10.1515/acsc-2017-0009

Archives of Control Sciences
Volume 27(LXIII), 2017
No. 2*, pages 151–167*

Cyclic flow shop scheduling problem with two-machine cells

WOJCIECH BOŻEJKO, ANDRZEJ GNATOWSKI, RADOSŁAW IDZIKOWSKI and MIECZYSŁAW WODECKI

In the paper a variant of cyclic production with setups and two-machine cell is considered. One of the stages of the problem solving consists of assigning each operation to the machine on which it will be carried out. The total number of such assignments is exponential. We propose a polynomial time algorithm finding the optimal operations to machines assignment.

Key words: job shop, cyclic scheduling, multi-machine, assignment.

1. Introduction

Cells equipped with machines of the same types but with different efficiencies are elements of various manufacturing systems. Workpieces flow through a cell in a determined order and are processed on an assigned machine. In cyclic scheduling problems, a determined set of jobs (called MPS, *Minimal Part Set*) is performed multiple times at constant intervals called the cycle time. In another words, each operation of the job is executed on the same machine cyclically. A comprehensive introduction to the problems of cyclic scheduling includes work of Kampmeyer [8].

Processing times of operations on an assigned machine have a direct influence onto the length of the cycle (which is usually minimized). Additionally, machine setups between adjacent operations are considered, therefore minimal cycle time determination constitutes an NP-hard optimization problem, as it comes down to solving a particular traveling salesman problem (Bożejko, Uchroński, Wodecki [1]).

In the paper, we consider the cycle time minimization problem in two-machine cells, which are elements of the non-permutational (the order of operations in each cell can be different) Cyclic Flow Shop (FSP) manufacturing system. The problem, which will be referred to as CFSAP in this paper, is obviously more complex then classic FSP,

W. Bożejko (corresponding author), A. Gnatowski and R. Idzikowski are with Department of Automatics, Mechatronics and Control Systems, Faculty of Electronics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland. E-mails: {wojciech.bozejko, andrzej.gnatowski, radoslaw.idzikowski}@pwr.edu.pl. M. Wodecki is with Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland, e-mail: mieczyslaw.wodecki@uwr.edu.pl

The paper is supported by the funds for the development of young researchers.

Received 14.12.2016. Revised 27.04.2017.

as one must determine (i) an operations to machines assignment and also (ii) an order of operations execution on each machine; such that the cycle time (Bożejko, Wodecki [4]) is minimal. A review of computational complexity of the algorithms solving cyclic scheduling problems can be found in Levner i in. [9], similar makespan minimization problem is researched in Sawik [11].

The cycle time in each cell is minimized as it can have an impact onto the cycle time of the whole system. Because of the non-permutational aspect of the problem, each cell can be solved separately. On a top level a modified Tabu Search algorithm is used to determine the permutations in each cell. On the lower level, there are 2^o possible operations-to-machines assignments in each cell, where o is the number of operations. The algorithm with polynomial computational complexity, determining (for a given permutation of operations) the minimal cycle time in each two-machine cell is proposed.

2. Problem formulation

The considered problem consists of finding such permutations of operations in cells and such assignment of operations to one of two machines in each cell, that the cycle time of a manufacturing process is minimal. In the paper we consider non-permutational flow shop scheduling problem with two-machine cells (CFSAP), where (i) the order of operations in a cell is independent from other cells' operations orders and (ii) optimization criterion is a cycle time, which equals to the longest processing time of operations in a cell of the whole system. Due to these two properties, any instance of CFSAP can be divided into q independent, one-cell subproblems (Cyclic Permutation and Assignment Problems, CPAPs), where q is the number of cells. Let T_i denote the minimal cycle time obtained for the CPAP subproblem consisting of the cell i only. Then, the minimal cycle time obtained for CFSAP can be obtained from equation $T = \min\{T_i : i \in \{1,2,\dots,q\}\}$. For the sake of notation simplicity, hereinafter a single two-machine cell is considered (CPAP).

CPAP can be formulated in a following way: a set of operations $O = \{1,2,\dots,o\}$, which must be executed on machines from a set $\mathcal{M} = \{1,2\}$ (constituting a cell) is given. The processing order of the operations (sometimes refereed to later as *permutation*) can be represented as a tuple $\pi = (\pi(1),\pi(2),\dots,\pi(o)) \in \Pi$, where Π is a set of all the possible orders. Each operation $i \in O$ must be being executed uninterruptible on the assigned machine $l \in \mathcal{M}$ for p_i^l time, wherein in the cell at most one operation can be processed at the same time. The assignment $P = (\mathcal{Z}_1, \mathcal{Z}_2)$ is defined by the two disjoint sets $\mathcal{Z}_1, \mathcal{Z}_2$ of operations executed on the machines 1 and 2 respectively (of course $\mathcal{Z}_1 \cup \mathcal{Z}_2 = O$). The set of all the possible assignments

$$\mathcal{P} = \bigcup_{I \in \wp(O)} \{(\mathcal{Z}_1 = I,\ \mathcal{Z}_2 = O \setminus \{I\})\}, \tag{1}$$

where $\wp(O)$ is an exponential set and has a cardinality of $|\mathcal{P}| = 2^o$. For a given assignment P, a tuple

$$\pi_P^l = (\pi_P^l(1), \pi_P^l(2), \ldots, \pi_P^l(|\pi_P^l|)),\ l \in \mathcal{M}, \tag{2}$$

defines processing order of the operations a machine l, and $v_P(i)$ the machine on which operation i is executed. Between each two adjacent operations on a machine $l \in \mathcal{M}$, a setup with a duration of

$$s^l_{\pi_P^l(i), \pi_P^l(i+1)},\ l \in \mathcal{M},\ i \in \{1, 2, \ldots, |\pi_P^l| - 1\}, \tag{3}$$

must be done, when no other setups can take place, nor operations can be executed. Additionally, due to the cyclic character of the problem, an initial setup

$$s^l_{\pi_P^l(|\pi_P^l|), 1},\ l \in \mathcal{M}, \tag{4}$$

must be done before the first operations of each MPS on each machine.

The solution of CPAP consists of the operations-to-machines assignment and times of operations starts and finishes in consecutive MPSes. Starting times of operations in x-th MPS are denoted by $S^x = (S_1^x, S_2^x, \ldots, S_o^x)$ and by $C^x = (C_1^x, C_2^x, \ldots, C_o^x)$ finishing times. These sequences must fulfill following constrains:

$$\forall i \in \{1, 2, \ldots, o\} \quad C^x_{\pi(i)} = S^x_{\pi(i)} + p^{v_P(\pi(i))}_{\pi(i)}, \tag{5}$$

$$\forall i \in \{1, 2, \ldots, o\} \quad S^{x+1}_{\pi(i)} = S^x_{\pi(i)} + T, \tag{6}$$

$$\forall i \in \{2, 3, \ldots, o\} \quad S^x_{\pi(i)} \geqslant C^x_{\pi(i-1)} + s_P^\pi(\pi(i-1)), \tag{7}$$

$$S^{x+1}_{\pi(1)} \geqslant C^x_{\pi(o)} + s_P^\pi(\pi(o)), \tag{8}$$

where x denotes the MPS number, T is the cycle time and

$$s_P^\pi(\pi_P^l(i)) = \begin{cases} s^l_{\pi_P^l(i), \pi_P^l(i+1)} & \text{for } i = \{1, 2, \ldots, |\pi_P^l| - 1\} \\ s^l_{\pi_P^l(i), \pi_P^l(1)} & \text{for } i = |\pi_P^l| \end{cases},\ l \in \mathcal{M}, \tag{9}$$

is the setup time before execution of an operation $\pi_P^l(i) \in \mathcal{Z}_l$ for an assignment P and permutation π. The constrain (5) ensures the uninterruption of operations execution and the equation(6) its cyclic character. The equation (7) represents setups within, and the equation (8) between MPSes.

For an assignment P and permutation π, let $T(P, \pi)$ denote the minimal time of a cell work (cycle time), for which sequences S^x and C^x fulfilling constrains (5)–(8) exist. It is easy to observe, that

$$T(P, \pi) = \sum_{i=1}^{o} \left(p_i^{v_P(i)} + s_P^\pi(i) \right). \tag{10}$$

The Cyclic Permutation and Assignment Problem boils down to finding such $P^* \in \mathcal{P}$ and $\pi^* \in \Pi$ that minimizes equation (10).

3. Graph model

As discussed in the previous section, each cell constitutes a separate subproblem of finding the optimal order (permutation) and assignment of operations. In the paper, two-level problem solving approach is devised:

Level 1 Search for the optimal permutation by altering the operations order of execution only. Evaluate each solution, assuming that the optimal assignment for any given permutation is known.

Level 2 For a given permutation, calculate the optimal assignment minimizing equation (10).

In the following section, a graph model used in solving Level 2 is presented. Therefore, without loss of generality, following assumptions constituting Cyclic Assignment Problem (CAP) are taken: (i) there is only one cell (each cell is a separate subproblem); (ii) the permutation is natural $\pi = (1,2,\dots,o)$ and therefore omitted (since operations in the cell can be renumbered).

The graph presented below cannot be used to model the assignments in which all the operations are executed on a single machine (constituting the set P^Z, the assignments from the set P^Z can be evaluated in $O(o)$ time). Therefore, from now on unless stated otherwise, only the assignments from the set $\mathcal{P} \setminus P^Z$ are considered.

Directed graph $\mathcal{A}$ is defined as follows:

$$\mathcal{A} = (\mathcal{W} \cup \mathcal{W}',\ \mathcal{E} \cup \mathcal{E}'). \tag{11}$$

where $\mathcal{W}$ i $\mathcal{W}'$ are sets of vertices, $\mathcal{E}$ and $\mathcal{E}'$ are sets of arcs, such that:

$$\mathcal{W} = \bigcup_{i \in \mathcal{M}} \bigcup_{j=1}^{o} \{j^i\}, \qquad \mathcal{W}' = \bigcup_{i \in \mathcal{M}} \bigcup_{j=o+1}^{2o} \{j^i\}, \tag{12}$$

$$\mathcal{E} = \bigcup_{a \in \mathcal{M}} \bigcup_{i=1}^{o-1} \bigcup_{j=i+1}^{o} \left\{ \left(i^a, j^b\right) \right\}, \qquad \mathcal{E}' = \bigcup_{a \in \mathcal{M}} \bigcup_{i=2}^{o} \bigcup_{j=o}^{o-1+i} \left\{ \left(i^a, j^b\right) \right\}, \tag{13}$$

where $b \in \mathcal{M} \setminus \{a\}$. Vertex $i^a \in \mathcal{W}$ matches the operation i executed on the machine a, while vertex $j^a \in \mathcal{W}'$, respectively, a copy of the operation $j-o$ from next MPS. Set $\mathcal{E}$ consists of arcs between the vertices of $\mathcal{W}$; $\mathcal{E}'$ between the vertices of the set $\mathcal{W}$ and $\mathcal{W}'$. An example of the graph $\mathcal{A}$ for the number of operations $o = 5$ is presented in Figure 1.

Vertices in a graph $\mathcal{A}$ have no weights. A weight of an arc $\left(i^a, j^b\right) \in \mathcal{E} \cup \mathcal{E}'$ is the sum of execution and setup times of the operations (or their copies from the next MPS) from i to $j-1$. The weights can be calculated from the formula

$$d\left(i^a, j^b\right) = p'^a_i + s'^a_{i,j+1} + \sum_{k=i+1}^{j-1} \left(p'^b_k + s'^b_{k,k+1}\right) \tag{14}$$

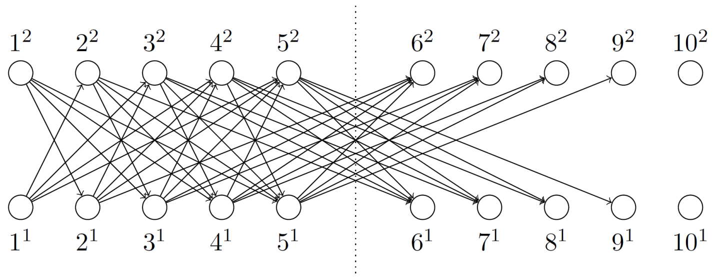

Figure 1: Graph $\mathcal{A}$ for $o = 5$.

where

$$s'^{a}_{i,j} = s^{a}_{((i-1) \bmod o)+1,\ ((j-1) \bmod o)+1},\ a \in \mathcal{M},\ i \in \mathcal{O},\ j \in \mathcal{O} \setminus \{i\}, \tag{15}$$

is the setup time between operations matching the vertices i^a and j^b, while

$$p'^{a}_{i} = p^{a}_{((i-1) \bmod o)+1}, \quad a \in \mathcal{M},\ i \in \mathcal{O} \tag{16}$$

is the execution time of the operation represented by the vertex i^a.

For a given assignment $P \in \mathcal{P} \setminus P^Z$, a tuple $\pi'_P = (\pi'_P(1), \pi'_P(2), \ldots,\ \pi'_P(|\pi'_P|))$ denotes all the operations with following operations executed on a different machine

$$\forall i \in \mathcal{O} \setminus \{o\} \quad v_P(i) \neq v_P(i+1) \Rightarrow i \in \pi'_P, \tag{17}$$

$$v_P(o) \neq v_P(1) \Rightarrow o \in \pi'_P, \tag{18}$$

preserving the order of executed operations from π.

For example, for $P = (\{2,3,5\},\{1,4,6\})$, $\pi'_P = (1,3,4))$. Then, let

$$\nu(P) = (\nu_1(P), \nu_2(P), \ldots, \nu_{|\pi'_P|+1}(P)), \tag{19}$$

where:

$$\nu_k(P) = \begin{cases} \pi'_P(k)^{v_P(\pi'_P(k))} & \text{for } k \in \{1,2,\ldots,|\pi'_P|\}, \\ (\pi'_P(1)+o)^{v_P(\pi'_P(1))} & \text{for } k = |\pi'_P|+1. \end{cases} \tag{20}$$

be a path defined by its vertices in the graph $\mathcal{A}$.

Definition 1 *A path* $(i^a, (i+1)^b, (i+2)^a, \ldots, (i+o)^a)$, $a, b \in \mathcal{M}$, $a \neq b$, $i \in \mathcal{O} \setminus \{o\}$ *in the graph* $\mathcal{A}$ *is called the highlighted path. A set of all such paths is denoted by* $\mathcal{N}$.

An example of the *highlighted* path for the assignment $P = (\{1,4,6\},\ \{2,3,5\})$ and the number of operations $o = 5$ is presented in Figure 2. Black circles correspond to the assignment.

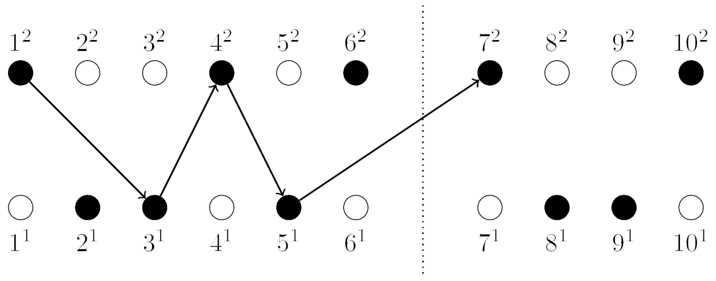

Figure 2: Exemplary *highlighted* path in graph $\mathcal{A}$.

Lemma 1 *For any highlighted path $\mu \in \mathcal{N}$, corresponding assignment $P \in \mathcal{P} \setminus P^Z$, such that $\nu(P) = \mu$ exists; i.e.*

$$\forall \mu \in \mathcal{N},\ \exists P \in \mathcal{P} \setminus P^Z \quad \nu(P) = \mu. \tag{21}$$

Proof Let $\mu = (\mu_1 = i^a, \mu_2, \ldots, \mu_{|\mu|} = (i+o)^a)$ be a highlighted path connecting vertices i^a and $(i+o)^a$, $1 \leqslant i \leqslant o$, $a \in \mathcal{M}$. Sequences φ and γ are defined in such a way, that any vertex from the path $\mu_k \in \mu$ can be expressed as $\mu_k = \varphi_k^{\gamma_k}$, $k \in \{1, 2, \ldots, |\mu|\}$. It is easy to see that for the assignment $P = (\mathcal{Z}_a, \mathcal{Z}_b)$, where

$$\mathcal{Z}_a = \{1, \ldots, \varphi_1\} \cup \{\varphi_{|\varphi|-1} + 1, \ldots, n\} \cup \bigcup_{k=1}^{|\varphi|/2} \bigcup_{l=\varphi_{2k-1}}^{\varphi_{2k}} \{l\}, \tag{22}$$

$$\mathcal{Z}_b = \mathcal{O} \setminus \mathcal{Z}_a, \tag{23}$$

and $a = \gamma_1$, there is $\nu(P) = \mu$. $\square$

Lemma 2 *For any assignment $P \in \mathcal{P} \setminus P^Z$, a path $\nu(P)$ in the graph $\mathcal{A}$ is a highlighted path, i.e.:*

$$\forall P \in \mathcal{P} \setminus P^Z,\ \exists \mu \in \mathcal{N} \quad \nu(P) = \mu. \tag{24}$$

Proof The Lemma 2 results directly from the definition of path $\nu(P)$.

Theorem 1 *Function $f : \mathcal{P} \setminus P^Z \to \mathcal{N}$, $f(P) = \nu(P)$ is bijective.*

Proof Function f must fulfill properties of bijection:

$$\forall \mu \in \mathcal{N},\ \exists P \in \mathcal{P} \setminus P^Z \quad f(P) = \nu(P) = \mu, \tag{25}$$

$$\forall P \in \mathcal{P} \setminus P^Z,\ \exists \mu \in \mathcal{N} \quad f(P) = \nu(P) = \mu, \tag{26}$$

$$\forall P_1, P_2 \in \mathcal{P} \setminus P^Z \quad f(P_1) = f(P_2) \Leftrightarrow P_1 = P_2. \tag{27}$$

From Lemmas 1 and 2, conditions (25) and (26) are met. Let us consider first $|\nu(P)| - 1$ vertices of a path $\nu(P)$. From the definition, $\nu_k(P) = \pi'_P(k)^{\nu_P(\pi'_P(k))}$,

$k \in \{1,2,\ldots,|\nu_P|-1\}$. Since permutation π'_P includes all the operations followed by the changes in the machine assignments, it is easy to see that

$$\forall P_1, P_2 \in \mathcal{P} \quad (P_1 = (\mathcal{Z}_1, \mathcal{Z}_2) \wedge P_2 = (\mathcal{O} \setminus \mathcal{Z}_1, \mathcal{O} \setminus \mathcal{Z}_2) \vee P_1 = P_2) \Leftrightarrow \pi'_{P_1} = \pi'_{P_2}. \tag{28}$$

The case $P_1 = (\mathcal{Z}_1, \mathcal{Z}_2)$, $P_2 = (\mathcal{O} \setminus \mathcal{Z}_1, \mathcal{O} \setminus \mathcal{Z}_2)$ can be rejected because then: $v_{P_1}(k) \neq v_{P_2}(k)$, $k \in \mathcal{O}$. Hence, the condition from the equation (27) is fulfilled. $\square$

The consequence of the Theorem 1 is an existence of the inverse function to f, $g(f(P)) = P$, $P \in \mathcal{P} \setminus P^Z$. Bijective function g assigns a highlighted path to an assignment from the set $\mathcal{P} \setminus P^Z$.

Lemma 3 *For any assignment $P \in \mathcal{P} \setminus P^Z$, the sum of weights of arcs of highlighted path $\nu(P)$ is equal to the minimal cycle time $T(P)$*

$$d(\nu(P)) = T(P). \tag{29}$$

Proof The proof is based on calculation of sum of weights of arcs of path $\nu(P)$

$$d(\nu(P)) = \underbrace{\sum_{k=1}^{|\pi'|-1} d((\nu_k(P), \nu_{k+1}(P)))}_{X(P)} + \underbrace{d((\nu_{|\pi'|}(P), \nu_{|\pi'|+1}(P)))}_{Y(P)}, \tag{30}$$

where $X(P)$ is the sum of weights of arcs belonging to set $\mathcal{E}$ and $Y(P)$ to set $\mathcal{E}'$. Values $X(P)$ and $Y(P)$ can be determined by Eq. (14). After transformations (described in detail in report [7])

$$\begin{aligned} d(\nu(P)) =& X(P) + Y(P) = \\ &= \sum_{k=\pi'_P(1)}^{\pi'_P(|\pi'_P|)-1} \left(p_k^{v_P(k)} + s_P^{\alpha}(k)\right) + \sum_{k=\pi'_P(|\pi'_P|)}^{o} \left(p_k^{v_P(k)} + s_P^{\alpha}(k)\right) + \\ &+ \sum_{k=1}^{\pi'_P(1)-1} \left(p_k^{v_P(k)} + s_P^{\alpha}(k)\right) = \\ &= \sum_{k=1}^{o} \left(p'^{v(k)}_k + s_P^{\alpha}(k)\right). \end{aligned} \tag{31}$$

right sides of Eqs. (31) and (10) are equal, therefore $d(\nu(P)) = T(P)$. $\square$

Theorem 2 *In graph $\mathcal{A}$, weight of highlighted path with the minimum weight is equal to the minimal cycle time $T(P)$ for $P \in \mathcal{P} \setminus P^Z$; i.e.*

$$\arg\max_{\mu \in N} \{d(\mu)\} = \nu(\arg \max_{P \in \mathcal{P} \setminus P^Z} \{T(P)\}). \tag{32}$$

Proof From Lemma 3 and Theorem 1

$$\arg\max_{\mu\in N}\{d(\mu)\} \overset{\text{th. 1}}{=} \arg \max_{\mu\in \bigcup\limits_{P\in\mathcal{P}\setminus P^Z}\{\nu(P)\}} \{d(\mu)\} =$$

$$= \nu\big(\arg \max_{P\in\mathcal{P}\setminus P^Z}\{d(\nu(P))\}\big) \overset{\text{lm. 3}}{=} \nu\big(\arg \max_{P\in\mathcal{P}\setminus P^Z}\{T(P)\}\big).$$

□

The obvious consequence of Theorem 2 is the equation

$$\max_{\mu\in\mathcal{N}}\{d(\mu)\} = \max_{P\in\mathcal{P}\setminus P^Z}\{T(P)\}. \tag{33}$$

4. Solving CAP

In this section, two CAP solving algorithms are presented and their computational complexity is discussed.

4.1. The polynomial algorithm (PA)

Proposed algorithm utilizes graph $\mathcal{A}$ (described in previous section) to determine the optimal assignment $P^* \in \mathcal{P}$, and therefore solve CAP. The algorithm is summarized in Algorithm 1.

Algorithm 1 The polynomial algorithm (PA)

```
1: Construct graph $\mathcal{A}$.
2: for all $i \in O \setminus \{o\}$ do
3:     for all $a \in \mathcal{M}$ do
4:         Find the path with minimum weight from vertex $i^a$ to $(i+o)^a$.
5: From paths obtained in steps 3–5, choose the path with minimal weight and determine the corresponding assignment $P_1 \in \mathcal{P} \setminus P^Z$.
6: Calculate the minimal cycle time $T(P)$ of the individually analyzed cases $P_2 = (\varnothing, O)$ and $P_3 = (O, \varnothing)$.
7: return $P^* = \arg\min_{P\in\{P_1,P_2,P_3\}}\{T(P)\}$
```

In lines 2–5. the algorithm determines the highlighted path with minimal weight

$$\min_{P\in\mathcal{P}\setminus P^Z}\{d(\nu(P))\} = \min_{P\in\mathcal{P}\setminus P^Z}\{T(P)\} \tag{34}$$

and thus, by Theorem 2, the corresponding assignment from $\mathcal{P} \setminus P^Z$ with the minimal cycle time. In line 7., the values of $T(P)$ for $P \in P^Z$ are calculated, hence the algorithm determines the optimal solution.

Theorem 3 *For the Cyclic Assignment Problem, the optimal assignment P^* minimizing the cycle time can be determined in $O(o^3)$ time.*

Proof The proof is based on the analysis of the computational complexity of Alg. 1. Constructing graph $\mathcal{A}$ from line 2. requires calculation of $O(o^2)$ weight of arcs, where each weight is the sum of $O(o)$ elements, hence the computational complexity is $O(o^3)$. Line 5. can be realized by sequentially determining the longest path from initial vertex to the following vertices (in topological order). Because there are $O(o)$ in- and out-arcs from each vertex, complexity of the lines 5. equals $O(o^2)$. The operations from line 5. are performed $O(o)$ times (lines 3–5.), resulting in $O(o^3)$ time complexity. Line 7. comes down to determining the minimal cycle time for separately evaluated assignments from P^Z. They can be calculated from the formula (10) in $O(o)$ time. Finally, the computational complexity of the algorithm equals $O(o^3) + O(o^3) + O(o) = O(o^3)$. □

4.2. One Opt algorithm

One opt algorithm (1-Opt) is a heuristic for CAP. Pseudocode for the algorithm is presented in Algorithm 2.

Algorithm 2 One Opt

```
1: for all $i \in O$ do
2:     $P' \leftarrow (Z_{v_P(i)} \setminus \{i\},\ Z_{\{1,2\} \setminus v_P(i)} \cup \{i\})$
3:     if $T(P) > T(P')$ then
4:         $P \leftarrow P'$
5:         goto line 1.
6: return $P$
```

In each iteration, an assignment of each operation (one at a time) is temporarily changed. If the change lowers the minimal cycle time, it becomes permanent. The algorithms stops when no change in an assignment of a single operation can improve the minimal cycle time.

Since there are 2^o different assignments and $T(P)$ can be calculated in $O(o)$ time, the total computational complexity of 1-Opt cannot exceed $O(o \cdot o \cdot 2^o) = O(o^2 2^o) = O(2^o)$. It is worth noting, that as shown in the computational experiments, in practical applications, 1-Opt can be much faster then the PA algorithm (with $O(o^3)$ computational complexity).

5. Solving CFSAP with Tabu Search algorithm

CFSAP consists of q independent subproblems, one for each cell. Therefore one should focus on optimizing (which is done in this paper, with Tabu Search algorithm) the subproblem determining the total cycle time of the problem (bottleneck subproblem). This strategy is implemented in Algorithm 3.

Algorithm 3 CFSAP solving strategy

```
1: For each cell, create a TS algorithm instance for solving one-cell subproblem.
2: Run 1 iteration of TS algorithm solving the subproblem with the longest minimal
   cycle time $i = \arg\max_{i=1,2,\ldots,q} \{T(P_i, \pi_i)\}$.
3: If the time for calculations is not over yet, go to line 2.
```

The Tabu Search (TS) algorithm is a local search metaheuristic, proposed for the first time by Glover in [5]. Through years, the original idea has been modified repeatedly, creating multiple variants of the TS algorithm; applied to a wide range of scheduling problems (such as famous TSAB [10], neuro-tabu [2], or parallel TS [3]).

The algorithm used in the paper utilizes two types of a memory:

TL Tabu List, designed to avoid cycles and to leave local minimums. It consists of L last moves. Whenever the capacity is exceeded, the oldest move is removed from the list.

LTM Long-Term Memory, storing promising solutions (and associated TS states consisting of: current solutions, tabu lists and neighbourhoods) to provide diversification, each state can be used only once.

The algorithm (shown on Fig. 3) starts with generation of an initial solution. The solution is provided by a simple heuristic — operations are scheduled in an ascending order according to their number and assigned to a single machine. Then, a neighbourhood is generated (the neighbour is defined by the swap move, e.g. unordered pair of operations to be swapped in the permutation.) by the procedure described in Algorithm 4. Then, the

Algorithm 4 Neighnourhood creating procedure

```
1: Find a pair of consecutive operations with different machines assigned. Take the first
   operation.
2: If line 1. provided no operations, take first and last operation from the permutation.
3: Create neighbours, by swapping in the permutations operations from line 1. or 2.
   with $\alpha$ following and $\alpha$ preceding operations.
4: Remove duplicates (neighbours with the same pairs of operations to be swapped).
```

neighbours are evaluated by one of the three methods:

PA For all the neighbours, compute the exact value of the minimal cycle time, using the polynomial algorithm.

1-Opt For all the neighbours, estimate the value of the minimal cycle time, using the 1-Opt heuristic.

Hybrid perform the previously described **1-Opt** method and then compute the exact value of the minimal cycle time of the best β neighbours, using the polynomial algorithm.

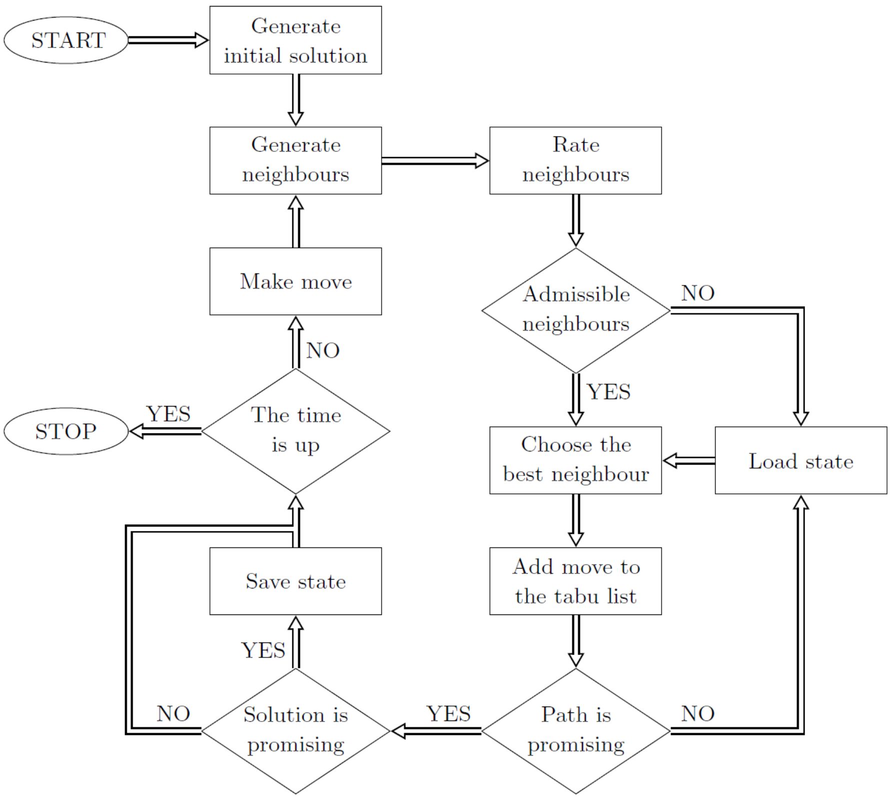

Figure 3: Schematic diagram of the Tabu Search algorithm.

The neighbours consisting of the moves from TL are removed, unless they improve the best known value of the minimal cycle time (aspiration criterion). If all the neighbours are removed, the last promising state is loaded from LTM. Then, the neighbour with the lowest value of the minimal cycle time is chosen and the associated move is added to TL. If the best solution found has not been improved since $1000 + 0.1 \cdot \textit{iteration number}$ iterations, the last promising state is loaded from LTM. If the neighbour fulfills at least one of the following conditions:

- there are less then 5 states memorized;
- there are less then 200 states memorized and $T \leqslant 1.1 \cdot T_{best}$;
- $T < T_{best}$;

the current state of TS is saved in LTM. If time for calculations is not over yet, the move is applied to the current solution, finishing an iteration.

6. Computational experiments

This section provides a description of an experimental evaluation of the proposed algorithms. First, the comparison of speed and quality of the results obtained in various scenarios by the CAP solving algorithms is presented. Then, effectiveness of the three Tabu Search variants on benchmark instances is tested.

The algorithms were implemented in C++ programming language, compiled with the default compiler of Microsoft Visual Studio 2015. The programs were executed on PC equipped with Intel Core i7-4930K CPU @3.4GHz, 32GB RAM and Windows 10 Education.

6.1. CAP solving algorithms

The two Cyclic Assignment Problem solving algorithms (namely 1-Opt and PS) were experimentally compared. Performance was measured on randomly generated instances of the following sizes: $o = 10, 20, 40, 80, 160, 320, 640, 1280, 2560$. For each size, 16 instances were generated (144 in total).

The algorithms were tested in three usage scenarios (as shown in Fig. 4), with different initial solutions for the experiments:

- experiments 1 and 2 – random solution;
- experiments 3 and 4 – optimal assignment, with a random swap move performed on the permutation;
- experiments 5 and 6 – random solution processed by 1-Opt algorithm, with a random swap move performed on the permutation.

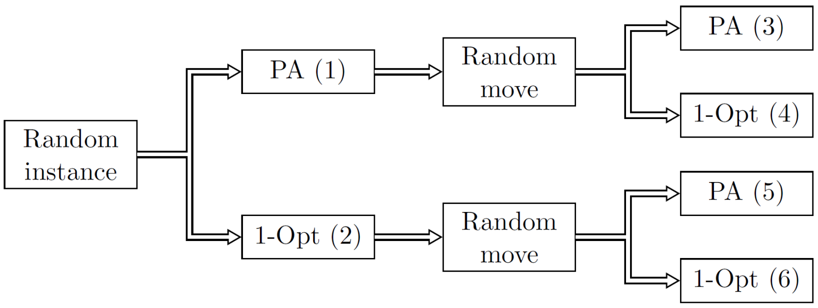

Figure 4: Setup for the experiments on CAP solving algorithms.

For each experiment and instance, computation times and the minimal cycle times T were measured. The 1-Opt algorithm is a heuristic, therefore a gap ΔT between the obtained cycle time T and the optimal cycle time T^* was also calculated

$$\Delta T = \frac{T - T^*}{T^*} \cdot 100\%.$$

The results of the experiments are presented in Fig. 5 and Tab. 1. The computation time of PA is almost unaffected by the initial solution and 2–4 times longer then 1-OPT starting from a random solution. As shown in the experiments 2, 4, 6; the quality of the results obtained by 1-OPT are dependent on the quality of the initial solution. With an initial solution close to the optimal, 1-OPT provides relatively good results up to about 1000 times faster then PA.

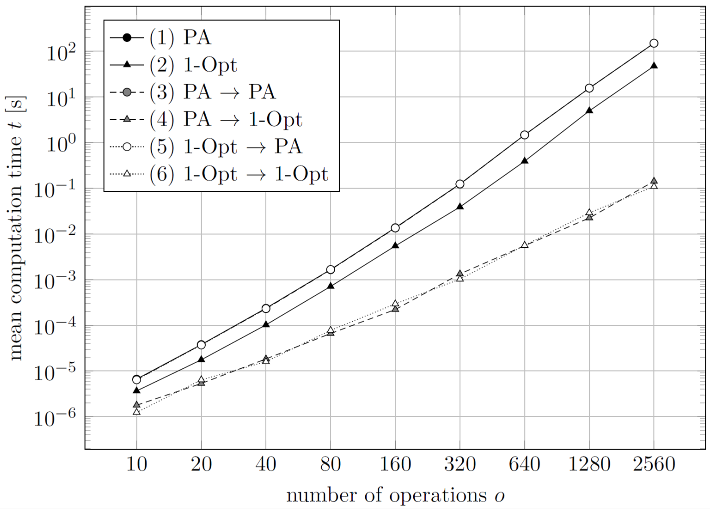

Figure 5: Computation times of the algorithms from experiments 1–6.

6.2. CFSAP solving algorithms

Since the problem has not yet been researched before, no available benchmarks existed. The test instances were therefore generated and published online [6]. More details on the data can be found in the report [7].In the paper, the first 120 instances were used (*gi0001–gi0120*, as shown in Tab. 4). The values of TS algorithm parameters were obtained experimentally (Tab. 2).

Three variants of the Tabu Search algorithm were tested with a time limit of 60 seconds for each instance. The minimal cycle time, mean neighbourhood size and a number of iterations were measured. The instances were divided according to their sizes into 12 groups. For each group, an average gap ΔT between obtained cycle time T and

Table 1: The results of the CAP solving algorithms tests. Columns correspond to experiments 1–6.

o	Mean computation time [s]						Mean ΔT [%]		
	1	2	3	4	5	6	2	4	6
10	$6.66E-6$	$3.66E-6$	$6.50E-6$	$1.78E-6$	$6.41E-6$	$1.24E-6$	2.48	$1.35E+0$	2.07
20	$3.79E-5$	$1.76E-5$	$3.77E-5$	$5.34E-6$	$3.70E-5$	$6.39E-6$	1.56	$2.18E-1$	1.08
40	$2.37E-4$	$1.02E-4$	$2.31E-4$	$1.84E-5$	$2.32E-4$	$1.60E-5$	1.71	$3.96E-1$	1.67
80	$1.66E-3$	$7.11E-4$	$1.64E-3$	$6.63E-5$	$1.67E-3$	$7.76E-5$	1.39	$9.68E-2$	1.23
160	$1.37E-2$	$5.47E-3$	$1.34E-2$	$2.21E-4$	$1.35E-2$	$2.95E-4$	1.47	$8.10E-2$	1.49
320	$1.24E-1$	$3.90E-2$	$1.23E-1$	$1.33E-3$	$1.24E-1$	$1.03E-3$	1.70	$5.43E-2$	1.68
640	$1.47E+0$	$3.91E-1$	$1.48E+0$	$5.53E-3$	$1.47E+0$	$5.63E-3$	1.54	$2.59E-2$	1.56
1280	$1.56E+1$	$4.94E+0$	$1.56E+1$	$2.24E-2$	$1.56E+1$	$2.91E-2$	1.45	$1.24E-2$	1.45
2560	$1.50E+2$	$4.72E+1$	$1.49E+2$	$1.42E-1$	$1.49E+2$	$1.09E-1$	1.45	$4.48E-3$	1.45

Table 2: The CFSAP solving algorithms parameters.

Algorithm	L	α	β
PA	50	15	-
1-OPT	100	10	-
Hybrid	75	15	1

Table 3: An average gap to the best obtained result, grouped by an instance size.

Group	ΔT [%]		
$n \times q$	PA	1-OPT	Hybrid
10×10	0	0	0
10×15	0	0	0
10×20	0.1393	0	0
20×10	0.1851	0.1044	0
20×15	0	0.0093	0
20×20	0.0206	0.1016	0
50×10	4.4533	0.4277	0.2084
50×15	5.4054	0.4060	0.2277
50×20	4.9372	0.6615	0.0787
100×10	21.2598	0.2158	0.1161
100×15	20.7294	0.4415	0.0994
100×20	20.3415	0.6333	0.2731
MEAN:	6.4560	0.2501	0.0836

the best cycle time across the three algorithms T_{min} was calculated

$$\Delta T = \frac{T - T_{min}}{T_{min}} \cdot 100\%.$$

The results of the experiments are summarized in Tables 3 and 4. The Hybrid TS algorithm provided better results for the majority of instances, followed by the 1-Opt. PA performance dropped significantly for the bigger instances, probably due to an insufficient number of TS iterations (for $n = 100$, an average of 36 iterations; compared to 597 for 1-OPT and 594 for Hybrid).

Table 4: The best results obtained for the instances *gi001–gi120*; 60 seconds of the computation time for each instance.

name	$n \times q$	T	name	$n \times q$	T	name	$n \times q$	T
gi001	10×10	608	*gi041*	20×15	992	*gi081*	50×20	2147
gi002	10×10	623	*gi042*	20×15	1097	*gi082*	50×20	2285
gi003	10×10	500	*gi043*	20×15	989	*gi083*	50×20	2270
gi004	10×10	585	*gi044*	20×15	1038	*gi084*	50×20	2264
gi005	10×10	549	*gi045*	20×15	998	*gi085*	50×20	2192
gi006	10×10	629	*gi046*	20×15	1077	*gi086*	50×20	2052
gi007	10×10	603	*gi047*	20×15	997	*gi087*	50×20	2130
gi008	10×10	506	*gi048*	20×15	921	*gi088*	50×20	2172
gi009	10×10	588	*gi049*	20×15	902	*gi089*	50×20	2530
gi010	10×10	493	*gi050*	20×15	1035	*gi090*	50×20	2498
gi011	10×15	597	*gi051*	20×20	914	*gi091*	100×10	4465
gi012	10×15	554	*gi052*	20×20	1019	*gi092*	100×10	4269
gi013	10×15	594	*gi053*	20×20	997	*gi093*	100×10	4201
gi014	10×15	505	*gi054*	20×20	928	*gi094*	100×10	4330
gi015	10×15	631	*gi055*	20×20	973	*gi095*	100×10	4158
gi016	10×15	626	*gi056*	20×20	1011	*gi096*	100×10	4355
gi017	10×15	585	*gi057*	20×20	943	*gi097*	100×10	4363
gi018	10×15	529	*gi058*	20×20	959	*gi098*	100×10	4268
gi019	10×15	649	*gi059*	20×20	1079	*gi099*	100×10	4143
gi020	10×15	561	*gi060*	20×20	940	*gi100*	100×10	4313
gi021	10×20	535	*gi061*	50×10	2176	*gi101*	100×15	4285
gi022	10×20	586	*gi062*	50×10	2100	*gi102*	100×15	4421
gi023	10×20	602	*gi063*	50×10	2147	*gi103*	100×15	4288
gi024	10×20	607	*gi064*	50×10	2289	*gi104*	100×15	4295
gi025	10×20	598	*gi065*	50×10	2171	*gi105*	100×15	4295
gi026	10×20	572	*gi066*	50×10	2185	*gi106*	100×15	4257
gi027	10×20	605	*gi067*	50×10	2152	*gi107*	100×15	4610
gi028	10×20	619	*gi068*	50×10	2321	*gi108*	100×15	4579
gi029	10×20	646	*gi069*	50×10	2173	*gi109*	100×15	4578
gi030	10×20	591	*gi070*	50×10	2242	*gi110*	100×15	4219
gi031	20×10	958	*gi071*	50×15	2222	*gi111*	100×20	4472
gi032	20×10	838	*gi072*	50×15	2403	*gi112*	100×20	4460
gi033	20×10	974	*gi073*	50×15	2305	*gi113*	100×20	4446
gi034	20×10	904	*gi074*	50×15	2279	*gi114*	100×20	4504
gi035	20×10	1002	*gi075*	50×15	2283	*gi115*	100×20	4417
gi036	20×10	998	*gi076*	50×15	2014	*gi116*	100×20	4503
gi037	20×10	988	*gi077*	50×15	2185	*gi117*	100×20	4442
gi038	20×10	872	*gi078*	50×15	2236	*gi118*	100×20	4411
gi039	20×10	1009	*gi079*	50×15	2223	*gi119*	100×20	4506
gi040	20×10	1000	*gi080*	50×15	2136	*gi120*	100×20	4556

7. Final remarks

Cyclic work of a two-machine production cell with setup times is considered in the paper. We proved, that the optimal operations to machines assignment (for a fixed order of operations), can be determined in the polynomial time $O(o^3)$, where o is the number of operations, despite the exponential number of all the possible assignments. In the further research we plan to extend our considerations onto cells with the number of machines greater than two.

References

[1] W. BOŻEJKO, M. UCHROŃSKI and M. WODECKI: Parallel metaheuristics for the cyclic flow shop scheduling problem. *Computers & Industrial Engineering*, **95** (2016), 156-163.

[2] W. BOŻEJKO, A. GNATOWSKI, T. NIŻYŃSKI and M. WODECKI: Tabu search algorithm with neural tabu mechanism for the cyclic job shop problem. In *Artificial Intelligence and Soft Computing: 15th International Conference, ICAISC 2016, Zakopane, Poland, June 12-16, 2016, Proceedings, Part II*, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds. Springer International Publishing, Cham, 2016, 409-418.

[3] W. BOŻEJKO, M. UCHROŃSKI and M. WODECKI: *Multi-GPU Tabu Search Metaheuristic for the Flexible Job Shop Scheduling Problem.* Springer International Publishing, Heidelberg, 2014, 43-60.

[4] W. BOŻEJKO and M. WODECKI: Problemy cykliczne z równoległymi maszynami (in Polish). In *Automatyzacja procesów dyskretnych, Teoria i zastosowania (red. A. Świerniak, J. Krystek)*, A. Świerniak and J. Krystek, Eds. Gliwice, 2014, 27-36.

[5] F. GLOVER: Future paths for integer programming and links to artificial intelligence. *Computer & Operational Research*, **13**(5), (1986), 533-549.

[6] A. GNATOWSKI: Benchmarks for nonpermutational flowshop with 2-machine nests and cycle time criterion. 2016. `http://wojciech.bozejko.staff.iiar.pwr.wroc.pl/Benchmarks/F_cycle_nonperm/F_cycle_2machines_benchmarks.zip`, Available online [2017-03-13].

[7] A. GNATOWSKI: Finding properties of an assignment problem in application to cyclic scheduling problems (in Polish). Tech. Rep. PRE No. 5, Faculty of Electronics, Wrocław University of Science and Technology, 2016.

[8] T. KAMPMEYER: *Cyclic Scheduling Problems*. PhD thesis, University Osnabrück, 2006.

[9] E. LEVNER, V. KATS, D.A.L. DE PABLO and T. CHENG: Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers & Industrial Engineering*, **59**(2), (2010), 352-361.

[10] E. NOWICKI and C. SMUTNICKI: A fast taboo search algorithm for the job shop problem. *Management science*, **42**(6), (1996), 797-813.

[11] T. SAWIK: A mixed integer programming for cyclic scheduling of flexible flow lines. *Bulletin of the Polish Academy of Sciences, Technical Sciences*, **62**(1), (2014), 121-128.