Computers & Industrial Engineering 81 (2015) 158–166

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Block approach to the cyclic flow shop scheduling

Wojciech Bożejko [a,*], Mariusz Uchroński [b], Mieczysław Wodecki [c]

[a] Department of Automatics, Mechatronics and Control Systems, Faculty of Electronics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
[b] Wrocław Centre of Networking and Supercomputing, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland
[c] Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland

ARTICLE INFO

Article history:
Received 31 May 2014
Received in revised form 10 November 2014
Accepted 5 January 2015
Available online 16 January 2015

Keywords:
Scheduling
Cyclic flow shop problem
Block properties

ABSTRACT

The cyclic flow show problem with machine setups is considered in this paper. It relies in producing of a set of certain elements in fixed intervals of time (cycle time). Process optimization is reduced to minimization of cycle time, i.e., the time after which the next batch of the same elements may be produced. Since the problem is strongly NP-hard, in order to solve it an approximate algorithm was used. There is presented a graph model of a problem and the so called block eliminating properties capable of reducing, in a significant way, neighborhood used in the tabu search algorithm. Conducted computational experiments confirm high efficiency of the proposed technique.

1. Introduction

For many years, one could observe an increasing market demand for diversity (multiassortment) of production. This may be provided, among many other issues, by means of cyclic production. In fixed intervals of time (cycle time) a certain 'batch' of assortment (a mix of kit, a set) is produced. Process optimization is typically reduced to minimization of cycle time. Proper selection of mix and cycle time enables not only to meet demand, but also to improve efficacy and effectiveness of machinery use. Thus, recently, one can observe a significant increase of interest in the problems of cyclic tasks scheduling theory. For they are usually important and difficult, (mostly *NP-hard*) problems, from the standpoint of not only theory, but also practice.

A comprehensive overview of the state of knowledge concerning the cyclic task scheduling problem can be found in the work of Levner, Kats, Lopez, and Cheng (2010) analyzing the issues of computational complexity of algorithms for solving various types of cycle scheduling problem. Here, in particular NP-difficult problems of various cyclic types including a variety of criterion functions and additional constraints (*no wait*, *no buffer*, etc.) are considered.

In the scientific work by Panwalkar, Dudek, and Smith (1973) on task scheduling it was found that 75% of problems occurring in practice requires at least one setup dependent on the order of tasks execution. However, in 15% of the problems a setup of all tasks should be taken into consideration. Nevertheless, in the vast majority of works, in the field of scheduling setups are not taken into account at all. This applies both to single and multi-machine problems and to different goal functions.

Cyclic problems belong to unique, relatively little researched subclass of scheduling problems. However, more and more practitioners and theorists show interest in the above issues mainly due to their great practical importance and the attempt to overcome difficulties in constructing relatively efficient algorithms. Strong NP-hardness of the simplest versions of the above problem limits the scope of applications of exact algorithms only to instances of small size.

In this paper a multi-machine cyclic production system is considered, in which any element of the fixed batch (mix) passes successively through each of the machines (permutation flow shop, see Nowicki & Smutnicki (1996) and Grabowski & Wodecki (2004)). Between successively produced elements there must be a setup of machines performed. The problem consists in finding minimization of cycle time, i.e. the time after which the next batch of the same elements may be produced. There will be proven *strong NP-hardness* already for some special case of the problem under consideration. The hardness of the problem may be confirmed by the fact that some simplified version boils down to solving the problem of a traveling salesman. For this reason, in order to effectively determine solutions a fast approximate algorithm is used. There will be also the so called 'block elimination properties' proven which will be used in the construction of tabu search algorithm. They provide an indirect search of certain subsets of solution space not only accelerating calculations, but also, at the same time, improving quality of designated solutions.

* Corresponding author.
E-mail addresses: wojciech.bozejko@pwr.edu.pl (W. Bożejko), mariusz.uchronski@pwr.edu.pl (M. Uchroński), mwd@ii.uni.wroc.pl (M. Wodecki).

http://dx.doi.org/10.1016/j.cie.2015.01.004
0360-8352/© 2015 Elsevier Ltd. All rights reserved.

Continuous flow production systems are among the most commonly encountered in industry. Everywhere where the production process consists of the following successive stages, one deals with such systems. Each stage of production is realized in a separate slot supplied with specialized machinery. In the literature, there are also other names describing this problem such as: a hybrid flow-shop system or a flexible production line. The hybrid flow-shop problem with setups was considered, among many others, in the works (Bożejko, Gniewkowski, Pempera, & Wodecki, 2014; Cavory, Dupas, & Goncalves, 2005; Caggiano & Jackson, 2008; Dbrowski, Pempera, & Smutnicki, 2007; Fournier, Lopez, & Lan Sun Luk, 2002; Kampmeyer, 2006; Sawik, 2014; Sawik, 2012).

The work consists of six chapters. The first and the second chapter, based on literature results, include a brief introduction and basic definitions related to cyclic scheduling tasks. The next two chapters constitute the new, genuine results of the authors. There are presented and proven the so called 'block properties' enabling elimination of certain elements from the neighborhood of tabu search algorithm. The last two chapters include the results of computational experiments and conclusions.

2. Problem formulation

Considered in the paper system of manufacturing is an extension of *strongly NP-hard*, classical in theory of scheduling, permutation flow problem (denoted in literature by $F^*||C_{\max}$). It can be formulated as follows:

Problem: There is given a set of n tasks $\mathcal{J} = \{1, 2, \ldots, n\}$, to be carried out recurrently (in a repeated manner) on machines from the set $\mathcal{M} = \{1, 2, \ldots, m\}$. Any task should be performed consecutively, on each m machine $1, 2, \ldots, m$ (technological line). The task $j \in \mathcal{J}$ is a sequence m of operations $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$. The operation $O_{k,j}$ corresponds to the activity of execution of j task on machine k, in time $p_{k,j}$ $(k = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$. After completion of certain operation and before the start of the next one there must be a setup of machine performed. Let $s^k_{i,j}$ $(k \in \mathcal{M},\ i \neq j\ i, j \in \mathcal{J})$ be a time of a setup of machine k between operation $O_{k,i}$ and $O_{k,j}$. There must be the order of tasks execution (the same on each machine) designated, which minimizes cycle time, i.e. the time of the beginning of tasks execution from the set $\mathcal{J}$ in the next cycle. The following restrictions must be fulfilled:

(a) each operation can be performed only by one determined by the production process, machine,
(b) no machine can perform at the same time more than one operation,
(c) production process order of operations execution must be preserved,
(d) execution of any operation cannot be interrupted before its completion,
(e) between successively executed, on the same machine, operations there must be a setup performed,
(f) each task is performed sequentially after the completion of cycle time.

The set of tasks $\mathcal{J}$ executed in a single cycle is called (*minimal part set*) – MPS. MPSs are processed directly one after the other in a cyclic manner. In each of the MPSs the tasks from the set $\mathcal{J}$ are performed on each machine in the same order (permutation flow shop). Thus, any order of tasks on machines can be represented by a permutation $\pi = (\pi(1), \ldots, \pi(n))$ of elements from the set $\mathcal{J}$. Let Φ be the set of all such permutations.

The considered in the paper problem boils down to such determining of the tasks permutations (i.e. moments of tasks execution start on machines that meet the constraints (a)–(f), that the cycle time (time after which any task is performed in the next MPS-e) was minimal. In brief, this problem will be denoted by **CFS**.

2.1. Mathematical model

Let $\pi \in \Phi$ be an order of tasks execution on machines (the same for all MPSs). By $[S^h]_{m\times n}$ it is denoted the matrix of the beginning of tasks execution in h-th MPS, where $S^h_{i,\pi(j)}$ is the starting time of task $\pi(j)$ on machine i in h-th MPS. It is assumed that not only the sequence of tasks is cyclically repeated in each of the MPSs, but that timetable of system operation (i.e., execution of the following MPSs) is cyclic. This means that there is a constant (period) $T(\pi)$ such that

$$S^{h+1}_{i,\pi(j)} = S^h_{i,\pi(j)} + T(\pi),\ \ i = 1, \ldots, m,\ j = 1, \ldots, n,\ h = 1, 2, \ldots \quad (1)$$

The period $T(\pi)$ is undeniably dependent on permutation π and is called *cycle time* of the system for the permutation $\pi \in \Phi$. The minimum value $T(\pi)$ will be called *minimum cycle time* and will be denoted by $T^*(\pi)$.

Optimal value of time of the cycle $T^*(\pi^*)$ (solution to the problem **CFS**) can be determined by solving the following optimization task: designate

$$T^*(\pi^*) = \min\{T^*(\pi) :\ \pi \in \Phi\}, \quad (2)$$

with constraints:

$$S^h_{i,\pi(j)} + p_{i,\pi(j)} \leqslant S^h_{i+1,\pi(j)},\ \ i = 1, \ldots, m-1,\ j = 1, \ldots, n, \quad (3)$$

$$S^h_{i,\pi(j)} + p_{i,\pi(j)} + s^i_{\pi(j),\pi(j+1)} \leqslant S^h_{i,\pi(j+1)},\ \ i = 1, \ldots, m,\ j = 1, \ldots, n-1, \quad (4)$$

$$S^h_{i,\pi(n)} + p_{i,\pi(n)} + s^i_{\pi(n),\pi(1)} \leqslant S^{h+1}_{i,\pi(1)},\ \ i = 1, \ldots, m, \quad (5)$$

$$S^{h+1}_{i,\pi(j)} \leqslant S^h_{i,\pi(j)} + T^*(\pi),\ \ i = 1, \ldots, m-1, \quad (6)$$

where $h = 1, 2, \ldots$.

The last constraint (6) is characteristic for cyclic production as it determines the relationship between beginning times of tasks execution in successively performed MPSs.

Without loss of generality, we can assume that the starting time of the first task performance on the first machine in the first MPS is $S^1_{1,\pi(1)} = 0$. For a fixed permutation $\pi \in \Phi$ and the first MPS, by

$$T_k(\pi) = \sum_{i=1}^{n-1} (p_{l,\pi(i)} + s^k_{\pi(i),\pi(i+1)}) + p_{l,\pi(n)} + s^k_{\pi(n),\pi(1)} \quad (7)$$

it is denoted the time of the tasks execution in order π, together with setups on k-th machine (this sum also includes setup time between the last operation $\pi(n)$ of the first MPS, and the first $\pi(1)$ operation of the second MPS). In short, this time will be called k-th *peak*.

It is easy to see that for permutation of tasks $\pi \in \Phi$ minimum cycle time is

$$T^*(\pi) = \max\{T_k(\pi) :\ k = 1, 2, \ldots, m\}. \quad (8)$$

2.2. The problem with zero setup time

Let us consider a simplified version of the problem **CFS**, in which machine setup times, between successively performed operations, are equal to zero. Thus, for any permutation $\pi \in \Phi$, $s^k_{\pi(i),\pi(i+1)} = 0,\ i = 1, 2, \ldots, n-1$ and $s^k_{\pi(n),\pi(1)} = 0,\ k = 1, 2, \ldots, m$. Then (7), time of the operation execution by k-th machine is

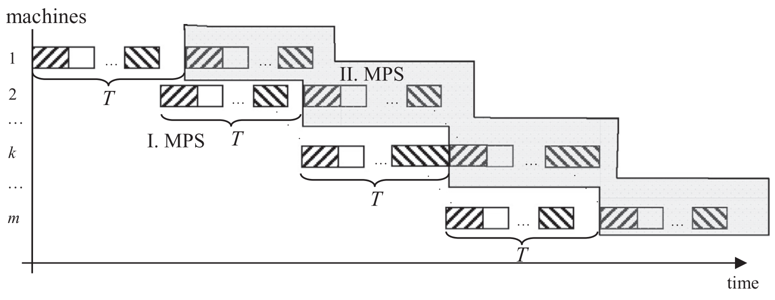

Fig. 1. Gannt chart for the two first MPSs.

$$T_k(\pi) = \sum_{i=1}^{n} p_{k,\pi(i)} \tag{9}$$

and does not depend on the order of tasks π execution. It is easy to notice that in this case the optimum cycle time

$$T^*(\pi^*) = \max\{T_k(\pi) : \; k = 1, 2, \ldots, m\}, \tag{10}$$

where π is any permutation of tasks from the set $\mathcal{J}$. Value $T^*(\pi^*)$ may be determined in $O(nm)$ time.

Fig. 1 shows a fragment of Gantt chart for the first two MPS-s of **CFS** problem with zero setup times. The optimal cycle time $T^*(\pi^*) = T$ (maximum in equality (10)) is achieved for k-th machine.

3. Properties of the problem

In this section certain properties are proved. They can be used in the construction of algorithms used for solving cyclic flow shop problem with machine setup providing an indirect search of some subsets of solutions space. In the following part of the work it is assumed that machine setup times are symmetric and satisfy the triangle condition.

3.1. Designation of cycle time

Let $\pi \in \Phi$ be a fixed permutation of tasks. Since the order of tasks execution in the frames of each MPS is the same, then it is sufficient to determine the starting point of the execution of tasks $S^h_{k,\pi(i)}$ for *the first* MPS and make the appropriate reallocation of size $kT(\pi)$. In this way the starting points of tasks in the next MPSs are obtained.

Fig. 2 shows a fragment of Gantt chart for the first two MPSs. It was assumed that the maximum of the equality (8) was achieved for k-th machine.

Designation of optimal cycle time for the flow shop problem with machine setup boils down to determining of the optimal permutation π^*, for which

$$T^*(\pi^*) = \min\{T^*(\pi) : \; \pi \in \Phi\}, \tag{11}$$

where

$$T^*(\pi) = \max\{T_l(\pi) : \; l = 1, 2, \ldots, m\}. \tag{12}$$

Let us assume that the maximum in Eq. (12) has been achieved for k-th machine, i.e. $T^*(\pi) = T_k(\pi)$. The sequence of machine operating time (peaks) $T_1(\pi), T_2(\pi), \ldots, T_m(\pi)$ is considered. It is symbolically shown in Fig. 3.

The necessary condition for reducing minimum cycle time $T^*(\pi)$ is the reduction of the maximum peak $T_k(\pi)$. In order to achieve it, for k-th machine $(k \in \mathcal{M})$ a symmetric full graph

$$H_k = \langle V, E; p; s \rangle, \tag{13}$$

with the burdened vertices and edges was constructed, wherein:

- set of vertices: $V = \mathcal{J}$,
- set of edges: $E = \{\{v, u\} : \; v \neq u, \; v, u \in \mathcal{J}\}$,
- weighs of vertices: $p : \; V \to R, p(v) = p_{k,v} \; v \in V$,
- weighs of edges: $s : \; E \to R, \; \forall e \in E, \; s(e) = s^k_e$.

Weights of vertices are equal to times of operation execution, whereas weighs of edges – equal to setup times.

A part of the structure of graph H_k with sample weights of vertices and edges is shown in Fig. 4.

Lemma 1. *For permutations of tasks $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, time of execution of tasks $T_k(\pi)$ (7) on k machine is equal to the length $L_k(\pi)$ of Hamiltonian cycle $P_k(\pi) = (\pi(1), \pi(2), \ldots, \pi(n), \pi(1))$ in graph $H_k, \in \mathcal{M}$.*

Proof. The proof is immediate. It follows directly from the definition (7) of operating time of machine $T_k(\pi)$ and the Hamiltonian cycle length $L_k(\pi)$. □

Lemma 2. *Permutation $\beta \in \Phi$ is the optimal order of tasks execution on k-th machine (i.e., it minimizes value (7)) if and only if the sequence of vertices $P^*_k = (\beta(1), \beta(2), \ldots, \beta(n), \beta(1))$ is traveling salesman cycle in the graph $H_k, \; k \in \mathcal{M}$.*

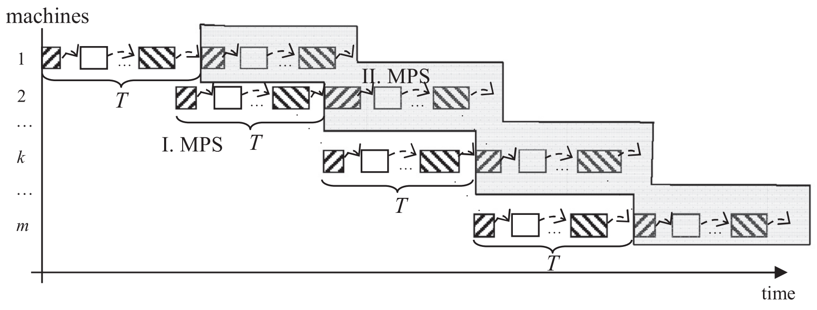

Fig. 2. Gantt chart for the first two MPSs for the problem of machine setup.

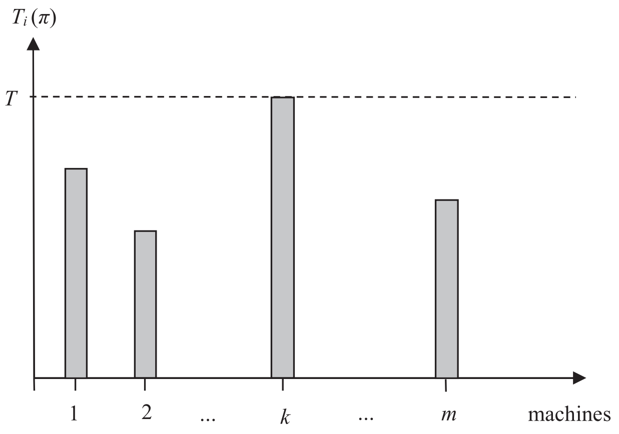

Fig. 3. Values $T_k(\pi),\ k = 1, 2, \ldots, m$.

Proof. Since the traveling salesman problem is Hamiltonian cycle with minimum length, the proof follows directly from Lemma 1. □

Let L_k^* be the optimal length of a traveling salesman cycle P_k^* in graph $H_k,\ k = 1, 2, \ldots, m$.

Remark 1. For two different machines k i l ($k \neq l,\ k, l \in \mathcal{M}$) task execution times $L_k^\star = L_l^\star$ or $L_k^\star \neq L_l^\star$.

Remark 2. The optimum value of the cycle time

$$T^*(\pi^*) \geqslant \min\{L_k^\star : k = 1, 2, \ldots, m\}. \tag{14}$$

Theorem 1. *If the time of the tasks execution are the same (i.e. $O_{i,j} = const,\ i \in \mathcal{M},\ j \in \mathcal{J}$) and times of machine setups between operations on each machine are the same ($s_{i,j}^k = s_{i,j},\ k \in \mathcal{M},\ i \neq j,\ i, j \in \mathcal{J}$), then the problem of finding the optimal cycle time is strongly NP-hard.*

Proof. In this case, any two graphs H_k i H_l ($k \neq l,\ l = 1, 2, \ldots, m$) are identical, the lengths of a traveling salesman cycles in these graphs are the same. It is easy to see (using the inequality (14)) that designation of an optimal cycle time is reduced to designation of a traveling salesman cycle in any graph $H_k \in \mathcal{M}$, for example, in H_1. Thus, the problem is strongly NP-hard. □

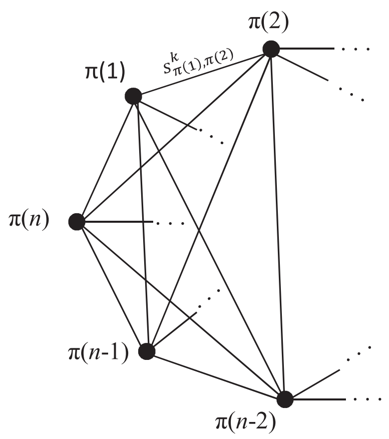

Fig. 4. Part of H_k graph.

Proposal 1. Problem **CFS** considered in the paper is strongly NP-hard.

Let $T^*(\pi)$ be the minimum cycle time for a permutation $\pi \in \Phi$. Let us also assume that $T_k(\pi)$ is the dominant peak. Therefore

$$T^*(\pi) = T_k(\pi) = L_k(\pi),$$

where $L_k(\pi)$ is the length of Hamiltonian cycle $(\pi(1), \pi(2), \ldots, \pi(n), \pi(1))$ in the graph $H_k(\pi)$. Therefore, '*in order to shorten the cycle time $T(\pi)$ we must reduce the value of the dominant peak, thus – reduce the length of Hamiltonian cycle in the graph $H_k(\pi)$*'.

Reducing of the length of Hamilton cycle in a graph H_k requires generation from π of the new permutation, i.e. it requires a change of the position of some elements in π. It should be checked whether, after this change, there is no, on any of the machines, new (larger) dominant peak. The ideas of the algorithm for determining optimal cycle time $T^*(\pi^*)$ (solution to the problem **CFS**) based on the above considerations, can be put down as follows:

```
$\pi$ – current solution;
Step 1:
  let $T_k(\pi) = \max\{T_i(\pi) :\ i = 1, 2, \ldots, m\}$;    {maximum peak}
Step 2:
  generate from $\pi$ permutation $\beta$
  swapping position of certain elements in $\pi$;    {move execution}
Step 3:
    if
        $T_i(\beta) < T_k(\pi),\ i = 1, 2, \ldots, m,\ i \neq k$    {improvement}
    then
        $\pi \leftarrow \beta$;
    if (end condition) then STOP else go to Step 1.
```

In the following part of the paper it is proven that certain moves (which do not reduce the dominant peak) can be omitted.

3.2. Tasks blocks

Let $\pi_k^* = (\pi_k^*(1), \pi_k^*(2), \ldots, \pi_k^*(n), \pi_k^*(1))$, be a traveling salesman cycle in the graph $H_k(\pi),\ k \in \mathcal{M}$. It is an optimal sequence (due to sum of times) of tasks execution on k-th machine. This permutation will be called *a pattern* for k-th machine.

Let

$$B = (\pi(a), \pi(a+1), \ldots, \pi(b)), \tag{15}$$

be a sequence of occurring immediately after one another tasks in permutation $\pi \in \Phi$, π_k^* – *pattern* for k-th machine and $u,\ v$ ($u \neq v,\ u, v = 1, \ldots, n$) a pair of positions of elements such that:

- **W1**: $\pi(a) = \pi_k^*(u), \pi(a+1) = \pi_k^*(u+1), \ldots, \pi(b-1) = \pi_k^*(v-1), \pi(b) = \pi_k^*(v)$, lub
- **W2**: $\pi(b) = \pi_k^*(u), \pi(b-1) = \pi_k^*(u+1), \ldots, \pi(a+1) = \pi_k^*(v-1), \pi(a) = \pi_k^*(v)$
- **W3**: B is the maximum subsequence due to the fact that it cannot be enlarged either by an element $\pi(a-1)$, or by $\pi(b+1)$, satisfying the constraints **W1**or **W2**).

If the sequence of tasks (15) in a permutation π satisfies the conditions **W1** and **W3** or **W2** and **W3**, then we call it a *block* on k-th machine ($k \in \mathcal{M}$).

If the number of elements is $b - a \geqslant 2$, then such block without the first and the last element is called *internal block*.

Below an algorithm for determining permutation π, of the first block on k-th machine is presented.

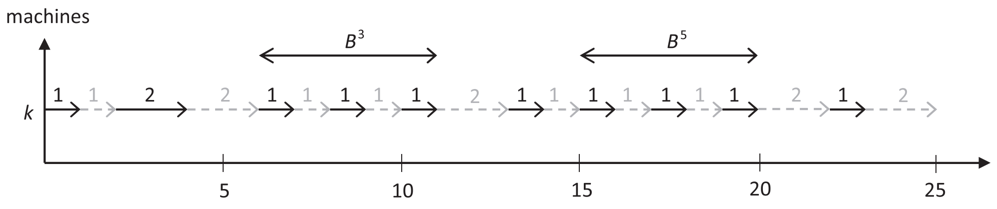

Fig. 5. Gantt chart for the k-th machine.

Algorithm. ***A*-block**

```
Input: permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$;  $\pi^*_k$ – pattern;
Output: subsequence (block)
        $\pi^{\mathcal{T}} = (\pi(l), \pi(l+1), \ldots, \pi(t-1), \pi(t))$;
    Let $\pi(l)$ be the first element in $\pi$ such that
    $\pi(l) = \pi^*_k(j)$ for certain $j$ $(1 \leqslant j \leqslant n)$;
    $\pi^{\mathcal{T}} \leftarrow \pi(l)$;
    $k \leftarrow l$;
    while $j < n$ do
    begin
      if $\pi(t+1) = \pi^*_k(j+1)$ then
      $\pi^{\mathcal{T}} \leftarrow (\pi^{\mathcal{T}}, \pi(t+1))$;
      $t \leftarrow t+1$; $j \leftarrow j+1$;
    end.
```

The computational complexity of the algorithm is $O(n)$.

Theorem 2. *For any machine there is a break of permutation $\pi \in \Phi$ into blocks.*

Proof. For a fixed machine $k \in \mathcal{M}$, considering the subsequent elements in the permutation π (starting from the first $\pi(1)$) and using the algorithm $\pi(1)$) we designate subsequence $(\pi(1), \ldots, \pi(s))$ – which constitutes the first block. Then, this procedure is continued (i.e., by re-using of the **A-block** algorithm) starting from the element of $\pi(s+1)$.

Let $\mathcal{B}$ be a sequence of subpermutations received as a result of the above described procedure. It is easy to show that $\mathcal{B}$ contains sequences of elements from the permutation π such that.

(1) elements $\mathcal{B}$ are blocks,
(2) subsequences from $\mathcal{B}$ have different elements,
(3) each element from the set $\mathcal{J}$ belongs to certain subsequence from $\mathcal{B}$.

Therefore $\mathcal{B}$ is a break of π into blocks, which completes the proof of the theorem. □

The algorithm of partitioning n-element permutations of tasks into blocks has a computational complexity $O(n)$.

Example 1. An instance of the **CFS** problem with a number of tasks $n = 10$ is considered. Let us assume, that there is a maximum peak on k-th machine for a tasks order $\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$. A part of the Gantt chart for the first MPS for this machine is shown in Fig. 5. Solid lines represents operations (machines work), dotted lines – setups. Numbers over arcs represent operations processing times of setups times, respectively. All other setups times between operations executed on this machines equal to 1. For a tasks execution order π the time of the machines work (with its setup times after execution of the last operation) and before the execution of the first operation in the next MPS is $T_k(\pi) = 25$.

The pattern, i.e. optimal tasks schedule for this machine, is a permutation

$\pi^* = (10, 3, 4, 5, 2, 6, 1, 9, 8, 7)$.

Value of the k-th peak $T_k(\pi^*) = 21$. Applying an **A-block** algorithm a partition $\mathcal{B} = (B^1, B^2, B^3, B^4, B^5, B^6)$ of the permutation π into blocks was made. This partition contains:

1. four one-elementary blocks: $B^1 = \pi(1)$, $B^2 = \pi(2)$, $B^4 = \pi(6)$, $B^6 = \pi(10)$,
2. a block $B^3 = (\pi(3), \pi(4), \pi(5))$ fulfilling constraints **W1** and **W3**, where $a = 3$, $u = 2$ and $b = 5$, $v = 4$,
3. a block $B^5 = (\pi(7), \pi(8), \pi(9))$ fulfilling constraints **W2** and **W3**, where $a = 7$, $u = 8$, and $b = 9$, $v = 10$. □

Lemma 3. *If $(v_1, v_2, \ldots, v_n, v_1)$ is a traveling salesman cycle in graph H_k $(k \in \mathcal{M})$, then $(v_t, v_{t+1}, \ldots, v_l)$ $(1 \leqslant t < l \leqslant n)$ is the shortest path from vertex v_t to v_l including elements from set $\{v_{t+1}, v_{t+2}, \ldots, v_{l-1}\}$.*

Proof. Let $C_H = (v_1, v_2, \ldots, v_n, v_1)$ be a traveling salesman cycle in graph H_k $(k \in \mathcal{M})$. We consider a subsequence (path) $C_{t,l} = (v_t, v_{t+1}, \ldots, v_l)$ $(1 \leqslant t < l \leqslant n)$ from sequence C_H.

It is assumed indirectly that $C_{t,l}$ is the shortest path in a graph H_k from vertex v_t to v_l including all vertices from a set $W = \{v_{t+1}, v_{t+2}, \ldots, v_{l-2}, v_{l-1}\}$. There is a path $d_{t,l} = (v_t, v'_{t+1}, v'_{t+2}, \ldots, v'_{l-2}, v'_{l-1} v_l)$, $v'_i \in W$ from vertex v_t to v_l such that że $L(C_{t,l}) > L(d_{t,l})$ (where $L(\cdot)$ is the length of the path). Then, putting C_H into traveling salesman cycle, by replacing path $C_{t,l}$, with path $d_{t,l}$ we obtain Hamiltonian cycle with the length smaller than traveling cycle length C_H, which stays in opposition with the indirect assumption. □

Theorem 3. *If permutation β was generated from permutation π by swapping the order of elements in certain internal block on machine $k \in \mathcal{M}$, then*

$T_k(\beta) \geqslant T_k(\pi)$.

Proof. In the proof, use Lemma 3. □

From Theorem 3 stems the so called '*block elimination property*'. By generating neighborhoods in tabu search algorithm, solutions determined by changing the order of elements in internal block can be skipped. This is due to the fact that they do not offer any immediate improvement of solution. This feature greatly speeds up algorithm action.

4. Tabu search method

In order to solve the problem of determining optimal cycle time there will be tabu search (TS) algorithm used. On the basis of the current publications it can be stated that it is currently one of the most effective, and at the same time – deterministic, methods for construction of approximation algorithms, which guarantee repetitiveness of computations.

The main idea of the TS method involves starting from an initial job schedule and searching through its neighborhood for a solution with the lowest value of the cost function. The search then is repeated starting from the best solution found and the process is continued. One of the main ideas of tabu search algorithm is the use of a forbidden (so called tabu) moves list to avoid cycling, overcoming local optimum, or continuing the search in a too narrow region and to guide the search process to the solutions regions which have not been examined. The tabu list records some attributes of the performed moves. The elements of this list, for the current iteration, determine the subset of forbidden solutions. A move having prohibited attributes is forbidden, although, it can be performed if it is sufficiently profitable.

An essential element of the tabu search method is neighborhood, i.e., to be specific – method of how it is generated and searchable. In order to reduce the execution time of a single iteration, the subneighborhoods generated with the use of 'block elimination properties' will be used.

4.1. Generating neighborhoods

The literature describes many moves based on swapping the execution order of tasks. On the basis of Theorem 3 one can draw the assumption that 'insert' type of move (*i-move*) involving swapping tasks before or after the block, may be an effective method of neighborhood generation for the problem under consideration. Generally, such move boils down to swapping of tasks from its position in the permutation into the position – before or after another task. More specifically, for various positions t, l in a permutation $\pi \in \Phi$ insert type of move (abbreviated to *i*-move) i_l^t generates a new permutation $\pi_l^t = i_l^t(\pi)$ by swapping task $\pi(t)$ from position t to position l in π. Exactly such moves are described and used in the algorithms described in the works of year Wodecki (2009) and Bożejko and Wodecki (2008).

If $\mathcal{I}(\pi)$ is the set of *i-moves* then a set

$$\mathcal{N}(\mathcal{I}(\pi)) = \{\pi_l^t : \; i_l^t \in \mathcal{I}(\pi)\}$$

is neighborhood of permutation $\pi \in \Phi$.

The number of elements of the neighborhood (cardinality of the set $\mathcal{N}(\mathcal{I}(\pi))$) generated by *i*-moves is $(n-1)^2$. In tabu search algorithm there will be subneighborhoods used which are based on block elimination properties. As computational experiments have shown, their application results in a significant reduction of calculation time (further acceleration can be achieved with the use of multi-moves (Bożejko & Wodecki, 2007)).

For definiteness, in the following part of this chapter we assume that the maximum peak is on *k*-th machine. Therefore, we omit the index of the machine. For any block $B^r = (\pi(a^r), \pi(a^{r+1}), \ldots, \pi(b^r))$ $r = 1, 2, \ldots, s$ of break $\mathcal{B}$ of permutation π, we consider a set of *i-internal moves* $\mathcal{I}^{in}(B^r)$, which swap only the order of elements in this block, i.e. on positions $a^r+1, a^r+2, \ldots, b^r-1$. Then

$$\mathcal{I}^{in}(\pi) = \bigcup_{r=1}^{s} \mathcal{I}^{in}(B^r),$$

is a set of all the internal moves in permutation π. Directly from Theorem 3 stems the possibility of eliminating the moves which generate such elements of the neighborhood that are not better than π.

Property 1. *If permutation π_l^t is generated from π by performing internal move $i_l^t \in \mathcal{I}^{in}(\pi)$, then $T(\pi_l^t) \geqslant T(\pi)$.*

Thus, taking into account the possibility of a direct improvement of minimum cycle time $T^*(\pi)$, we can skip all elements of neighborhood generated by the moves from the set $\mathcal{I}^{in}(\pi)$.

For *r*-th block

$$B^r = (\pi(a^r), \pi(a^r+1), \ldots, \pi(b^k)), \; r = 1, 2, \ldots, s,$$

in permutation π we define sets of tasks of **candidates** to be swapped in the frames of *i*-move

$$\mathcal{J}_{af}^r = \{\pi(1), \ldots, \pi(a^r-1), \pi(b^r+2), \pi(b^r+3), \ldots, \pi(n)\},$$
$$\mathcal{J}_{bf}^r = \{\pi(1), \ldots, \pi(a^r-3), \pi(a^r-2), \pi(b^r+1), \pi(b^r+2), \ldots, \pi(n)\}.$$

These sets contain elements that will be swapped 'behind' or 'before' appropriate block, i.e., behind the last or before the first element of the block. On this basis, we define a set of *i*-moves behind block

$$\mathcal{I}_{af}^r(\pi) = \{i_l^x : \; \pi(x) \in \mathcal{J}_{af}^r\},$$

and a set of moves before the block

$$\mathcal{I}_{bf}^r(\pi) = \{i_t^x : \; \pi(x) \in \mathcal{J}_{bf}^r\}.$$

Certainly moves of the sets $\mathcal{I}_{af}^r(\pi)$ and $\mathcal{I}_{bf}^r(\pi)$ **can** (but not necessarily) bring improvement in the present value of the minimum cycle time.

On the basis of the presented considerations in the tabu search algorithm there will be used a set of moves

$$\mathcal{I}(\pi) = \bigcup_{r=1}^{s} [\mathcal{I}_{af}^r(\pi) \cup \mathcal{I}_{bf}^r(\pi)], \tag{16}$$

generating subneighborhood $\mathcal{N}(\mathcal{I}(\pi))$ of permutation π. The size of this neighborhood does not depend directly on the number of tasks n and m machines.

Example 2. The first element of the block $B^3 = (\pi(3), \pi(4), \pi(5))$ takes position $a^3 = 3$ in the permutation π, the last element takes position $b^3 = 5$. Hence the set of tasks – candidates to move directly **after** the block B^3

$$\mathcal{J}_{af}^3 = \{\pi(1), \pi(2), \pi(7), \pi(8), \pi(9), \pi(10)\},$$

and **before** the block

$$\mathcal{J}_{bf}^3 = \{\pi(1), \pi(6), \pi(7), \pi(8), \pi(9), \pi(10)\}.$$

Corresponding moves sets take the form of

$$\mathcal{I}_{af}^3 = \{i_2^{\pi(1)}, i_2^{\pi(2)}, i_2^{\pi(7)}, i_2^{\pi(8)}, i_2^{\pi(9)}, i_2^{\pi(10)}\}$$

for moves **after** the block, and

$$\mathcal{I}_{bf}^3 = \{i_6^{\pi(1)}, i_6^{\pi(6)}, i_6^{\pi(7)}, i_6^{\pi(8)}, i_6^{\pi(9)}, i_6^{\pi(10)}\}$$

for moves **befor** the block. Similarly, one can define sets of tasks and moves for other blocks. □

Let us consider the description of the methods of determining the tasks that should be moved behind the last $\pi(b^r)$ (or before the first $\pi(a^r)$) element of *r*-th block. According to the strategy of neighborhood search in tabu search method, the search concerns such a move $i_l^x \in \mathcal{I}(\pi))$ that generates a permutation π_l^x of the smallest possible value of the dominant peak (i.e. current minimum value of the cycle time).

To simplify the notation, the following assumptions were made:

(i) current permutation $\pi(i) = (1, 2, 3, \ldots, n)$,
(ii) we consider *r*-th block $B = (a, a+1, \ldots, b-1, b)$.

In accordance with previously adopted assumption the indices of machines (e.g. marking setup times) are omitted.

For any move $i_l^v \in \mathcal{I}_{af}(\pi)$ of B block designation

$$\Delta_{af}(v, l) = s_{v-1,v+1} - s_{v-1,v} - s_{v,v+1} + s_{l-1,v} + s_{v,l} - s_{l-1,l} \text{ is introduced.} \tag{17}$$

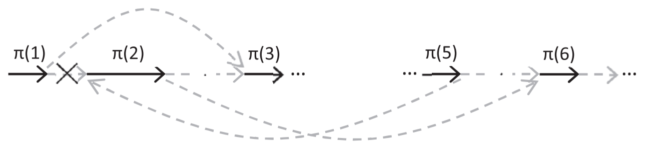

Fig. 6. Execution of the move $i_6^{\pi(2)}$.

Similarly, for move $i_l^v \in \mathcal{I}_{bf}(\pi)$

$$\Delta_{bf}(v,l) = s_{v-1,v+1} - s_{v-1,v} - s_{v,v+1} + s_{l-1,v} + s_{v,l} - s_{l-1,l}. \tag{18}$$

The computational complexity of determining the value of $\Delta_\lambda(v,l),\ \lambda \in \{af, bf\}$ is $O(1)$. The next two theorems show the relationships between the value of the peak of permutation π and permutations of the neighborhood generated by the move of a set $\mathcal{I}(\pi)$ defined in (16).

Theorem 4. *If permutation π_v^l was generated from $\pi \in \Phi$ by making a move $i_v^l \in \mathcal{I}_{af}(\pi)$, then the value of peak is*

$$T(\pi_l^v) = T(\pi) + \Delta_{af}(v,l).$$

Proof. It is enough to use the definition of peak (7) and swapping (17). □

Similarly, the moves, to be more specific – permutations – generated by these moves from the set $\mathcal{I}_{bf}(\pi)$ are estimated.

Example 3. Executing a move $i_6^{\pi(2)} \in \mathcal{I}_{af}^3$ (i.e., moving a task $\pi(2) \in \mathcal{J}_{af}^2$ in position 6-th in the permutation π) (directly after the block B^3) one can generate the new permutation $i_6^{\pi(2)}(\pi) = \pi_6^{\pi(2)}$ from π. This is symbolically shown in Fig. 6.

Taking advantage of Theorem 4 it is possible to calculate the value of the expression (17)

$$\begin{aligned}\Delta_{af}(\pi(2),6) &= s_{\pi(1),\pi(3)} - s_{\pi(1),\pi(2)} - s_{\pi(2),\pi(3)} + s_{\pi(2),\pi(6)} + s_{\pi(5),\pi(2)} \\ &\quad - s_{\pi(5),\pi(6)} \\ &= 1 - 1 - 2 + 1 + 1 - 2 = -2.\end{aligned}$$

Therefore, the value of a peak for the generated permutation

$$T_k(\pi_6^{\pi(2)}) = T_k(\pi) + \Delta_{af}(\pi(2),6) = 25 - 2 = 23.$$

So, the execution of the move $i_6^{\pi(2)}$ generates a permutation with lower peak value for the k-th machine. □

Theorem 5. *If permutation π_v^l was generated from $\pi \in \Phi$ by making a move $i_v^l \in \mathcal{I}_{bf}(\pi)$, then the value of peak is*

$$T(\pi_l^v) = T(\pi) + \Delta_{bf}(v,l).$$

Proof. Just use the definition of peak (7) and swapping (18). □

It is easy to see that with the use of Theorem 4 or 5) the maximum peak value for any machine generated by i-move of permutations can be determined.

By swapping the task $\pi(x) \in \mathcal{J}_{af}$ (or $\pi(x) \in \mathcal{J}bf$) behind the last element of block b (or before the first element a) in π, a permutation π_b^x (or π_a^x) is generated, for which we can determine (using respectively from Theorem 4 or 5) peak values on each machine. Thus, the values $\Delta_\lambda(x),\ \lambda \in \{af, bf\}$ can be used as a criterion for the selection from the neighborhood of the task to be swapped. Generally, a move with a minimum value of $\Delta_\lambda(v,l)$. is selected. In this way a permutation maximally reducing the peak value is generated.

Generated permutation (element of neighborhood) has the smallest cycle time (i.e., the least dominant peak).

4.2. Pattern designation

For the determination of the blocks on k-th machine there is a pattern required. Its designation comes down to solving the problem of traveling salesman in a graph H_k. The solution to this problem is *NP-hard*. Therefore, as the first there will be Algorithm 4.2-*opt* applied, one of the most popular approximate algorithms solving the traveling salesman problem.

Algorithm. 2-opt

```
Step 1:
    Determine starting solution (any Hamiltonian cycle);
Step 2:
    Check if there is a pair of edges, where the exchange for
 another
       generates Hamiltonian cycle of shorter length.
    If a pair of edges exists, then designate new (shorter)
       Hamiltonian cycle;
As long as it is possible, perform Step 2.
```

On the basis of the analysis of various approximation algorithms it was stated that the mean relative error of this algorithm, in reference to the best currently known solutions is about 5%. Algorithm has complexity of $O(n^2)$.

As a result of application of an approximation algorithm to solve traveling salesman problem assumptions of Lemma 3 cannot be met. In consequence, from the neighborhood of permutation $\pi \in \Phi$ 'good' elements (i.e, giving an improvement of the minimum cycle time) may be eliminated. The disadvantage of this problem can be partially improved by using, at a small number of vertices, the exact algorithm to solve traveling salesman problem, or even better – approximate algorithm (e.g., 3-opt, nevertheless, with more complex computation).

5. Computational experiments

There were computational experiments carried out whose aim was to assess the effectiveness of the modified tabu search algorithm from work (Nowicki & Smutnicki, 1996). The study was performed in two phases. In the first phase the effect of block properties on the efficiency of the algorithm was tested, whereas in the second – execution time of algorithm for practical applications.

The algorithm was implemented in C++ language in Visual Studio 2005 and was tested on a Compaq 8510w Mobile Workstation PC computer with Intel Core 2 Duo 2.60 GHz operating Microsoft VISTA system.

The calculations were performed on 960 examples for the hybrid flow-shop problem included in the work (Ruiz & Stützle, 2008). Gathered data was divided into 25 groups. Each consists of examples of the same number of tasks and machines. Individual groups differ from one another in the way of generation and/or the number of machines or tasks. Start solutions were set with the use of a simple construction algorithm based on the method of priority scheduling. The algorithm's operation can be divided into two phases. Firstly, for each slot there is a list of ordered tasks determined, whereas in the second phase the list is divided into fragments designating the sequence of tasks performance on each machine. Construction of the list in the first phase starts with any task. In the following step, there is one task requiring the shortest setup added to the list. Finally, a preliminary order of tasks on machines is obtained by assigning of each list to the first machine in the correct slot. In each iteration of the second phase there is the critical machine designated, i.e. a machine which determines the cycle time. The last operation performed on this machine is moved to the first position of the next machine in the same slot. The process is continued until the cycle time is shortened.

As a measure of the algorithm's A efficiency there was a Percentage Relative Deviation (PRD) of a goal function value assumed for the best solution π^A determined the algorithm in relation to the reference solution π^{ref}:

$$PRD(\pi^A) = 100\%(T(\pi^A) - T(\pi^{ref}))/T(\pi^{ref}). \tag{19}$$

The algorithm from work (Nowicki & Smutnicki, 1996) was implemented in two versions: TSF – with a full neighborhood, generated by the moves from the set $\mathcal{H}$ and TSB – with a reduced environment, generated through the moves from $\mathcal{N}$.

There were calculations (of TSF and TSB algorithms) performed for the length $L = 7$ of the tabu list and 1000 iterations with neighborhood of insert moves. For each example, as the reference solution, there was the best found solution adopted. The obtained comparative results are shown in Tables 1–3.

For comparison, a canonical Simulated Annealing (SA) algorithm was implemented. The following parameters of SA algorithm have been used: n^2 iterations with fixed temperature, neighborhood of insert moves, harmonic cooling scheme. Table 1 presents results for a fixed number of iterations (which equals 1000) of the proposed algorithms TSF and TSB compared with SA algorithm. Starting solutions were determined with the use of NEH (Nawaz, Enscore, & Ham, 1983) algorithm. The following notions were used:

- t – Execution time of a TSF algorithm without blocks.
- t_B – Execution time of a TSB algorithm with blocks.

Table 1
Computational time and Percentage Relative Deviation for fixed number of iterations (1000).

$n \times m$	t (s)	t_B (s)	t_{SA}	PRD	PRD_B	PRD_{SA}
20 × 5	2.24	0.80	0.73	−29.96	−32.74	−31.68
20 × 10	3.01	0.92	1.21	−29.49	−31.65	−30.79
20 × 20	4.67	1.18	2.29	−29.77	−30.68	−30.34
50 × 5	35.72	17.68	8.85	−32.49	−34.08	−35.12
50 × 10	48.42	21.96	17.16	−29.56	−34.48	−33.02
50 × 20	73.87	28.28	33.59	−28.34	−31.76	−30.79
100 × 5	292.08	181.84	67.79	−30.73	−36.71	−34.90
100 × 10	391.91	203.82	134.73	−28.10	−33.59	−31.88
100 × 20	604.70	266.44	273.44	−27.87	−31.63	−30.62
200 × 10	3212.17	1791.06	1091.53	−26.70	−32.95	−31.49
200 × 20	5255.53	2497.66	2645.24	−26.21	−31.11	−30.10
Average	902.21	455.60	388.78	−29.02	−32.85	−31.88

Table 2
The Percentage Relative Deviation for a fixed computations time.

$n \times m$	PRD	PRD_B	PRD_B^*	PRD_{SA}
20 × 5	−29.96	−32.71	−32.71	−31.47
20 × 10	−29.49	−31.81	−31.81	−30.63
20 × 20	−29.77	−31.04	−31.11	−30.37
50 × 5	−32.49	−34.40	−34.40	−35.33
50 × 10	−29.56	−34.67	−34.70	−33.15
50 × 20	−28.34	−31.97	−31.99	−30.95
100 × 5	−30.73	−37.15	−37.10	−34.91
100 × 10	−28.10	−33.96	−34.00	−32.11
100 × 20	−27.87	−31.98	−32.02	−30.60
200 × 10	−26.70	−33.39	−33.42	−31.79
200 × 20	−26.21	−31.39	−31.48	−30.13
Average	−29.02	−33.13	−33.16	−31.95

Table 3
Computational time for fixed cost function value.

$n \times m$	t (s)	t_B (s)	t^* (s)	t_B^* (s)	t_{SA}
20 × 5	2.24	4.24	1.90	5.33	20.74
20 × 10	3.01	0.58	2.44	0.24	36.07
20 × 20	4.67	0.23	3.53	0.28	67.36
50 × 5	35.72	66.68	28.29	68.53	2.72
50 × 10	48.42	0.80	34.27	0.60	173.39
50 × 20	73.87	0.90	46.86	0.56	10.17
100 × 5	292.08	10.16	220.26	7.72	30.36
100 × 10	391.91	11.10	266.12	7.23	57.92
100 × 20	604.70	12.86	358.37	6.72	115.62
200 × 10	3212.17	141.63	2067.67	85.28	678.03
200 × 20	5255.53	186.79	2754.64	79.03	1591.29
Average	902.21	39.63	525.85	23.77	253.06

- PRD – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for an algorithm without blocks, fixed number of iterations.
- PRD_B – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for an algorithm with blocks, fixed number of iterations.
- PRD_B – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for an algorithm with blocks, fixed number of iterations.
- PRD_{SA} – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for a Simulated Annealing algorithm, fixed number of iterations.

As it can be observed, the results of TSB algorithm (with blocks) outperforms TSF algorithm in both time and solution quality issues. The TSB algorithm is almost two times faster than TSF (902.21 vs. 455.60 s in average) and it obtains much better results (−29.02 vs. −32.85 of average improvement starting NEH solutions).

The t-Student test of statistical significance shows that the average value PRD_B of the TSB algorithm is significantly lower than the PRD_{SA} with the standard significance level $\alpha = 0.05 : H_0 : m_1 = m_2$, $H_1 : m_1 < m_2$, where m_1, m_2 denote a PRD of TSB and SA algorithms, respectively, for any set of test instances. The value of test statistic equals to

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{n_1 s_1^2 + n_2 s_2^2}{n+1+n_2-2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} = -12.03,$$

where $n_1 = n_2 = 960$, $\bar{x}_1 = -32.85$, $\bar{x}_2 = -31.88$, $s_1^2 = 3.15$, $s_2^2 = 3.06$. The critical set for $\alpha = 0.05$ is $(\infty, -1,65]$ (from the normal distribution); the value of test statistic belongs to the critical set, so H_0 is rejected and the hypothesis H_1 is taken which says that the PRD_B of the TSB algorithm is significantly lower than PRD_{SA}.

Table 2 presents results for a fixed computational time (equals computation time for 1,000 iterations of the algorithm without blocks). There following notions were used:

- PRD – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for an algorithm without blocks, 1000 iterations (chosen as a reference time – stop criterion for further algorithms compared in Table 2).
- PRD_B – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for an algorithm with blocks, fixed computation time.
- PRD_B^* – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for an algorithm with blocks, fixed computation time, with acceleration of the goal function calculation.

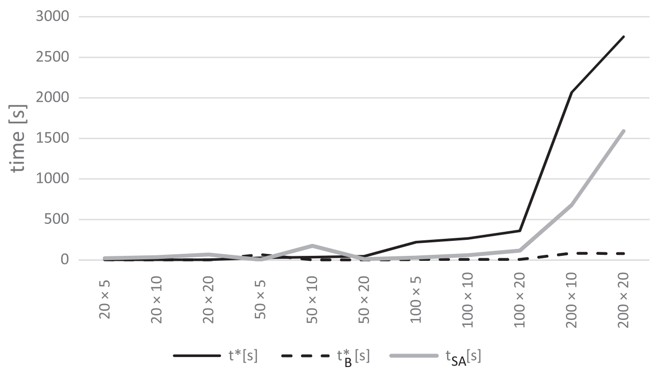

Fig. 7. Comparison of TSF and TSB algorithms computing times with accelerator of goal function calculation.

- PRD_{SA} – Percentage Relative Deviation to the starting solutions determined by the NEH algorithm, for a Simulated Annealing algorithm, for a fixed computation time.

As it can be observed, application of blocks gives additional 4.11% (-29.02% to -33.13%, in average) improvement of the PRD of the TSB, comparing to TSF, considering fixed time of computations. Applying a goal function calculation accelerator gives additional 0.03% improvement of PRD (-33.13% to -33.16%). As in previous, the *t*-Student test of statistical significance shows that the average value of PRD_B of the TSB algorithm is significantly lower than the PRD_{SA} for a fixed time of computations, with the significance level $\alpha = 0.05$. For a test $H_0 : m_1 = m_2,\ H_1 : m_1 < m_2$, where $m_1,\ m_2$ denote a *PRD* of TSB and SA algorithms achieved after a fixed time of computations, the value of test statistic equals to $t == -14.78$, where $n_1 = n_2 = 960,\ \bar{x}_1 = -33.16,\ \bar{x}_2 = -31.95,\ s_1^2 = 3.19,\ s_2^2 = 3.23$. For the critical set $(\infty, -1, 65]$ we reject H_0 and take the hypothesis H_1 – the PRD_B of the TSB algorithm is significantly lower than PRD_{SA} for a fixed computations time.

Table 3 presents time comparison for algorithms with fixed solution quality – each algorithm stops after achieving (or exceeding) assumed goal function value of the TSF algorithm with stop criterion of 1,000 iterations. The following notions were used:

- t – execution time of a TSF algorithm without blocks,
- t_B – execution time of a TSB algorithm with blocks,
- t^* – execution time of a TSF algorithm without blocks, with acceleration of the goal function calculation,
- t_B^* – execution time of a TSB algorithm with blocks, with acceleration of the goal function calculation.
- t_{SA} – execution time of a SA algorithm.

It is visible, that application of blocks gives over 20-times shortening of the computation time (902.21 s of TSF to 39.63 s of TSB, in average), taking under consideration computation time needed to obtain a fixed quality of solutions. Similar situation appears after using accelerator of the goal function calculation. Simulated Annealing method needs the time which is over 10 times longer than the time of TSB to obtain solutions with comparable level of *PRD*. Fig. 7 show differences of computation times for each group of benchmark instances.

6. Conclusions

The new block elimination criteria for the cyclic flow shop problem were proposed in this paper. We designed genuine problem properties which enable obtaining good efficiency of the local search algorithm. Computational experiments show both shortening of the computation time and improving of the quality of the obtained solutions, as a result of application of the block properties to the tabu search metaheuristic. The advantage is especially visible for large instances of the considered cyclic flow shop scheduling problem.

Acknowledgements

The work was partially supported by the OPUS Grant DEC-2012/05/B/ST7/00102 of Polish National Centre of Science. Calculations have been carried out using resources provided by Wrocław Centre for Networking and Supercomputing (http://wcss.pl), Grant No. 96.

References

Bożejko, W., Gniewkowski, Ł., Pempera, J., & Wodecki, M. (2014). Cyclic hybrid flow-shop scheduling problem with machine setups. *Procedia Computer Science, 29*, 2127–2136.
Bożejko, W., & Wodecki, M. (2007). On the theoretical properties of swap multimoves. *Operations Research Letters, 35*(2), 227–231.
Bożejko, W., & Wodecki, M. (2008). Parallel scatter search algorithm for the flow shop sequencing problem. *Lecture Notes in Computer Science, 4967*, 180–188.
Caggiano, K., & Jackson, P. L. (2008). Finding minimum flow time cyclic schedules for non-identical, multistage jobs. *IIE Transactions, 40*, 45–65.
Cavory, G., Dupas, R., & Goncalves, G. (2005). A genetic approach to solving the problem of cyclic job shop scheduling with linear constraints. *European Journal of Operational Research, 161*, 73–85.
Dbrowski, P., Pempera, J., & Smutnicki, C. (2007). Minimizing cycle time of the flow line-genetic approach with gene expression. *Lecture Notes in Computer Science, 4431*, 194–201.
Fournier, O., Lopez, P., & Lan Sun Luk, J.D. (2002). Cyclic scheduling following the social behavior of ant colonies. In *IEEE international conference publications systems, man and cybernetics*.
Grabowski, J., & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers and Operations Research, 31*, 1891–1909.
Kampmeyer, T. (2006). Cyclic scheduling problems. Ph.D. Dissertation, Universitat Osnabruck.
Levner, E., Kats, V., Lopez, A. P., & Cheng, T. C. E. (2010). Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers and Industrial Engineering, 59*, 352–361.
Nawaz, M., Enscore, E. E., Jr., & Ham, I. (1983). A heuristic algorithm for the *m*-machine *n*-job flowshop sequencing problem. *OMEGA International Journal of Management Science, 11*, 91–95.
Nowicki, E., & Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research, 91*, 160–175.
Panwalkar, S. S., Dudek, R. A., & Smith, M. L. (1973). Sequencing research and the industrial scheduling problem. In S. E. Elmaghraby (Ed.), *Symposium on the theory of scheduling and its applications*. Berlin: Springer-Verlag.
Ruiz, R., & Stützle, T. (2008). An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research, 187*(3), 1143–1159.
Sawik, T. (2012). Batch vs. cyclic scheduling in flexible flow shops by mixed integer programming. *International Journal of Production Research, 50*(18), 5017–5034.
Sawik, T. (2014). A mixed integer program for cyclic scheduling of flexible flow lines. *Bulletin of the Polish Academy of Sciences, Technical Sciences, 62*(1), 121–128.
Wodecki, M. (2009). A block approach to earliness–tardiness scheduling problems. *International Journal on Advanced Manufacturing Technology, 40*, 797–807.