_{i-1,k}, S_{i,k} - p_{i-1,k} - b_{i-1,k}\}$
    end for $i$;
    for $j = 2, \ldots, n$ do
      $k = \pi(j)$; $l = \pi(j-1)$; $S_{1,k} \leftarrow \max\{S_{1,k}, S_{1,l} + p_{1,l}\}$;
      for $i = 2, \ldots, m$ do
        $S_{i,k} \leftarrow \max\{S_{i,k}, S_{i,l} + p_{i,l}, S_{i-1,k} + p_{i-1,k} + a_{i-1,k}\}$
      end for $i$;
      for $i = m, \ldots, 2$ do
        $S_{i-1,k} \leftarrow \max\{S_{i-1,k}, S_{i,k} - p_{i-1,k} - b_{i-1,k}\}$
      end for $i$;
    end for $j$;
    if not $tag$ then return OK;
  end for $v$;
  if $tag$ then return ERR;
  return OK;
```

Fig. 6.4 Algorithm LA$^+$+E

where $d_{i,\pi(1),i,\pi(n)}$ is the length of the longest path between start of node $(i, \pi(1))$ and end of node $(i, \pi(n))$ in $H(\pi)$. This finding allow us to set

$$T_* = \max_{1 \le i \le m} d_{i,\pi(1),i,\pi(n)}. \tag{6.28}$$

Appropriate algorithm of finding $d_{i,\pi(1),i,\pi(n)}$, $i = 1, \ldots, m$, can be obtained by a slight modification of LA to run m times from the graph sources $(i, \pi(1))$, $i = 1, 2, \ldots, m$. It has the computational complexity $O(m^2 \cdot n)$. Notice, in a general case (6.28) is not a tight bound (Fig. 6.5).

6.3.3 *Chain of Graphs*

$T(\pi)$ can be calculated precisely by using so-called chain graph $G^*(\pi)$. Briefly speaking, $G^*(\pi)$ is the concatenation of $m + 1$ identical graphs $G(\pi)$, see Fig. 6.6, to represent the sequence of $m + 1$ successive cycles indexed by $x = 1, 2, \ldots, m + 1$. The concatenation has been made with the help of additional arcs going from each last operation on a machine in some cycle to the first operation on this machine

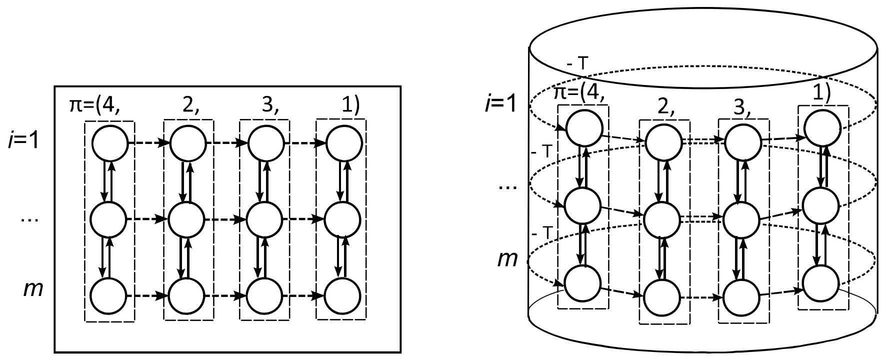

Fig. 6.5 Illustrative graph $G(\pi)$ for LA and strips used in LA* (left). Wrap-around-cylinder graph $H(\pi)$ for this instance (right)

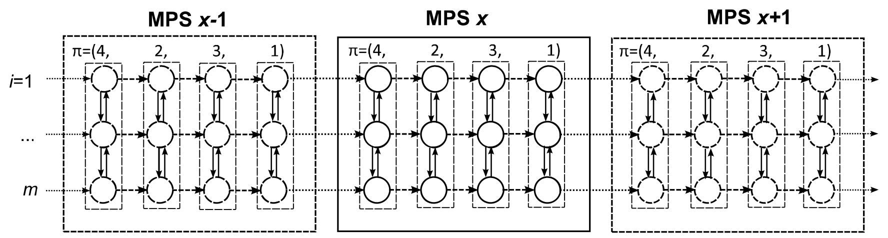

Fig. 6.6 Chain graph for the instance

in the next cycle. We will provide hereafter a more precise and formal description. In order to avoid too excessive mathematics, in this subsection we denote by single index j the operation $j \in O$ (actually, j is the pair). The graph has the form

$$G^*(\pi) = (O^*, E^*(\pi) \cup W^*(\pi)), \tag{6.29}$$

with a set of nodes

$$O^* = \bigcup_{x=1}^{m+1} O^x, \quad O^x = \{j^x : j \in O\}, \tag{6.30}$$

where O^x is the xth copy of the set O; we denote by j^x the operation j in the xth cycle. Likewise, we define the extended set of arcs

$$E^*(\pi) = \bigcup_{x=1}^{m+1} E^x(\pi), \quad E^x(\pi) = \{(i^x, j^x) : (i,j) \in E(\pi)\}, \tag{6.31}$$

where $E^x(\pi)$ is the xth copy of $E(\pi)$. To link copies of graphs we define additional arcs with no weights

$$W^*(\pi) = \bigcup_{x=1}^{m} W^x(\pi), \quad W^x(\pi) = \{((l, \pi(n))^x, (l, \pi(1))^{x+1}) : l \in M\}. \tag{6.32}$$

$G^*(\pi)$ can be perceived as an extension of $G(\pi)$, so we will introduce also the extended schedule vector $S_j, j \in O^*$.

Any path from i^x to $j^y, x \leq y$, in $G^*(\pi), y \leq m + 1$ can be described by a sequence of nodes from $\mathcal{O}^*$ occurring in this path

$$U_{i^x, j^y} = (u_0, u_1, \ldots, u_v), \tag{6.33}$$

where $u_0 = i^x$, $u_v = j^y$. Since there may exist many alternative paths from i^x to j^y, then we denote by $L(i^x, j^y)$ the length of the longest one among all paths from i^x to j^y, formally

$$L(i^x, j^y) = \max_U \sum_{i=0}^{v-1} (p_{u_i} + c_{u_i, u_{i+1}}), \tag{6.34}$$

where $c_{u_i,u_{i+1}}$ is the weight of the arc $u_i \rightarrow u_{i+1}$ and maximization runs over all paths U defined by (6.33). We supplement this definition setting $L(i^x, j^y) = \infty$ if path U does not exist. $L(i^x, j^y)$ depends on π, however for the convenience sake of notation we do not express this fact directly. Values $L(i^x, j^y)$, $i^x, j^x \in O^*$ can be found either by LA or LA* with respect to $G^*(\pi)$.

Property 4 *The calculation of $L(i^x, j^y)$ for any $i^x, j^y \in O^*$ in $G^*(\pi)$, can be done by LA in the time $O(m^6 n^3)$ and by LA$^+$ in the time $O(m^4 n^2)$.*

Graph $G^*(\pi)$ has $\Theta(m^2 n)$ nodes and $\Theta(m^2 n)$ of arcs, see (6.30)–(6.32). For each source node $i^x \in O^*$ we can run either LA or LA$^+$ to obtain paths from this i^x to all $j^y \in O^*$. Whence, the final computational complexity for LA is $O(m^2 n) \cdot O((m^2)^2 n^2) = O(m^6 n^3)$, whereas for LA$^+$ is $O(m^2 n) \cdot O(m^2 n) = O(m^4 n^2)$.

Considering relations between time events in the extended schedule S found on the base on $G(\pi)$, we conclude that

$$S_{j^y} - S_{i^x} \geq L(i^x, j^y). \tag{6.35}$$

Inequality (6.35) has an immediate application as follows.

Property 5 *We have*

$$T(\pi) = \max_{k=2,\ldots,m+1} \max_{l=1,\ldots,m} \frac{L(a_l^1, a_l^k)}{k - 1}, \tag{6.36}$$

where $a_l^k = ((l, \pi(1))^k$.

Now, we are ready to formulate the algorithm of finding $T(\pi)$.

Property 6 *$T(\pi)$ can be calculated from (6.36) by using graph $G^*(\pi)$ and LA in the time $O(m^5n^2)$ and by using LA$^+$ in the time $O(m^3n)$.*

The properties imply various optimization strategies: (A-1) use $C_{\max}(\pi)$ instead of $T(\pi)$ in order to evaluate the upper bound on a time window for cyclical schedule S (see Property 3), (A-2) find π^A by using an approximate method A with $C_{\max}(\pi)$ criterion, then calculate $T(\pi^A)$ once (see Property 3), (A-3) find π^A by using an approximate method A with $T(\pi)$ criterion (see Property 6).

It is clear, comparing Property 6 with Property 3, that variants (A-1) and (A-2) are faster than (A-3). The question "whether these algorithms remain competitive in terms of quality" will be verified next in computer tests.

6.3.4 Block Properties

The path in any graph we describe by a sequence of nodes, see (6.33). Path length is defined as the sum of node weights plus the arcs weights, see (6.34). The path $U^\pi = (w_0, \ldots, w_v)$ for which right-hand-side of (6.36) reaches the maximal value we call the *extended critical path* (ECP). U^π can be naturally decomposed into sub-sequences of operations having specific features: "pass through machine" and "pass through the job". Operations passing through the machine can be intermixed (more precisely jobs having these operations can be altered), while operations passing through the job cannot be intermixed due to belonging to the same job. Then, only the former type of sub-sequence can be considered in the context of possible improving $T(\pi)$. The maximal sub-sequence of the critical path containing nodes corresponding to successive operations processed on the same machine is called the *block* and is denoted by $B_{a:b}$, $a \leq b, a, b \in \{0, \ldots, v\}$. In each block we distinguish the first a and the last b operation of the block; the remaining operations $B_{a+1:b-1}$ constitute the *internal block*. Although notions block and internal blocks have appeared in many our earlier papers, see for example [13, 19], for the considered problem these notions are unusual. They have completely different meaning since ECP corresponds to several laps around the cylinder, but still behaves analogous elimination properties.

Property 7 *Let σ be the new processing order of jobs obtained from π by altering jobs corresponding to operations processed on certain machine. If $T(\sigma) < T(\pi)$, then at least one operation from at least one internal machine block is processed before the first or processed after the last operation of this block. Processing of jobs are adjusted accordingly.*

Property 7 is a nontrivial extension of the so-called block property from our earlier papers, [13, 19], which has become the flagship approach of our research team.

6.4 Parallel Methods

Two methods of parallel determination of the minimum cycle time for the determined order of operations π will be presented below, differing in their operation time and the required number of processors.

6.4.1 Parallel Determination of the Longest Paths in the Graph

Later in the work it will be required to quickly determine the longest paths in the graph between all pairs of vertices. Typically, a variant of the Warshall-Floyd algorithm with the longest paths (the shortest in the original being determined) is used sequentially, see Cormen et al. [7] for detail. However, when running the algorithms in a multiprocessor environment, it is preferable to parallelize the less efficient algorithm sequentially, called the longest path algorithm based on the Cormen et al. [7] matrix multiplication idea. This algorithm has a sequential complexity of $O(\frac{o^3}{\log o})$, where o is the number of vertices in the graph.

In the further part of the work, a parallel version of this algorithm will be proposed, with the computational complexity of $O(\log^2 o)$ with the use of $O(\frac{o^3}{\log o})$ processors. Algorithm starts from an idea of Gibbons and Rytter [10] for the shortest paths. It has been developed in the paper of Steinhöfel et al. [26] for the longest path and requires $O(o^3)$ processors, and then in the work [3] in the version for $O(\frac{o^3}{\log o})$ processors. As the algorithm requires simultaneous reading and there is no parallel writing to memory cells, the CREW PRAM (Concurrent Read Exclusive Write Parallel Random Access Machine) is an adequate model of parallel computing.

Let $A = [a_{ij}]_{o\times o}$ be the distance matrix of a certain graph, which is the input of the algorithm. As the output algorithm will return the matrix $A' = [a'_{ij}]_{o\times o}$ defined as follows:

$$a'_{ij} = \begin{cases} 0 & \text{for } i = j, \\ \max_{i=i_0,i_1,\ldots,i_k=j}\{a_{i_0 i_1} + a_{i_1 i_2} + \cdots + a_{i_{k-1} i_k}\} & \text{for } i \neq j, \end{cases} \tag{6.37}$$

where the maximum is determined for all possible strings of indices $i_0, i_1, i_2, \ldots, i_k$, so that $i = i_0$ and $j = i_k$. The value of a'_{ij} is the length of the longest path from the vertex i to j in the graph with the distance matrix A. The algorithm will additionally use auxiliary matrices: $M = [m_{ij}]_{o\times o}$ (two-dimensional) and $Q = [q_{ijk}]_{o\times o\times o}$ (three-dimensional).

Steps 1 and 3 can be performed in $O(1)$ constant time on $O(o^2)$ processors. In step two, the maximum can be calculated over time $O(\log o)$ with $O(\frac{o}{\log o})$ processors (see Bożejko [4]). By using o^2 processors, each of which has a group of $O(\frac{o}{\log o})$ processors to determine the maximum in parallel, the entire Step 2 can be performed

```
**Input** : $A$ matrix;
**Output**: $A'$ matrix;
{Step 1}
parfor all $(i,j)$, $i = 1,2,\ldots,o$, $j = 1,2,\ldots,o$ do
    $m_{ij} \leftarrow a_{ij}$;
end
{Step 2}
for $iter = 1,2,\ldots,\lceil \log_2 o \rceil$ do
    parfor all $(i,j,k)$, $i = 1,2,\ldots,o$, $j = 1,2,\ldots,o$, $k = 1,2,\ldots,o$ do
        $q_{ijk} \leftarrow m_{ij} + m_{jk}$;
    end
    parfor all $(i,j)$, $i = 1,2,\ldots,o$, $j = 1,2,\ldots,o$ do
        $m_{ij} \leftarrow \max\{m_{ij}, q_{i1j}, q_{i2j}, \ldots, q_{inj}\}$;
    end
    parfor all $(i,j)$, $i = 1,2,\ldots,o$, $j = 1,2,\ldots,o$ do
        $m_{ij} \leftarrow \max\{m_{ij}, q_{i1j}, q_{i2j}, \ldots, q_{ioj}\}$;
    end
end
{Step 3}
parfor all $(i,j)$, $i = 1,2,\ldots,o$, $j = 1,2,\ldots,o$ do
    if $i \neq j$ then
        $a'_{ij} \leftarrow m_{ij}$;
    else
        $a'_{ij} \leftarrow 0$;
    end
end
```

Algorithm 1: `ParLongestPaths(A,A')`

in $O(\log^2 o)$ using $O(\frac{o^3}{\log o})$ processors, which determines the complexity and the number of processors required for the entire method.

6.4.2 Parallel Determination of the Minimum Cycle Time

Property 8 *The value of the minimum cycle time in the problem under consideration can be determined in time $O(\log^2 o)$ on $O(\frac{o^3}{\log o})$-processors CREW PRAM.*

Proof The longest paths in a chain graph $G^*(\pi)$ can be determined by the `ParLongestPaths` algorithm in the time $O(\log^2 m)$ on $O(\frac{m^3}{\log m})$-processors CREW PRAM (substituting the number of machines m in place of o). As an result we will obtain a matrix of m^2 lower bounds of the minimal cycle time, from which at least one—the greatest one—is an exact value of the minimal cycle time $T(\pi)$. The minimum of m^2 values can be determined over time $O(\log m^2) = O(2\log m) = O(\log m)$ on a CREW PRAM with $O(\frac{m^2}{\log m^2}) = O(\frac{m^2}{\log m}) = O(\frac{m^3}{\log m})$ processors. Additionally, the time required to determine the weights in the $G^*(\pi)$ graph should be taken into account, which (using the already mentioned `ParLongestPaths` algorithm applied to the $G(\pi)$ graph) is $O(\log^2 o)$ on $O(\frac{o^3}{\log o})$-processors the CREW PRAM and determines the complexity of $O(\max\{\log^2 o, \log^2 m\}) = O(\log^2 o)$ and the number of processors $O(\max\{\frac{o^3}{\log o}, \frac{m^3}{\log m})\}) = O(\frac{o^3}{\log o})$ of the entire method. The last inequality is due to the fact that $o > m$.

Property 9 *The value of the minimum cycle time in the problem under consideration can be determined in time $O(o + \log^2 m)$ using $O(\frac{m^3}{\log m})$-processors the CREW PRAM.*

Proof The procedure presented in the proof of the previous theorem should be used, with the difference that the longest paths in chain graph $G^*(\pi)$ should be determined using the method based on a vertex review in topological order, analogically to the LA$^+$ algorithm. This method has a sequential complexity of $O(mo) = O(m^2n)$ (the number of operations $o = m \cdot n$) due to the fact that m times it must perform the vertex review procedure in topological order, starting each time from the next source—a vertex representing the execution of the first operation on the machine. This routine can be easily paralleled by using m processors and getting $O(o)$ parallel run-time.

The remaining elements remain the same as in the proof of Property 8, i.e., the longest paths in time should be found in $O(\log^2 m)$ with using $O(\frac{m^3}{\log m})$ processors, and then find the maximum from m^2 calculated values, in time $O(\log m)$ with $O(\frac{m^2}{\log m}) = O(\frac{m^3}{\log m})$ processors. The total time will then be $O(\max\{o, \log^2 m\}) = O(o + \log^2 m)$ and $O(\max\{m, \frac{m^3}{\log m})\}) = O(\frac{m^3}{\log m})$ processors.

Despite the worse computation time of the method related to the Property 9, it may be in practice more beneficial from a practical point of view, because it requires a much smaller number of processors.

6.5 Application

A firm cooperating with our University would like to optimize the process of producing building blocks made of the cellular concrete. The factory provides blocks in combination of various sizes and various physical features fitted to the building construction purposes. The technological process consists of eighth stages performed consecutively, see Grabowski and Pempera [12] for details:

1. mixing the concrete ingredients (depending on the concrete sort),
2. filling the chosen size mould by a liquefied concrete,
3. primal establishing concrete in a mould to get a soft texture,
4. snipping blocks from soft concrete,
5. product quality control,
6. hardening of blocks in specific conditions (pressure, temperature),
7. transportation of blocks to the warehouse,
8. transport of blocks to clients.

A fixed portion of blocks (their number, geometry, physical properties) ordered by a client is called the production task. There is a pre-defined set of various tasks to be performed in the factory. Because of the technology requirements, there exist bounds on waiting times between operations 1 and 2, 2 and 3, 3 and 4. Indeed, filling the form can be performed immediately after the mixing components step, but not too late because of the fast hardening process of liquefied concrete in the mixing unit. After the filling the mould in step 2, the primal hardening of the concrete in the mould (step 3) should be in some range of waiting time given by appropriate lower/upper bounds. Snipping operation in step 4 cannot be preformed too late after operation 3 because of possible hardening of the concrete in the mould. The optimization process can be considered in two contexts: (a) having the given the set of tasks, we would like to find the schedule minimizing the makespan, (b) having the set of tasks performed in the repetitive way, we would like to find the cyclic schedule minimizing the cycle time.

6.6 Conclusions

To our best knowledge, the proposed modelling fashion of "limited waiting-time" constraints as well as cyclical schedule are unusual. Various graphs play the significant role in the modeling as well as the solution technology. The problem has been decomposed into two sub-problems: (1) to find the optimal processing order, and (2) to find the minimal cycle time for the given processing order. For the sub-problem (2) we have proposed a few methods based on graph properties as well as parallel computations. For the sub-problem (1) we have introduced through graphs some elimination properties for local neighbourhood search metaheuristics.

Further research are needed to verify experimentally theoretical findings and to compare proposed approaches. The whole approach can be extended on other, more complex cyclic scheduling problems (as an example job-shop), problems with buffers, and so on. Each of proposed algorithm can be embedded in any meta-heuristic algorithm seeking the best job processing order π. In turn, the proposed methodology of parallel determination of the minimum cycle time can be used both on the multi-core CPU platform and GPU co-processor platform, which has recently been growing in popularity.

Acknowledgements The paper is partially supported by the National Science Centre, grant OPUS no. DEC 2017/25/B/ST7/02181.

References

1. Bocewicz, G., Muszynski, W., Banaszak, Z.: Models of multimodal networks and transport processes. Bulletin of the Polish Academy of Sciences: Technical Sciences **63**(3), 635–650 (2015)
2. Bocewicz, G., Nielsen, I., Banaszak, Z., Majdzik, P.: A cyclic scheduling approach to maintaining production flow robustness. Adv. Mech. Eng. **10**(1) (2018)
3. Bożejko, W.: Solving the flow shop problem by parallel programming. J. Parallel Distrib. Comput. **69**, 470–481 (2009)
4. Bożejko, W.: A new class of parallel scheduling algorithms. Oficyna Wydawnicza Politechniki Wroclawskiej textit1 (2010)
5. Brucker, P., Kampmeyer, T.: Cyclic job shop scheduling problems with blocking. Annals of Operations Research **159**(1), 161–181 (2008a)
6. Brucker, P., Kampmeyer, T.: A general model for cyclic machine scheduling problems. Discrete Appl. Math. **156**(13), 2561–2572 (2008)
7. Cormen, T.H., Leiserson, C.E. Rivest, R.L.: Introduction to algorithms (revised ed), 2nd edn. MIT Press, Cambridge (2001)
8. Dios, V., Fernandez-Viagas Framinan, M.J.: Efficient heuristics for the hybrid flow shop scheduling problem with missing operations. Comput. Ind. Eng. **115**, 88–99 (2018)
9. Dolgui, A., Ivanov, D., Sethi, S.P. Sokolov, B.: Scheduling in production, supply chain and industry 4.0 systems by optimal control: fundamentals, state-of-the-art and applications. Int. J. Prod. Res. **5**(2), 411–432 (2019)
10. Gibbons, A., Rytter, W.: Efficient Parallel Algorithms. Cambridge University Press, Cambridge (1988)
11. Gilmore, P., Gomory, R.: Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. Operations Research **12**(5), 655–679 (1964)
12. Grabowski, J., Pempera, J.: Sequencing of jobs in some production system. European Journal of Operational Research **125**(3), 535–550 (2000)
13. Grabowski, J., Pempera, J.: New block properties for the permutation flow shop problem with application in tabu search. Journal of Operational Research Society **52**(2), 210–220 (2001)
14. Grabowski, J., Pempera, J.: Some local search algorithms for no-wait flow-shop problem with makespan criterion. Computers & Operations Research **32**(8), 2197–2212 (2005)
15. Hall, N.G., Sriskandarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. Operations Research **44**(3), 510–525 (1996)
16. Henneberg, M., Neufeld, J.: A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations. International Journal of Production Research **54**(12), 3534–3550 (2016)
17. Howard, R.: Dynamic programming and markov processes. Technology Press and Wiley, New York, NY (1960)
18. McCormick, S., Pinedo, M., Shenker, S., Wolf, B.: Sequencing in an assembly line with blocking to minimize cycle time. Operations Research **37**(6), 925–935 (1989)
19. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research **91**(1), 160–175 (1996)
20. Pempera, J., Smutnicki, C.: Open shop cyclic scheduling. European Journal of Operational Research **269**(2), 773–781 (2018)
21. Pinedo, M.: Scheduling, vol. 5. Springer, Berlin (2012)
22. Rajendran, C.: A no-wait flowshop scheduling heuristic to minimize makespan. Journal of the Operational Research Society **45**(4), 472–478 (1994)
23. Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research **165**(2), 479–494 (2005)
24. Saravanan, M., Sridhar, S., Harikannan, N.: Minimization of mean tardiness in hybrid flow shop with missing operations using genetic algorithm. Journal of Advanced Manufacturing Systems **15**(02), 43–55 (2016)

25. Smutnicki, C.: A new approach for cyclic manufacturing. In: International Conference on Intelligent Engineering Systems, pp. 275-280 (2018)
26. Steinhöfel, K., Albrecht, A., Wong, C.K.: Fast parallel heuristics for the job shop scheduling problem. Computers & Operations Research **29**, 151–169 (2002)
27. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)
28. Tseng, C., Liao, C., Liao, T.: A note on two-stage hybrid flowshop scheduling with missing operations. Computers & Industrial Engineering **54**(3), 695–704 (2008)