BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 66, No. 3, 2018
DOI: 10.24425/123440

Minimal cycle time determination and golf neighborhood generation for the cyclic flexible job shop problem

W. BOŻEJKO[1]*, J. PEMPERA[1], and M. WODECKI[2]

[1]Department of Automatics, Mechatronics and Control Systems Faculty of Electronics, Wrocław University of Science and Technology
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

[2]Telecommunications and Teleinformatics Department, Faculty of Electronics Wrocław University of Science and Technology,
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract. In the paper, a problem of scheduling operations in the cyclic flexible job shop system is considered. A new, very fast method of determining the cycle time for any order of tasks on machines is also presented. It is based on the analysis of the paths in the graph representing the examined problem. The theorems concerning specific properties of the graph are proven and used in the construction of the heuristic algorithm searching the solutions space by using the so-called golf neighborhood, which is generated in a way similar to the game of golf, which helps to intensify and diversify calculations. The conducted computational experiments fully confirmed the effectiveness of the proposed method. The proposed methods and properties can be adapted and used in the construction of local search algorithms for solving many other optimization problems.

Key words: cyclic scheduling, metaheuristic, discrete optimization.

List of main symbols

- a^l – l-th copy of operation a in l-th MTS
- C_a – operation a completion time in 1-th MTS
- $G^{\oplus}(\pi)$ – graph for solution π of cyclic job shop
- $H(\pi)$ – graph for solution π of job shop problem
- $\mathscr{J}$ – set of tasks
- $\mathscr{M}$ – set of machines
- MTS – minimal task set
- $\mathscr{O}$ – set of operations
- π – solution (m-tuple of permutations)
- π_i – permutation of operations on i-th machine
- S_a – operation a starting time in 1-th MTS
- S_a^l – operation a starting time in l-the MTS
- $T(\pi)$ – cycle time of solution π
- $T^{\circ}(\pi)$ – minimal cycle time of solution π
- T^* – optimal cycle time

1. Introduction

Flexible manufacturing systems are currently the object of very intensive research in many scientific centers. This process is caused by the fact that many companies adopted the strategy of manufacturing on demand, where production is conditioned by current orders thereby reducing the costs of storage of raw materials and finished products. Moreover, technological development and in particular the machines configured and controlled by computers enable a short-term and multi-assortment production. Scheduling of operations in the flexible job shop system requires taking a decision simultaneously on two levels: (i) the allocation of operations to machines, (ii) determination of the order of operations on each machine. Compared to conventional scheduling problems, it is a meaningful generalization and significantly hinders the design of efficient algorithms. The vast majority of works devoted to the flexible job shop problem concerns the minimization of completion of all executed operations. Due to the NP-hardness of the problem, the attention of scientists was focused on the construction of heuristic algorithms, or exact approaches of a small size (e.g. mixed integer programming, Sawik [20], branch and bound method with using max-plus algebra, Houssin [12]). These are mainly algorithms based on tabu search method (Hurink, Jurish and Thole [13], Mastrolilli and Gambardella [17], Bożejko et al. [3, 5]) or simulated annealing (Bożejko et al. [6]). On the other hand, a genetic algorithm was used by Yang, Kacem and Borne [15]. The most effective are hybrid algorithms. Xia and Wu [21] proposed a particle swarm algorithm using an additional simulated annealing, whereas Jie, Linyan and Mitsuo [14] – a genetic algorithm combined with tabu search algorithm with a variable

*e-mail: wojciech.bozejko@pwr.edu.pl

Manuscript submitted 2017-08-29, revised 2017-11-13, initially accepted for publication 2017-11-15, published in June 2018.

neighborhood. In turn, Bożejko et al. [4] presented parallel population-based metaheuristics.

In the cyclic production system, the basic set of tasks is executed repeatedly at fixed intervals (cycle time). This allows a considerable simplification of the logistical operations related to the supply of raw materials and receiving of products, because these activities are carried out at regular intervals. The main problem with which we are dealing in constructing algorithms for such problems is the lack of effective methods of determining cycle time and good lower or upper bounds. General computational models for cyclic scheduling problems are presented in the work of Kampmayer [16]. In conclusion, the author stated that the use of universal packages of discrete optimization allows the solution in a reasonable time only in case of instances of small size. Local search algorithms for cyclic problems were used for the first time by Brucker and Kampmayer [8]. A cyclic job shop problem with additional no storage constraint is considered in this paper. An algorithm based on tabu search method implementation is used to solve that problem. Current research concerning solutions to the problems of production scheduling focus on new searching methods of the solution space, mainly inspired by the processes occurring in nature (evolutionary, ant colony, gregarious, swarm search methods, etc.). In this paper we propose a new method of constructing of local search algorithms which has a similarity to shots in the golf game, from which we have taken the name of the neighborhood.

2. Flexible job shop problem

In this section there is a brief presentation of the flexible job shop problem, whereas in the next, the cyclic version of this problem is described, which is the essential theme of the work. There is a set of tasks $\mathscr{J} = \{1, 2, \ldots, n\}$ given, to be executed on machines from the set $\mathscr{M} = \{1, 2, \ldots, m\}$. A task is a sequence of certain operations occurring in the technological order. For each operation there is a subset of machines defined called a nest. One operation must be performed on one machine of this subset. Due to the different machine performance, execution time of the operation depends on the assigned machine. Problem (briefly denoted by FJS) relies on assignment of the operation to the machines and setting the order of operations on the machines to optimize a certain criterion. Hereby the following constraints must be met:

a) each operation is performed by only one, selected from a subset, machine,
b) operation execution cannot be interrupted before its completion,
c) the machine cannot perform more than one operation at the same time,
d) technological order of executing of operations must be preserved.

Let $\mathscr{O} = \{1, 2, \ldots, o\}$ be the set of all operations. The set can be partitioned into sequences corresponding to the tasks, where the task $j \in \mathscr{J}$ is a sequence o_j of operations to be successively executed on the respective machines (i.e. in the technological order). These operations are indexed by numbers $(l_{j-1} + 1, \ldots, l_{j-1} + o_j)$, where $l_j = \sum_{i=1}^{j} o_i$ is the number of the operations of the first j tasks, $j = 1, 2, \ldots, n$, wherein $l_0 = 0$, $o = \sum_{i=1}^{n} o_i$. Next, let $\mathscr{M}^i \subset \mathscr{M}\,(i \in \mathscr{O})$ be the set of machines, on which operation i is to be executed and $p_{i,k}\ (k \in \mathscr{M}^i)$ is the execution time of i operation on machine k. By $\mu = (\mu_1, \ldots, \mu_o)$ we can denote the assignment of operations to machines, where $\mu_a \in \mathscr{M}^a$ is the machine assigned to execute an operation $a \in \mathscr{O}$. The set

$$\mathscr{O}^l = \{a \in \mathscr{O} : \mu_a = l\} \tag{1}$$

includes operations executed on machine $l \in \mathscr{M}$, whereas $\bigcup_{i=1}^{m} \mathscr{O}^i = \mathscr{O}$.

Let permutation π_l be a certain sequence of executing operations from the set $\mathscr{O}^l$ on machine l $(|\mathscr{O}^l| = n_l)$ and Π^l be the set of all permuations of elements from $\mathscr{O}^l$. The sequence of operations on the machines is determined by the concatenation of m permutations $\pi = (\pi_1, \pi_2, \ldots, \pi_m) \in \Pi$, where $\Pi = \Pi^1 \times \Pi^2 \times, \ldots, \times \Pi^m$. Let us note that a m-tuple $\pi \in \Pi$ unambiguously defines the assignment of operations to machines and the order of operations execution on individual machines.

For a fixed order of operations execution on machines $\pi \in \Pi$, the schedule for their execution may be represented by starting moments S_i and completion times C_i of execution of operations $i = 1, 2, \ldots, o$. If we assume that $C_i = S_i + p_i$, then we can limit our scope to the starting moments of operation S_i, $i = 1, 2, \ldots, o$. Hereby the following constraints must be fulfilled:

$$\begin{gathered} S_i + p_i \le S_{i+1}, \\ i = l_{j-1} + 1, \ldots, l_{j-1} + o_j - 1,\ j = 1, \ldots, n, \end{gathered} \tag{2}$$

$$\begin{gathered} S_{\pi_l(j)} + p_{\pi_l(j)} \le S_{\pi_l(j+1)}, \\ l = 1, \ldots, m,\ j = 1, \ldots, n_l - 1, \end{gathered} \tag{3}$$

$$S_{\pi(i)} \ge 0 \quad i = 1, \ldots, o. \tag{4}$$

Inequality (2) corresponds to the constraints (a) and (d), whereas (2) to the constraints (c) and (b). Without loss of generality, we can assume that the starting moment of execution of the first operation on the first machine is $S_{1,\pi(1)} = 0$.

The sequence of operations execution $\pi \in \Pi$ is feasible for FJS problem, if there are starting moments of operations execution (schedule) $S_{\pi(i)}$, $i = 1, 2, \ldots, o$, satisfying the constraints (2–4). The set of these feasible solutions will be denoted by Φ $(\Phi \subseteq \Pi)$.

2.1. Graph model of the flexible job shop problem. Any feasible solution $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, $\pi \in \Phi$, can be represented by a directed graph $H(\pi) = (\mathscr{V}, \mathscr{E}(\pi))$ with weighted vertices and arcs. A set of vertices $\mathscr{V} = \mathscr{O}$, vertices correspond to operations. The weight of a vertex $v = \mathscr{O}$ is equal to p_v – time of

execution of the operation v on the machine $\mu(v)$. On the other hand, a set of arcs $\mathscr{E} = \mathscr{R} \cup \mathscr{K}(\pi)$, where:

1) $\mathscr{R} = \bigcup_{j=1}^{n} \bigcup_{i=1}^{o_j-1} \{(l_{j-1}+i, l_{j-1}+i+1)\}$.

The arcs combine subsequent operations of the same task. They are called technological arcs (as in fact they represent the technological order).

2) $\mathscr{K}(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{|\mathscr{O}^k|-1} \{(\pi_k(i), \pi_k(i+1))\}$.

Arcs from this set join the operations performed on the same machine (ordered arcs). They represent the order π_k of operations execution from the set $\mathscr{O}^k$ on k-th machine ($k = 1, 2, \ldots, m$).

The weight of any arc of the graph equals zero. Since there is a mutual equivalence between operations and the vertices of the graph, to simplify the notation, the operation will be identified with its corresponding vertex.

Property 1. A solution $\pi \in \Phi$ is feasible for the FJS problem if and only if the graph $H(\pi)$ does not contain cycles.

3. Cyclic flexible job shop problem

In the cyclic production system, a fixed set of tasks called MTS (*minimal task set*, see minimal part set in Brucker and Kampmeyer [9]) is performed repeatedly in the production cycles. MTSs are carried out directly one after another in a cyclic manner. We assume that in each of the MTSs on each machine operations are performed in the same order. Therefore, in each cyclic schedule, the order of operations may be represented by constructing a m-tuple of permutations of operations on individual machines in the first MTS. The cyclic nature of the process is subject to the following constraints:

e) each operation is performed sequentially (in consecutive MTSs) after the cycle time completion.

It was assumed that in each MTS the operations executed on machines are performed in the same order. For a given solution (m-tuple) $\pi \in \Phi$ (Φ – set of FJS problem feasible solutions), let $\mathscr{S}^k = (S_1^k, S_2^k, \ldots, S_o^k)$ be a sequence of starting moments for execution of operations in the k-th MTS, where S_i^k denotes the moment of execution of an operation i on machine μ_i in k-th cycle (MTS). We assumed that the time schedule (i.e. operations execution in subsequent MTS) is cyclic. This means that there is a fixed $T(\pi)$ (period) such that

$$S_{\pi(i)}^{k+1} = S_{\pi(i)}^{k} + T(\pi), \quad i = 1, \ldots, o, \quad k = 1, 2, \ldots \tag{5}$$

The above presented equality is the realization of the constraint (e).

The size of $T(\pi)$ depends obviously on the solution π and is called cycle time. The minimum value of $T(\pi)$, for a fixed order of operations on machines π will be called *minimum cycle time* and denoted by $T^{\circ}(\pi)$. Because the order of operations execution for each MTS is the same, therefore it is enough to designate the starting moments of execution of operations $S_1, S_2, \ldots, S_o$ for the first MTS and make the shift by the size of $T(\pi)$. Therefore,

$$S_i^k = S_i + (k-1) \cdot T(\pi) \tag{6}$$

is the starting time of an operation $i \in \mathscr{O}$ in k-th cycle (i.e. in k-th MTS), $k = 1, 2, \ldots$

Minimum cycle time $T^{\circ}(\pi)$, for a fixed order of execution of tasks π, can be determined by solving the following linear programming task: determine

$$T^{\circ}(\pi) = \min\{T\}, \tag{7}$$

s.t.:

$$\begin{gathered} S_i^k + p_i \le S_{\pi(i+1)}^k, \\ i = l_{j-1}+1, \ldots, l_{j-1}+o_j-1 \\ j = 1, \ldots, n, \ k = 1, 2, \ldots, \end{gathered} \tag{8}$$

$$\begin{gathered} S_{\pi_l(j)}^k + p_{\pi_l(j)} \le S_{\pi_l(j+1)}^k, \\ l = 1, \ldots, m, \ j = 1, \ldots, n_l - 1, \ k = 1, 2, \ldots, \end{gathered} \tag{9}$$

$$\begin{gathered} S_{\pi_l(n_l)}^k + p_{\pi_l(n_l)} \le S_{\pi_l(1)}^k + T, \\ l = 1, \ldots, m, \ k = 1, 2, \ldots, \end{gathered} \tag{10}$$

$$S_{\pi(i)}^k \ge 0 \quad i = 1, \ldots, o, \ k = 1, 2, \ldots \tag{11}$$

In this paper we consider the problem of designation of the optimal cycle time T^*, which comes to determine such a solution π^*, for which

$$T^* = T^{\circ}(\pi^*) = min\{T^{\circ}(\pi) : \ \pi \in \Phi\}.$$

In short this problem will be denoted by CFJS (Cyclic Flexible Job Shop).

4. Determination of minimum cycle time

In this chapter, for a given order of operations execution on the machines (an element of the set Φ), we present the new method of determination the minimum cycle time in the flexible job shop problem. This method is based on a graph representing the first $(m+1)$ MTSs. In the following part, for simplification of the notation it was assumed that $\eta = m + 1$.

4.1. Cyclic graph. Let $\pi \in \Phi$ be a feasible solution, and $H^1 = (\mathscr{V}^1, \mathscr{E}^1)$ the first component, i.e., a graph representing

the order of operations execution on machines for the first MTS (description of the graph is given in Section 2.1).

By $H^l(\pi) = (\mathscr{V}^l, \mathscr{E}^l)$ $(l = 2, 3, \ldots, \eta)$ we denote a graph representing the order of operations execution for l-th MTS. The set of vertices of this graph

$$\mathscr{V}^l = \{v + (l-1)\cdot o \colon v \in \mathscr{V}^1\}. \tag{12}$$

A pair of vertices from the set $\mathscr{V}^l$ is an arc

$$(u-v) \in \mathscr{E}^1 \Longleftrightarrow \big(u + (l-1)\cdot o,\, v + (l-1)\cdot o\big) \in \mathscr{E}^1. \tag{13}$$

Graph $H^l(\pi)$ will be called l-th component. Undeniably, $H^1(\pi)$ is isomorphic with each graph $H^l(\pi)$, $i = 2, 3, \ldots \eta$.

The set of vertices of graph $H^l(\pi)$

$$\begin{aligned}\mathscr{A}^l = \{&v \in \mathscr{V}^1 \colon v = \pi_j(1) + (l-1)\cdot o,\\ &j = 1, 2, \ldots, m\},\end{aligned} \tag{14}$$

comprises the first operations and the set

$$\begin{aligned}\mathscr{B}^l = \{&u \in \mathscr{V}^1 \colon u = \pi_j(n_j) + (l-1)\cdot o,\\ &j = 1, 2, \ldots, m\},\end{aligned} \tag{15}$$

the last operations of tasks performed by the individual machines in the l-th MTS. Undoubtedly, $|\mathscr{A}^l| = |\mathscr{B}^l| = m$, $l = 1, \ldots, \eta$.

For a fixed permutation $\pi \in \Phi$ we consider the η first MTSs. We assign them a graph $G^{\oplus}(\pi) = (\mathscr{V}^{\oplus}, \mathscr{E}^{\oplus}(\pi))$, called cyclic graph, which is the sum of η first consecutive components, namely

$$G^{\oplus}(\pi) = H^1(\pi) \oplus H^2(\pi) \oplus, \ldots, \oplus H^{\eta}(\pi), \tag{16}$$

whereby the set of vertices

$$\mathscr{V}^{\oplus} = \mathscr{V}^1 \cup \mathscr{V}^2 \cup, \ldots, \cup \mathscr{V}^{\eta},$$

and set of arcs

$$\mathscr{E}^{\oplus} = \mathscr{E}^1 \cup \mathscr{E}^2 \cup, \ldots, \cup \mathscr{E}^{\eta} \cup \mathscr{W},$$

where $\mathscr{W}$ is a set of arcs between successive components. They combine the last operation performed on the machine in a certain component with the first operation carried out on the same machine in the next component, namely

$$\mathscr{W} = \{(u, v) \colon u \in \mathscr{B}^i, v \in \mathscr{A}^{i+1}, \mu_u = \mu_v, i = 1, 2, \ldots, m\}.$$

The number of vertices of the graph $|\mathscr{V}| = o\cdot\eta$, and arcs

$$|\mathscr{E}^{\oplus}| = \eta\cdot\sum_{j=1}^{n}(o_j - 1) + \eta\cdot\sum_{l=1}^{m}\big(|\mathscr{O}^l| - 1\big) + n\cdot m,$$

where $\sum_{j=1}^{n}(o_j - 1)$ and $\sum_{l=1}^{m}\big(|\mathscr{O}^l| - 1\big)$ are respectively the number of arcs (i.e. designating the sequence of operations within a task) and arcs designating the order of execution of operations by the machines in the first component, while $n\cdot m$ is the number of arcs in the set $\mathscr{W}$.

A sequence of vertices $P(v_1, v_k) = (v_1, v_2, \ldots, v_k)$ in the graph $G^{\oplus}$ such that $(v_i, v_{i+1}) \in \mathscr{E}^{\oplus}$ for $i = 1, 2, \ldots, k-1$, is called a way (or a path) from vertex v_1 dot v_k, a $L(v_1, V_k) = \sum_{i=1}^{k-1} p_{v_i}$ its length, where p_{v_i} is the weight v_i in the graph $G^{\oplus}$. It is worth noting, that the path length includes the first vertex weight v_1 but it does not include the weight of the last vertex v_k.

We consider the vertex $a \in \mathscr{A}^1$. It corresponds to the first operation of a certain task in the first MTS. From the definition of graph $G^{\oplus}(\pi)$, the vertex $a^l = a + (l-1)\cdot o$ corresponds to the same operation, but in the l-th component. By $L^l(a, a^l)$ we denote the length of the longest path in the graph $G^{\oplus}(\pi)$ from vertex a to the vertex a^l, $l = 2, 3, \ldots, \eta$.

The matrix $\boldsymbol{L}$ is defined with n columns and m rows (for simplicity indexed by numbers 2, 3, …, η), whose element

$$\lambda_{a,l} = L^l(a, a^l)/(l-1),\ a \in \mathscr{A}^1,\ l = 2, 3, \ldots, \eta. \tag{17}$$

Then, let

$$\Lambda^* = \max_{v \in \mathscr{A}^1}\ \max_{2 \le l \le \eta}\ \{\lambda_{v,l}\}, \tag{18}$$

be the maximum element of the matrix $\boldsymbol{L}$. $\Lambda^* = L^k(a, a^k)/(k-1)$, then the path $P^k(a, a^k)$ with length $L^k(a, a^k)$ is called a critical path in a cyclic graph $G^{\oplus}(\pi)$. Below two theorems on the relationship between the value of Λ^* and the minimum cycle time $T^{\circ}(\pi)$ will be proven.

Theorem 1. If $\pi \in \Phi$ is a feasible sequence of operations, then the minimum cycle time is $T^{\circ}(\pi) \ge \Lambda^*$.

Proof. Let us assume indirectly that for a certain feasible solution $\pi \in \Phi$, $T^{\circ}(\pi) \ge \Lambda^*$, where Λ^* the value determined in (18) and $T^{\circ}(\pi)$ – minimum cycle time.

For definiteness, it is assumed that the maximum in (18) was achieved for some vertex $a \in \mathscr{A}^1$ and k-th component of the graph $G^{\oplus}(\pi)$, i.e. the maximum element of the matrix $\boldsymbol{L}$,

$$\Lambda^* = \lambda_{a,k} = L^k(a, a^k)/(k-1), \tag{19}$$

where $L^k(a, a^k)$ is the length of the longest path in the graph $G^{\oplus}(\pi)$ from vertex a to vertex a^k.

It follows from the definition of the cycle time that if S_a $(a \in \mathscr{O})$ is the starting moment of execution of operations a in the first MTS, then its starting moment in i-th MTS (i.e. operation a^i) is

$$S_{a^i} = S_a + (i-1)\cdot T^{\circ}(\pi),\ i = 2, 3, \ldots, \eta. \tag{20}$$

In particular (for $i = k$), from the starting moment of execution of a operation until the starting of operation a^k there must elapse exactly

$$S_{a^k} - S_a = (k-1)\cdot T^{\circ}(\pi) \tag{21}$$

time units (see Fig. 1).

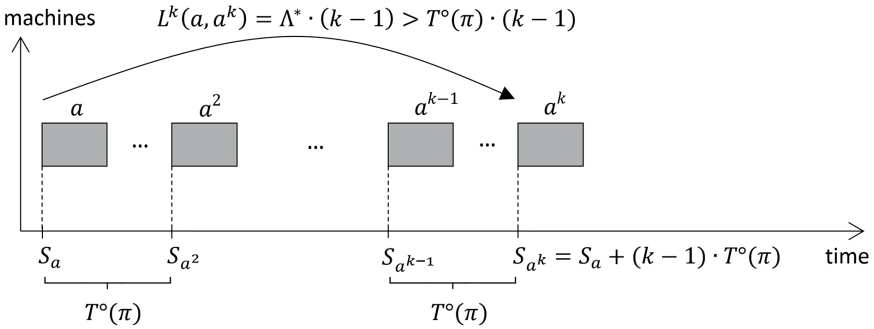

Fig. 1. The starting moments of execution of a operations in consecutive MTSs

It follows from definitions (17) and (18) that in the graph $G^{\oplus}(\pi)$ there is a path from vertex a to vertex a^k of length $L^k(a, a^k) = \Lambda^* \cdot (k-1)$ (Fig. 1). Using the indirect assumption $\Lambda^* > T^\circ(\pi)$, we obtain the inequality

$$L^k(a, a^k) = \Lambda^* \cdot (k-1) > T^\circ(\pi) \cdot (k-1). \tag{22}$$

Therefore, (from (21) and (22)) in the graph $G^{\oplus}(\pi)$ there is a path from vertex a to vertex a^k of length

$$L^k(a, a^k) > T^\circ(\pi) \cdot (k-1) = S_{a^k} - S_a. \tag{23}$$

It follows directly from the construction of graph $G^{\oplus}(\pi)$ and constraints (8–10) that before starting the operation a in k-th MTS (vertex a^k) all operations lying on the road between the vertices a and a^k must be executed. It follows from inequality (23) that the time that must elapse between the starting moment of operations a in the first and k-th MTS (i.e. operation a^k) must be greater than $S_{a^k} - S_a$, which leads to a contradiction with equality (21), thus with the indirect assumption $(\Lambda^* > T^\circ(\pi))$ and completes the proof of the theorem. □

Theorem 2. If $\pi \in \Phi$ is a feasible solution to CFJS problem, then the minimum cycle time $T^\circ(\pi) \leq \Lambda^*$.

Proof. We will show that the value Λ^* defined in (18) is a certain cycle time for permutation $\pi \in \Phi$.

Let $T^\circ(\pi)$ be the minimum cycle time for permutation, $\pi \in \Phi$. Thus, there is a sequence of starting moments of operation execution in the first MTS. Therefore, the starting moments of operation execution in the consecutive MTSs is determined by shifting S_i, $i \in \mathcal{O}$ by multiplication of the cycle time $T^\circ(\pi)$.

For any operation $a \in \mathcal{O}$ executed in the first MTS, the starting moments of operation execution in the consecutive MTS are defined in the following manner:

$$\begin{gathered} S_a^j = S_a + (j-1) \cdot \Lambda^*, \\ j = 2, 3, \ldots, \eta, \;\; a = 1, 2, \ldots, o. \end{gathered} \tag{24}$$

We will prove that the above defined execution starting times and the value $T = \Lambda^*$ meet the constraints (8–11). Therefore, Λ^* is a certain (feasible) cycle time, thus ultimately $T^\circ(\pi) \leq \Lambda^*$.

Let us assume indirectly, that operation a^l in l-th MTS is the first (of the smallest number in a graph $G^{\oplus}$), whose starting moment designated on the basis of (24) does not meet the constraint (8) or (9) (it is easy to note with the use of (24), that in this case the condition (10) is fulfilled). If the operation a is the equivalent of a^l in the first MTS, i.e. $a^l = a + (l-1) \cdot o$, then on the basis of (24) the starting moment of execution of this operation (see Fig. 2)

$$S_{a^l}^l = S_a^1 + (l-1) \cdot \Lambda^*. \tag{25}$$

We assumed implicitly that the moment does not comply with the constraints (8) or (9) which means that in the moment $S_{a^l}^l$:

(i) the machine, on which the operation a^l is to be performed, is not free (i.e. as it is still executing the preceding a^l operation), or

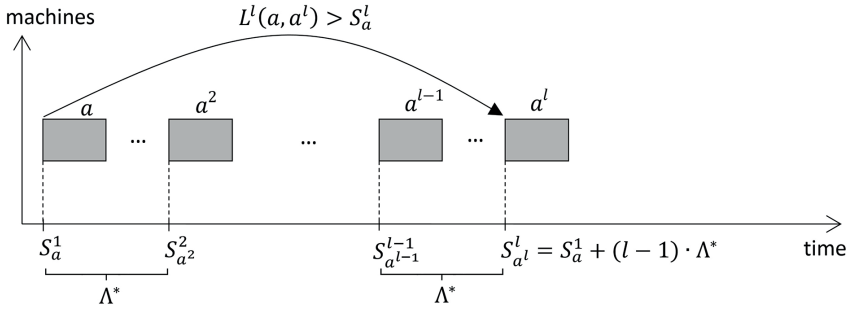

Fig. 2. The maximum path and the starting moments of operation execution

(ii) operation preceding a^l, in the technological line, has not been completed.

We are considering two cases (including (i) and (ii) at the same time).

Case 1. The operation a and a^l are the first operations executed on the machine respectively in the first and l-th MTS. Let $L^l(a, a^l)$ be the length of the longest path in a graph $G^{\oplus}$ from vertex a to a^l. It follows from the definition of the longest path that $S_a + (l-1)\cdot L^l(a, a')/(l-1)$ is the earliest moment, in which it is possible to start executing of operation a^l (all operation preceding a^l have been completed). Using the definition (18) it can be assumed, that the maximum was reached for operation $b \in \mathcal{O}$ and k-th MTS, i.e. $\Lambda^* = L^k(b, b^k)/(k-1)$. Therefore, the moment (25) of starting the execution of operation a^l is

$$S^l_{a^l} = S_a + (l-1)\cdot\Lambda^* = S_a + (l-1)\cdot L^k(b, b^k)/(k-1) \geq$$
$$\geq S_a + (l-1)\cdot L^l(a, a^l)/(l-1) = S_a + L^l(a, a^l).$$

The last inequality follows from the fact that $\Lambda^* = L^k(b, b^k)/(k-1)$ as the maximum element in the matrix L fulfils the inequality $L^k(b, b^k)/(k-1) \geq L^l(a, a^l)/(l-1)$.

We have proved that $S^l_{a^l} \geq S_a + L^l(a, a^l)$, therefore execution of operation a^l can be started in the moment $S^l_{a^l}$ designated from (25), as all the predecessors of a^l (including the ones on the critical path) have already been executed. The above description stays in contradiction with the adopted implicit assumption that the task a^l, cannot be started in the moment $S^l_{a^l}$ since there are cases (i) or (ii).

Case 2. Operations a and a^l are not the first operations executed on machines in the first and l-th MTS respectively. Let a_1 be the first operation executed on machine on which operation a is executed and a^l_1 its equivalent in l-th MTS (undoubtedly $a^l > a^l_1$).

The difference between the starting moments of operations a_1 and a in the first MTS is the same as the operation a^l_1 and a^l in l-th MTS (denoted by x in Fig. 3).

In addition, the difference between the starting moments of operations a and a^l is the same as between a_1 and a^l_1 (y in Fig. 3). Thus, if the commencement of operations in the first MTS meets all the feasible constraints, then also the commencement moments of these operations in the l-th MTS meet these constraints. In this way we have proved the theorem. □

It follows from Theorems 1 and 2 that for a fixed operations order $\pi \in \Phi$ (i.e. the sequence of operations execution on machines in the first MTS) the value $\Lambda^*(\pi)$ (designated in (18)) is the minimum cycle time, i.e. $T^{\circ}(\pi) = \Lambda^*(\pi)$. Therefore, to reduce the cycle time, that is $T^{\circ}(\pi)$, there must be a solution $\beta \in \Phi$, generated from π, for which the value $\Lambda^*(\beta)$ will be smaller than $\Lambda^*(\pi)$. So, if $\Lambda^*(\pi) = L^k(a, a^k)/(k-1)$, then the necessary condition (reducing the value of $\Lambda^*(\pi)$) is shortening the length of the critical path $L^k(a, a^k)$ in the graph $G^{\oplus}(\pi)$. Generating β will consist of changing the order of performing certain operations on the machine or transferring the operations to another machine. In the further part of this section we present the theorem from which it results the fact that, changing the order of certain operations on the machine does not reduce the length of the critical path, and thus the value of $\Lambda^*(\pi)$.

Let $P^k(a, a^k)$ $(a \in \mathcal{A}^1, a^k = a + (k-1)\cdot o, 2 \leq k \leq \eta)$ be the critical path in the graph $G^{\oplus}(\pi)$. The maximum subsequence of occurring directly one after another operations on a critical path executed on the same machine will be called a **block**. In the block one can distinguish the first and the last operation, whereas all the other operations will be called internal ones. Below there is a theorem which is an extension to the cyclic problem the so called 'block elimination properties'. They are successfully used in many algorithms for solving scheduling problems (e.g. Nowicki and Smutnicki [18], Wodecki [19]).

Theorem 3. If permutation β was generated from $\pi \in \Phi$ by changing the order of certain internal operations of the block, then the length of the cycle time $T^{\circ}(\beta) \geq T^{\circ}(\pi)$.

Proof. The theorem is proved just in the same way as in case of the block theorems for a wide class of scheduling problems with the criterion $C_{\max}$, in particular for flexible job shop problem [4]. □

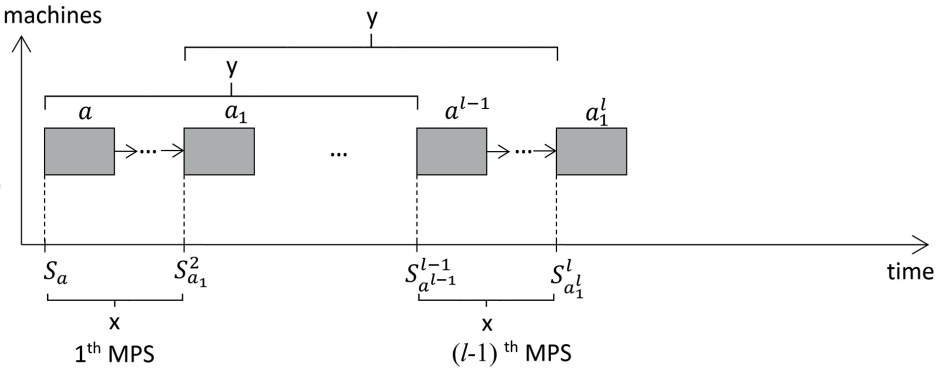

Fig. 3. The beginning moments of operations in the first and l-th MTS (arcs represent a technological line)

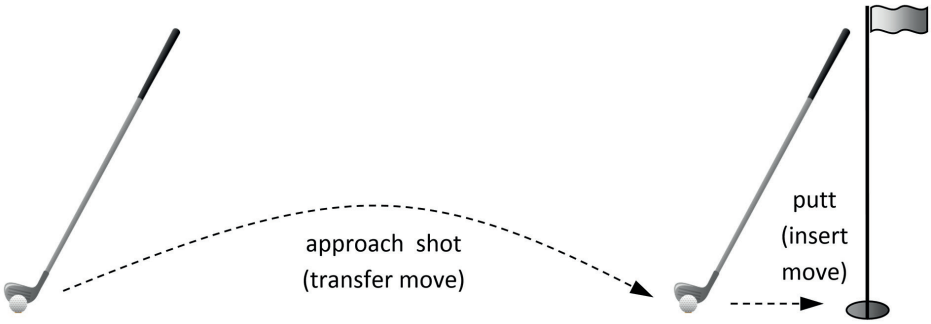

Fig. 4. Strokes in the game of golf

5. Golf algorithm

In this part we shortly present a new heuristic method of searching the space of feasible solutions. Its basic element is the neighborhood, i.e. the set of solutions generated by a simple modification (moves) in the current solution. The idea of determining the neigborhood is based on the game of golf. This game, in large simplification, relies in carring out two types of strokes (see Fig. 4):

1. strong, transferring the ball to the further area of the golf course, and
2. weak, whose aim is a direct transfer of the ball into the hole.

The game takes place in several rounds. The player uses some knowledge (e.g. gets to know the topography of the playing area) what enables him to improve the efficiency of the game). Based on this idea we introduce two types of moves transforming the elements of the solution space

$$\Gamma, \gamma\colon\ \Phi \to \Phi. \tag{26}$$

The first of the moves Γ, corresponding to the strong stroke in a game of golf, generates from $\pi \in \Phi$ the solution $\Gamma(\pi) \in \Phi$ significantly different from it, causing diversification of exploration. In case of move of γ, type the solution $\gamma(\pi) \in \Phi$ is only little different from π (thus it corresponds to the weak stroke) and causes intensification of exploration. By applying these moves it is possible to determine, depending on the needs, neighborhoods which vary greatly.

We are considering the problem of minimizing the F function on the set Φ. Let $S^{\triangleright\triangleright}$ and $S^{\triangleright}$ be respectively the set of strong and weak moves. If the transformation $\tau(\pi) = \gamma\big(\Gamma(\pi)\big)$, where $\Gamma \in S^{\triangleright\triangleright}$, $\gamma \in S^{\triangleright}$, $\pi \in \Phi$, we can say that τ is a complex move and we can put it down as $\tau = \gamma \circ \Gamma$. Then the set

$$\begin{aligned}\mathcal{N}(\pi) = \big\{\tau(\pi)\colon \tau = \gamma \circ \Gamma,\ \Gamma \in S^{\triangleright\triangleright},\\ \gamma \in S^{\triangleright},\ \pi \in \Phi\big\}.\end{aligned} \tag{27}$$

is golf neighborhood of the element $\pi \in \Phi$ (see also [2]). While generating it we will use the 3, theorem, i.e. we can omit the moves changing the order of tasks within the block. The neighborhood will be used in the algorithm based on the local search method.

In the description of the algorithm, the search history memory *MEM* is limited in length and supported in the principle of the queue of *FIFO*. The $ATR(\pi)$ function returns attributes of the solutions π. The algorithm terminates after execution of *Maxiter* iterations.

Golf algorithm (AGF)

Let $\pi \in \Phi$ be any starting solution;
$\pi_{best} \leftarrow \pi$; $MEM \leftarrow 0$; $iter \leftarrow 0$;
repeat
  Step 1: Generate golf neighborhood $\mathcal{N}(\pi)$
    of the solution π omitting the elements,
    whose attributes are on the *MEM* list;
  Step 2: Determine the element $\beta^* \in \mathcal{N}(\pi)$ Such that
    $F(\beta^*) = \min\{F(\delta)\colon\ \delta \in \mathcal{N}(\pi)\}$;
    if $F(\beta^*) < F(\pi_{best})$, **then** $\pi_{best} \leftarrow \beta^*$;
  Step 3: Substitute $MEM \leftarrow MEM \cup ATR(\beta)$;
  $iter \leftarrow iter + 1$;
until $iter < Maxiter$;

Implementation of the golf algorithm requires defining of:

1. 'weak' and 'strong' moves, as well as sets of these moves used to generate the neighborhood;
2. determination of attribute moves and the principles of creating and using the *MEM* memory (it should protect against returning to previously viewed areas of the solution space).

6. Golf algorithm for CFJS problem

One of the most important elements of algorithms based on the local search methods is the neighborhood. In case of task scheduling problems for which solutions are represented by permutations, the elements of the neighborhood are usually generated by swapping or moving the elements in the permutation. In the considered problem of minimizing the cycle time, solutions are arrangements of a permutation of operations on individual machines.

Changing of the order of operations on any machine undoubtedly does not change the assignment of operations to machines. Therefore, it is necessary to introduce a mechanism of

reallocating the operations to machines. In the case of the golf algorithm course, this change will be implemented by a strong stroke. In contrast, weak stroke will cause the change in the order of operations on a machine. Below there is an detailed description of both moves.

Let $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ be a certain solution of the CJFS problem. Therefore, on a machine $M_i \in \mathscr{M}$ there are executed operations from the set $\mathscr{O}^i$ in the order π_i. We consider two machines M_k, M_l from the same nest. While generating the neighborhood of π we will use two types of moves:

1. Insert (in short *i*-move) $i^k_{a,b}$ the equivalent of the weak stroke. This moves depicts the element $\pi_k(a)$, from position a to position b in π_k, generating permutation $i^k_{a,b}(\pi) = \beta(M_k \in \mathscr{M},\ 1 \le a, b \le n_k)$ such that $\beta_l = \pi_l$, $l = 1, 2, \ldots, k-1, k+1, \ldots, n)$ and if $b \ge a$, then

$$\beta_k(l) = \begin{cases} \pi_k(l), & \text{if } 1 \le l < a \vee b < l \le n_k, \\ \pi_k(l+1), & \text{if } a \le l < b, \\ \pi_k(a), & \text{if } l = b. \end{cases}$$

   It is similar when $b < a$.
   The set of *i*-moves of operation s is denoted by $\mathscr{I}_s$, whereas by $\mathscr{I}(\pi) = \sum_{s \in \mathscr{O}} \mathscr{I}_s$. The set of all such moves.
2. Transfer (in short *t-move*) $t^{k,l}_a$ the equivalent of strong stroke. It transfers the operation $\pi_k(a)$ from k-th machine on the last position on l-th machine (machines k, l are from the same nest). In this case $t^{k,l}_a(\pi) = \beta$, where
3. $\beta_s = \pi_s,\ s \ne k, l,\ s = 1, 2, \ldots, m$
   Moreover, operation $\pi_k(a)$ is removed from k-th machine, i.e.

$$\beta_k(j) = \pi_k(j),\ j = 1, 2, \ldots, a-1$$
$$\beta_k(j) = \pi_k(j+1),\ j = a, a+1 \ldots, n_k - 1$$

   and then we put the operation on the last position on the l-th machine, i.e.

$$\beta_l(j) = \pi_l(j),\ j = 1, 2, \ldots, n_l, \text{ and } \pi_l(n_l + 1) = \pi_k(a).$$

   There are also swaps $n_k \leftarrow n_k - 1$ and $n_l \leftarrow n_l + 1$ performed. The set $\mathscr{T}_s$ includes *t*-moves of operation s, and $\mathscr{T}(\pi) = \sum_{s \in \mathscr{O}} \mathscr{T}_s$ is the set of all such moves.

According to the definition (27) the neighborhood $\mathscr{N}(\pi)$ is generated by the compilation of all *i*-moves and *t*-moves, i.e.

$$\mathscr{N}(\pi) = \{\tau(\pi):\ i \circ t,\ i \in \mathscr{I}(\pi),\ t \in \mathscr{T}(\pi)\}.$$

In the golf algorithm AGF the move (stroke) generating the best element from the neighborhood is to be determined. It follows from property 1 and Theorem 3 that while determination of neighborhood there are only the following types of moves to be considered:

(i) generating feasible solutions,
(ii) changing the order of operations on the critical path.

In addition, in the case of *i*-moves, the operations will be repositioned immediately before the first and after the last operation in the block. This method of generating the neighborhood is successfully used in the best algorithms for solving the the flexible job shop problem with criterion $C_{\max}$ ([2, 4, 11]).

Attributes of generated solutions are stored in the *MEM* memory in the form of triplets (s, v, k). The elements s and v identify operations whereas k – the machine. In Step 1 of AGF algorithm for each triplet (s, v, k) in the *MEM*, memory from the set $\mathscr{N}(\pi)$ there are solutions removed in which operations s and v are performed on the machine k in the order s before v.

In Step 3 there is an addition of the attributes of the selected element from the neighborhood into *MEM* memory. If the execution of the complex move τ causes transfer of the operations $\pi_l(a)$ from machine M_l on position b on machine M_k, then to *MEM* memory the following three are added:

– $(\pi_k(b-1), \pi_k(a), k)$ for $k = l,\ a > b$,
– $(\pi_k(a), \pi_k(a+1), k)$ for $k = l,\ a < b$,
– $(\pi_k(b-1), \pi_k(a), k)$ and $(\pi_k(a), \pi_k(b+1), k)$ for $k \ne l$.

7. Strategy of neighborhood search

In the algorithms based on the search of the solution space the procedure for determination of the value of the objective function has a major impact on the running time of the algorithm. In case of the considered in the work problem determination of the minimum cycle time (for a single solution) has a computational complexity $O(om^2)$. Thus, instead of calculating the exact value of the cycle time, we will use lower bound which is determined much faster.

7.1. Lower bound of the minimum cycle time. Any move $\tau \in \mathscr{I}(\pi) \cup \mathscr{T}(\pi)$ can be unambiguously represented by a triple $v = (i, k, s)$ $(i \in \mathscr{O}, k \in \mathscr{M}, s = 1, 2, \ldots, n)$. Its execution can be divided into two stages:

(*i*) removal of operation i,
(*ii*) inserting of operation i on position s on the machine k.

Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ be a solution generated from π in the first stage, whereas $\beta = (\beta_1, \beta_2, \ldots, \beta_m)$ during the second stage. In the first component of the cyclic graph $G^{\oplus}(\alpha)$ we define certain paths (more precisely – the length of these paths, i.e. the sum of the weights of all vertices):

1. $R^l(j)$ $(l \in \mathscr{M}, j \in \mathscr{O})$ – length of the longest path from the vertex $\alpha_l(1)$ to vertex j,
2. $Q^l(j)$ we designate the length of the longest path from vertex j to vertex representing operation $\alpha_l(n_l)$.

The value of the formula

$$LB^l(\pi, v) = \max\{R^l(\alpha_k(s-1)),\ R^l(v) - p_v\} + \max\{Q^l(\alpha_k(s)),\ Q^l(v) - p_v\} + p_v\} \tag{28}$$

is the length of the longest path in the graph $G^{\oplus}(\beta)$, from vertex $\alpha_l(1)$ do $\alpha_l(n_l)$ passing through vertex i.

Lemma 1. If β was generated from the $\pi \in \Phi$ by making a move $v = (i, k, s) \in \mathscr{I}(\pi) \cup \mathscr{T}(\pi)$, then the minimum cycle time

$$T^o(\beta) \ge LB(\alpha) = \max_{l \in \mathscr{M}} LB^l(\alpha). \tag{29}$$

Proof. For the proof it is enough to note that the $LB^l(\pi, v)$ is equal to the length of the longest path from the vertex $\beta_l(1)$ to $\beta_l(n_l)$ passing through the vertex representing operation i. Since $LB^l(\pi, v)$ is equal to the length of one of the paths from the vertex $\beta_l(1)$ to $\beta_l(n_l)$ (not necessarily the longest one). Therefore, $LB^l(\pi, v) \leq \lambda_{\beta_l(1),2} \leq \Lambda^*$ for $l \in \mathscr{M}$, where $\lambda_{\beta_l(1),2}$ and Λ^* were determined for the solution β. □

The set $\mathscr{Z}_j(\pi) = \{i \circ t:\ i \in \mathscr{I}_j, t \in \mathscr{T}_j\}$ includes moves representing the operation $j \in \mathscr{O}$ to another position or another machine.

Lemma 2. Computation of all the lower bounds of the cycle time $LB(\pi, v)$, $v \in \mathscr{Z}_j(\pi)$, $j \in \mathscr{O}$ can be executed in time $O(o \cdot m)$.

Proof. For any move from the set $\mathscr{Z}_j(\pi)$, the first move is the same. For a fixed machine $l \in \mathscr{M}$ designation of the value $R^l(k)$ and $Q^l(k)$ for all the operations from the set $\mathscr{O}$ can be realized in time $O(o)$. Thus, execution of the first stage for all the operations has a complexity $O(m \cdot o)$. Computation of the lower bound (28) is performed in a constant time, i.e. $O(1)$. Operation j can be moved to at most $\sum_{l \in \mathscr{M}}(n_l + 1) = o + m - 1$ positions. Therefore, the computations associated with the second stage, for all the moves, can be performed in a total time $O\big((n + m) \cdot m\big)$. □

The average number of elements of the set $\mathscr{Z}_j(\pi)$ is o/m. Thus, dividing the total calculation time (Lemat 2) by the average number of $\big|\mathscr{Z}_j(\pi)\big|$ we receive an average computation time for generating one element from the neighborhood $\mathscr{N}(\pi)$. It equals $O\big(m^2\big)$.

7.2. Selection of an element from the neighborhood. The idea to use a lower bound in the procedure of neighborhood searching comes from the methods of construction optimal Branch and Bound algorithms and it consists of two phases. In the first phase solutions are ordered in accordance with decreasing values of the lower bounds of the cycle time. In the second phase the solutions are examined in accordance with the sequence from the first phase. For further solutions there is a minimum cycle time (Section 4) determined. If for some solution the minimum time, from so far calculated cycle times, is not greater than the lower bound of the considered solution, then we finish calculations (other solutions are omitted, because they have not reduced the minimum cycle time). Therefore Step 2 of AGF algorithm is modified, which takes the form:

Step 2:

PHASE 1: Sort the elements of the neighborhood
$\mathscr{N}(\pi) = \{\beta_1, \ldots, \beta_r\}$ such, that
$LB(\beta_1) \leq LB(\beta_2) \leq \ldots \leq LB(\beta_r)$;

PHASE 2: Substitute $\beta^* \leftarrow \beta_1$; $i \leftarrow 1$;
while $(i \leq r)$ **and** $\big(LB(\beta_i) < T^\circ(\beta^*)\big)$ **do**
if $T^\circ(\beta^*) < T^\circ(\beta_i)$ **then** $\beta^* \leftarrow \beta_i$;
$i \leftarrow i + 1$;

The effectiveness of the described method depends on the quality of the lower bounds which can be evaluated statistically by performing certain computational experiments. Sorting the list increases the chances of finding a good solution β^* already in the first iterations (while instruction). The conducted computational experiments confirmed the effectiveness of the proposed method.

8. Computational experiments

The main aim of the computational experiments was to determine the speed up process of calculations of the algorithm using the lower bound of the length of the cycle time relative to the algorithm defining the exact value. In the further part of the work the above mentioned versions of the algorithms will be marked with symbols AGF_{LB} and AGF. Algorithms AGF and AGF_{LB} were programmed in C++ in Visual Studio 2010. The computations were performed on an Intel I7-core 2.4 GHz, 4 GB RAM, managed by 32-bit operating system Windows 7. The experimental study was conducted on two groups of the test data:

(a) Barnes and Chambers [1], 21 instances,
(b) Brandimarte [7], 10 instances.

Individual examples differ in the number and structure of tasks, the number of machines and the operation and the degree of flexibility *Flex* (the average number of alternative machines per one operation). The first group includes instances consisting of 11–18 machines, 10–15 tasks and 100–225 operations. The degree of flexibility falls within the range 1.07–1.30. In turn, the second group of instances has a greater degree of flexibility, i.e. 1.43–4.10. The number of machines is 6–15, 10–20 tasks and 55–240 operations. The initial solutions were determined by the golf algorithm based on the metaheuristics TSM^{2}h described in the work [4] for the flexible job shop problem with the criterion of minimizing the completion time of all tasks $(C_{\max})$.

Both algorithms AGF and AGF_{LB} have been started from the same initial solution with a number of iterations 10 000 and the size of short-term memory of 15.

With every start of the algorithm the following values were set: the best solution (i.e. the value of cycle time), the number of starting procedures for the designation of the exact cycle time and the time of calculations. Then, basing on these results, for each example, there were determined:

- T^* – approximate value of the optimal length of the cycle time,
- $CPU(\mathbf{A})$ – time of computations of the algorithm $\mathbf{A}$, $\mathbf{A} \in \{AG, AG_{LB}\}$,
- $EPI(\mathbf{A})$ – the average number of solutions, for which the minimum length of the cycle was designated, per one iteration of the algorithm $\mathbf{A} \in \{AG, AG_{LB}\}$.

Tables 1 and 2 summarize the total results of the golf algorithm AGF_{LB}. The first column presents the name of example, and next: the number of tasks (n), the number of machines (m) and

Table 1
Computational results of the algorithm AGF_{LB}, for instances from group (a)

instance	$n \times m$	o	*Flex*	$C_{\max}$	T^*
mt10c1	10×11	100	1.10	927	631*
mt10cc	10×12	100	1.20	908	631*
mt10x	10×11	100	1.10	918	579
mt10xx	10×12	100	1.20	918	595.50
mt10xxx	10×13	100	1.30	918	576
mt10xy	10×12	100	1.20	905	576
mt10xyz	10×13	100	1.30	847	667
setb4c9	15×11	150	1.10	914	910.50
setb4cc	15×12	150	1.20	907	886
setb4x	15×11	150	1.10	925	876
setb4xx	15×12	150	1.20	925	883
setb4xxx	15×13	150	1.30	925	873
setb4xy	15×12	150	1.20	910	845*
setb4xyz	15×13	150	1.30	903	838*
seti5c12	15×16	225	1.07	1174	1126
seti5cc	15×17	225	1.13	1136	1468
seti5x	15×16	225	1.07	1198	1105
seti5xx	15×17	225	1.13	1197	1115
seti5xxx	15×18	225	1.20	1197	1363.50
seti5xy	15×17	225	1.13	1136	1468
seti5xyz	15×18	225	1.20	1125	1052

* – optimal solution

Table 2
Computational results of the algorithm AGF_{LB}, for instances from group (b)

instance	$n \times m$	o	*Flex*	$C_{\max}$	T^*
Mk01	10×6	55	2.09	927	36*
Mk02	10×6	58	4.10	908	26
Mk03	15×8	150	3.01	918	204*
Mk04	15×8	90	1.91	918	60
Mk05	15×4	106	1.71	918	176
Mk06	10×15	150	3.27	905	58
Mk07	20×5	100	2.83	847	153
Mk08	20×10	225	1.43	914	523*
Mk09	20×10	240	2.53	907	299
Mk10	20×15	240	2.98	925	198.67

* – optimal solution

a degree of flexibility (*Flex*). The next column presents reference values for the flexible job shop problem with criterion $C_{\max}$ (this value is the upper bound for the optimal cycle time). The last column contains values determined by the algorithm of the cycle length value.

They are significantly smaller than the upper bound $C_{\max}$. This relationship is present in 18 out of 21 examples. Only three examples: seti5cc, seti5xxx and seti5xy the obtained solutions were worse. In case of mtl0cl instance designated cycle time is more than 30% smaller than the upper bound value $C_{\max}$. In four cases the determined length of the cycle time is not an integer. It follows form the above that the critical path in the best found solution, passes through two or more components of the cyclic graph.

Table 2 shows the results of examples from group (b). Notations of columns are the same as in the case of Table 1 (column $C_{\max}$, was omitted since its values are not known). It should be emphasized that these examples have much higher values of flexibility coefficient *Flex*. For eight cases of instances, with varying degrees of flexibility the optimum solution was found.

The results showing the calculation time for the two algorithms are presented in Table 3. Before its thorough analysis, it is worth noting that the time of a single iteration of the golf algorithm depends, above all, on the size of the neighborhood, which depends on the number of:

1. operations on the critical path,
2. machines, on which operation from the critical path can be executed,
3. operations allocated to each machine.

These figures indirectly result from two parameters of a single instance: the number of operations and the degree of flexibility of the system. In the case of the algorithm AGF_{LB} an important element is the effectiveness of the lower bound bound. It allows us for the omission of certain elements from the neighborhood.

Results presented in the Table 3 clearly show that the time of running of the algorithm AGF_{LB} is much shorter than the algorithm *AGF*. Algorithm *AGF* operates from 3.1 to as much as 332 times longer than the algorithm AGF_{LB}. In both algorithms the most time (computing power) took the procedure for the designation of the exact value of the cycle time. In the algorithm AGF_{LB}, due to pre-assessing the lower bound of this value, in most cases the exact value of the cycle time is determined, on average only for a few best solutions from the neighborhood. The conducted computations show that this number (column *EPI* AGF, Table 3) practically does not depend on the size of the instance and slightly increases with the degree of flexibility. It follows from the additional testing of the algorithm AGF_{LB} that the determination of the exact length of the cycle time consumes at least 90% of the total computation time. Therefore, the time of the running of algorithm AGF_{LB} is much less sensitive to the number of viewed elements of the neighborhood. For instance Mkl0 the value *EPI* of the algorithm AGF_{LB} is 1.6 whereas for the algorithm *AGF* it is 171, i.e. approximately 107 times more. The running time of the algorithm AGF_{LB} executing 10,000 iterations, in most cases, it was a few dozen seconds, while in the worst case it did not exceed 300 seconds. Given the fact that the best solutions were determined after a small number of iterations, it is possible to state that this algorithm can be successfully applied to solve practical examples of large sizes.

Table 3
Times of algorithm computations (in seconds), for instances from the groups (a) and (b)

instance	$n \times m$ (o, $flex$)	algorithm AGF_{LB}		algorithm AGF		Acceleration
		CPU	*EPI*	*CPU*	*EPI*	
mt10c1	10×11 (100, 1.10)	0.20	3.2	6.96	6.5	34.8
mt10cc	10×12 (100, 1.20)	0.09	3.6	13.87	10.3	154.1
mt10x	10×11 (100, 1.10)	14.63	1.8	113.01	13.9	7.7
mt10xx	10×12 (100, 1.20)	9.13	1.1	28.50	2.7	3.1
mt10xxx	10×13 (100, 1.30)	36.71	4.6	179.60	18.5	4.8
mt10xy	10×12 (100, 1.20)	16.93	2.2	81.12	8.9	4.7
mt10xyz	10×13 (100, 1.30)	18.49	1.9	101.58	10.8	5.4
setb4c9	15×11 (150, 1.10)	37.64	3.0	187.57	13.5	4.9
setb4cc	15×12 (150, 1.20)	45.81	3.4	298.98	21.7	6.5
setb4x	15×11 (150, 1.10)	35.87	2.9	190.93	14.5	5.3
setb4xx	15×12 (150, 1.20)	42.83	3.2	231.09	16.0	5.3
setb4xxx	15×13 (150, 1.30)	51.09	3.6	298.85	19.6	5.8
setb4xy	15×12 (150, 1.20)	21.39	3.2	65.38	22.0	3.0
setb4xyz	15×13 (150, 1.30)	4.74	3.2	47.47	28.8	10.0
seti5c12	15×16 (225, 1.07)	83.61	2.7	577.94	17.9	6.9
seti5cc	15×17 (225, 1.13)	97.44	2.9	828.96	27.4	8.5
seti5x	15×16 (225, 1.07)	80.66	2.7	510.24	17.4	6.3
seti5xx	15×17 (225, 1.13)	91.43	3.0	673.52	24.5	7.3
seti5xxx	15×18 (225, 1.20)	298.70	11.0	1367.38	56.9	4.5
seti5xy	15×17 (225, 1.13)	101.59	2.9	794.62	27.4	7.8
seti5xyz	15×18 (225, 1.20)	103.20	3.0	2514.71	47.9	24.3
Mk01	10×6 (55, 2.09)	0.01	4.1	0.48	155.0	48
Mk02	10×6 (58, 4.10)	18.99	7.9	39.52	19.3	2.0
Mk03	15×8 (150, 3.01)	0.31	98.2	5.71	976.3	18.4
Mk04	15×8 (90, 1.91)	56.68	19.0	193.37	47.3	3.4
Mk05	15×4 (106, 1.71)	10.58	1.0	853.09	574.7	80.6
Mk06	10×15 (150, 3.27)	52.64	1.1	6180.64	726.6	117.4
Mk07	20×5 (100, 2.83)	24.25	7.2	1893.40	766.3	78.0
Mk08	20×10 (225, 1.43)	2.89	6.8	51.48	199.2	17.8
Mk09	20×10 (240, 2.53)	2.25	17.0	747.14	842.7	332.0
Mk10	20×15 (240, 2.98)	118.99	1.6	10 h	171.1	302.5

9. Summary

In the paper, a cyclical flexible job shop problem was presented. A graph model was also constructed for a fixed order of operations on individual machines. Based on the analysis of the paths in the graph, the theorems were proven which enable efficient testing for the feasibility of any order of operation execution, setting the minimum cycle time, and its lower bound. A new method of constructing heuristic algorithms was used with the so-called golf neighborhood, enabling both intensification and diversification of the search process for the feasible solutions. In the algorithm a unique strategy was applied for the determination of solutions, with minimum cycle time significantly accelerating calculations. The efficiency of the algorithm has been the subject of experimental tests on instances taken from the literature. Based on these results, it can be stated that the presented approach in a short time determines the approximate solutions accepted in practice for the problem of minimal cycle time determination. In practice, the proposed properties can be successfully used in the construction of new fast algorithms for solving difficult problems from the field of optimization of cyclic manufacturing systems. cyclic manufacturing systems.

Acknowledgement. The paper was partially supported by the National Science Centre of Poland, grant OPUS no. DEC 2017/25/B/ST7 /02181.

References

[1] J.W. Barnes and J.B. Chambers, “Flexible job shop scheduling by tabu search”, Graduate program in operations research and industrial engineering The University of Texas at Austin, Technical Report Series, ORP96–09 (1996).

[2] W. Bożejko, M. Uchroński, and M. Wodecki, "The new golf neighborhood for the flexible job shop problem", *Procedia Computer Science* 1, 289–296 (2010).
[3] W. Bożejko, M. Uchroński, and M. Wodecki, "Parallel metaheuristics for the flexible job shop problem", *Lecture Notes in Artificial Intelligence* 6114, 395–402 (2010).
[4] W. Bożejko, M. Uchroński, and M. Wodecki, "Parallel hybrid metaheuristics for the flexible job shop problem", *Computers and Industrial Engineering* 59, 323–333 (2010).
[5] W. Bożejko, M. Uchroński, and M. Wodecki, "Solving the flexible job shop problem on multi GPU", *Procedia Computer Science* 9, 2020–2023 (2012).
[6] W. Bożejko, J. Pempera, and M. Wodecki, "Parallel simulated annealing algorithm for cyclic flexible job shop scheduling problem", *Lecture Notes in Artificial Intelligence* 9120, 603– 612 (2015).
[7] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search", *Annals of Operations Research* 41, 157–183 (1993).
[8] P. Brucker and T. Kampmeyer, "Cyclic job shop scheduling problems with blocking", *Annals of Operations Research* 159, 161– 181 (2008).
[9] P. Brucker and T. Kampmeyer, "A general model for cyclic machine scheduling problems", *Discrete Applied Mathemathics* 156, 13 2561–2572 (2008).
[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and S. Clifford, *Introduction to Algorithms*, MIT Press and McGraw-HilL, Second Edition, MIT Press and McGraw-Hill, 2001.
[11] J. Grabowski and J. Pempera, "New block properties for the permutation flow shop problem with application in tabu search", *Journal of Operational Research Society* 52, 210–220 (2001).
[12] L. Houssin, "Cyclic jobshop problem and (max,plus) algebra", *World IFAC Congress*, Milan, 2717–2721 (2011).
[13] E. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job shop scheduling problem with multi-purpose machines", *Operations Research Spektrum* 15, 205–215 (1994).
[14] H.Z. Jia, A.Y.O. Nee, J.Y.H. Fuh, and Y.F. Zhang, "A modified genetic algorithm for distributed scheduling problems", *International Journal of Intelligent Manufacturing* 14, 351–362 (2003).
[15] I. Kacem, S. Hammadi, and R. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job shop scheduling problems", *IEEE Transactions on Systems, Man, and Cybernetics Part C* 32(1), 1–13 (2002).
[16] T. Kampmeyer, "Cyclic scheduling problems", Ph. D. Thesis, University Osnabriick (2006).
[17] M. Mastrolilli and L.M. Gambardella, "Effective neighborhood functions for the flexible job shop problem", *Journal of Scheduling* 3(1), 3–20 (2000).
[18] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for permutation flow shop problem", *European Journal of Operational Research* 91, 160–175 (1996).
[19] M. Wodecki, "A block approach to earliness-tardiness scheduling problems", *International Journal on Advanced Manufacturing Technology* 40, 797–807 (2009).
[20] T. Sawik, "A mixed integer program for cyclic scheduling of flexible flow lines", *Bull. Pol. Ac.: Tech.* 62(1), 121–128 (2014).
[21] W. Xia and Z. Wu, "An effective hybrid optimization approach for multiobjective flexible job shop scheduling problem", *Computers and Industrial Engineering* 48, 409–25 (2005).