BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 68, No. 1, 2020
DOI: 10.24425/bpasts.2020.131829

Blocks for two-machines total weighted tardiness flow shop scheduling problem

W. BOŻEJKO, M. UCHROŃSKI, and M. WODECKI

[1] Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wyb. Wyspianskiego 27, 50-370 Wrocław, Poland
[2] Department of Telecommunications and Teleinformatics, Faculty of Electronics, Wrocław University of Science and Technology, Wyb. Wyspianskiego 27, 50-370 Wrocław, Poland

Abstract. The paper discusses a two-machine flow shop problem with minimization of the sum of tardiness costs, being a generalization of the popular NP-hard single-machine problem with this criterion. We propose the introduction of new elimination block properties allowing for accelerating the operation of approximate algorithms of local searches, solving this problem and improving the quality of solutions determined by them.

Key words: flow shop, two machine, due date, minimal costs, blocks of tasks, parallel algorithm.

List of main symbols

$\mathscr{J}$	set of tasks
$\mathscr{M}$	set of machines
n	number of tasks
π	permutation defining tasks order
T_i	tardiness of a task i
w_i	weight of tardiness of a task i
d_i	demanded completion time (due date)
$O_{i,k}$	operation of task i on machine k
$p_{i,k}$	duration of an operation $O_{i,k}$
$S_{i,j}$	starting time of an operation $O_{i,j}$
$C_{i,j}$	finishing time of an operation $O_{i,j}$
$\mathscr{C}_i$	finishing time of a task i

1. Introduction

In the problem called 'total tardiness minimization in two machine flow shop problem' (denoted by $F2||\sum w_iT_i$ according to Graham's notation [13]), each of the n tasks should be executed sequentially on the first and then on the second machine. There are times of task completion and the due dates of their completion (on the second machine) given. Exceed-ing this due dates results in a penalty, which depends on the amount of delay and a fixed penalty rate. It is necessary to determine the order of performing tasks (the same on both machines), which minimizes the sum of penalties. This problem will be denoted by T2FS in brief. It is a generalization of NP-hard, single-machine problem of scheduling tasks with minimization of the sum of penalties for tardiness ($1||\sum w_iT_i$). A detailed description of the problem, its specific properties and a very effective algorithm based on the tabu search method were presented in the work by Bożejko, Grabowski and Wodecki [5]. The properties shown there are a certain development of the ideas contained in Wodecki's works [35] and [36] regarding multi-machine problems with all-cost goal functions. The two-machine flow shop problem with criterion $C_{\max}$ (i.e., minimizing the end of execution for all tasks, $F2||C_{\max}$) is a polynomial problem (Johnson's algorithm [17]). However, the problem with the sum-cost criterion $\sum w_iT_i$ belongs to *NP-hard* class of problems, Lenstra et al. [22]. In Schaller's work [28] a two-machine flow shop problem with minimization of the sum of total tardiness, $F2||\sum T_i$ was presented. In it, there were five elimination properties given: '*if there are some dependencies between the l and k task parameters, then in the optimal solution the task l is before the task k*'. Due to them, by searching the neighborhood, one can omit many suboptimal solutions. In the work [28], using the Koulamus results [20], there was presented the idea of a proof of a strong NP-hardness of the considered in this work problem $F2||\sum w_iT_i$. In turn, Lee and Kim [21] consider a two-machine flow shop problem $F2||\sum T_i$ with an additional constraint concerning the availability of tasks on the first machine. Metaheuristic algorithms for this problem are presented in the work by Ta et al. [30]. Furthermore, optimal algorithms (based on the B&B scheme) for different variants of the tasks scheduling problem on two machines with sum criteria are described in the works: Bank et al. [4], Moukrim et al. [24], as well as Hamdi et al. [15]. As the authors write, one can solve examples with a number of tasks not exceeding 50 in a reasonable time. There are many more works devoted to approximate algorithms: construction, metaheuristic ones and their hybrids. Especially interesting seem to be the

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

Manuscript submitted 2019-08-02, revised 2019-09-16, initially accepted for publication 2019-10-18, published in February 2020

works that have appeared in recent years: Ahmadi et al. [1], Cheng et al. [10], as well as Ardakan et al. [3] and Bożejko et al. [7]. There are also new, promising methods of local search, using new neighborhoods, such as the golf one (e.g. Bożejko et al. [8]).

The problem of tasks scheduling on two machines in the Just-in-Time system is considered in the work of Al-Salema et al. [2]. There is presented an algorithm based on the dynamic programming method. For small examples, this is an optimal algorithm. However, for larger examples there are some cut-offs causing the algorithm to work much faster, yet the designated solutions are only suboptimal. In turn, Kharbeche and Haouari [19] used the discrete integer programming method to solve this problem.

There are relatively few publications devoted exclusively to the problem considered in this work and methods of solving it. Some theoretical results and good approximation algorithms are presented in the works: Gupta and Harari [14], Lina [23] and Bulfin and Hallaha [9]. In turn, Khalili et al. [18] consider the multicriteria optimization problem with two goal functions: maximum completion time of tasks – makespan) and the sum of the total weighted tardiness).

Problems of tasks scheduling on a single or two machines with sum-cost goal functions have very long, over 50 years of history. Despite the simplicity of formulation, they mostly belong to the class of NP-hard problems. They are important both from the point of view of theory and practice. Their various variants are still intensively studied, and the results obtained are also an inspiration for research on much more complex multi-machine problems. In many practical applications, a single or two-machine cell is an important part of more complex production systems and constitutes a "bottleneck" of the system. In this case, production scheduling boils down to the optimal use of these machines. For example, in the production process of bicycle frames, the cell is formed by two different presses working in a flow system. Some elements of each frame are pressed on the first and then on the second press. When stamping various types of frames, specific templates are used. There is therefore a need for frequent replacements. Hence, stamping operations are very time consuming compared to other operations and determine the efficiency of the entire system. Due to the size, weight and the cost of purchase and operation, it is not possible to increase the number of presses. According to the order, a certain batch of frames should be made in advance. The production scheduling process has been divided into two stages:

1) optimization of the presses cell,
2) optimizing the work of other cells including the press schedule.

Both issues are NP-hard, so metaheuristic algorithms were used to solve them. Despite this, better results were obtained than when the entire production process was optimized simultaneously.

In the further part of the work we present a detailed description of the considered problem $F2||\sum w_iT_i$ and its mathematical model. We prove the specific properties, namely – the so-called block elimination properties, which are a generalization of ideas included in Bożejko et al. [5], which significantly improve efficiency of algorithms for solving multi-machine cost problems. In case of their use in constructions of algorithms based on the local search method there was observed a significant reduction in the number of elements of generated in each iteration neighborhood. Due to this fact, better results were obtained in shorter time of the calculations.

2. Description of the problem and its mathematical model

Two-machine permutation flow shop problem with minimization of the sum the tardiness costs can be formulated as follows:

Problem 1. There is a set of tasks $\mathscr{J} = \{1,2,\ldots,n\}$ and the set of machines $\mathscr{M} = \{1,2\}$ given. The task $i \in \mathscr{J}$ consists of two operations $O_{i,1}$, $O_{i,2}$. Operation $O_{i,k}$ corresponds to execution of task i on machine $k \in \mathscr{M}$. For task $i \in \mathscr{J}$, let d_i be the *demanded completion time (due date)*, w_i *coefficient of penalty function*, whereas $p_{i,k}$ *time of execution* of operation $O_{i,k}$. It is necessary to perform tasks on the machines, wherein the following restrictions must be fulfilled:

(a) each task should be performed on the first one and then on the second machine,
(b) the task execution cannot be interrupted,
(c) the task can be performed simultaneously on only one machine,
(d) the machine cannot perform more than one task at the same time,
(e) the order of performing tasks, on both machines must be the same.

For the determined order of performing tasks on machines, let $S_{i,j}$ be the moment of starting the operation $O_{i,j}$ $(i \in \mathscr{J}, j = 1,2)$. It follows from restrictions (b) and (c) that $C_{i,j} = S_{i,j} + p_{i,j}$ is the time the completion of operation O_{ij}. These moments can be determined from the following recursive dependencies:

$$C_{i,j} = \max\left\{C_{i-1,j},\, C_{i,j-1}\right\} + p_{i,j}\,, \qquad i = 1,2,\ \ldots\ .n, \quad j = 1,2, \tag{1}$$

with initial conditions:

$$C_{0,j} = 0, \quad j = 1,2 \quad \text{and} \quad C_{i,0} = 0, \quad i = 1,2,\ \ldots\ ,n. \tag{2}$$

By $\mathscr{C}_i = C_{i2}$ we denote the date of completion of task i, i.e. operation O_{i2}. Then

$$T_i = \max\{0, \mathscr{C}_i - d_i\} \tag{3}$$

is *tardiness* for completing the task i, $f_i = w_i \cdot T_i$ *penalty* for tardiness (in other words – *cost of task execution*). If $T_i = 0$, the task is called *early*, otherwise – *tardy*.

Each solution, i.e. the order of performing tasks (the same on both machines) can be represented by the permutation of tasks from the set $\mathscr{J}$. Let Π be a set of all such permutations.

For a permutation

$$\pi = (\pi(1), \ldots, \pi(n))$$

penalty for tardiness of tasks execution (in short *solution cost*)

$$F(\pi) = \sum_{i=1}^{n} f_{\pi(i)} = \sum_{i=1}^{n} w_{\pi(i)} \cdot T_{\pi(i)}. \tag{4}$$

In the problem under consideration, there should be determined the order of performing tasks that minimize the penalty for tardiness, i.e. optimal permutation $\pi^* \in \Pi$, for which

$$F(\pi^*) = \min\{F(\pi) : \pi \in \Pi\}. \tag{5}$$

3. Johnson's algorithm

In the introduction we wrote that the two-machine flow shop problem with the $C_{\max}$ criterion belongs to the $\mathscr{P}$ class. In order to solve it there is Johnson's algorithm used [17] (see Algorithm 1).

Algorithm 1: Johnson's algorithm – determination of the optimal solution for the problem $F2||C_{\max}$

```
Input  : $n$ – number of tasks,
         $p_{i,j}$ – tasks time of execution, $i = 1,2,\ldots,n$,
         $j = 1,2$.
Output: $\pi$ – tasks permutation (order of tasks execution)
for $i \leftarrow 1,2,\ldots,n$ do
    $\lambda_i \leftarrow \min\{p_{i,1}, p_{i,2}\}$;
    sort elements $\lambda_i$ such that $\lambda_1 \le \lambda_2, \ldots, \lambda_n$;
    $mi \leftarrow 1$; $mx \leftarrow n$;
for $i \leftarrow 1,2,\ldots,n$ do
    if $\lambda_i = p_{i,1}$ then
        $\pi(mi) \leftarrow i$;
        $mi \leftarrow mi + 1$;
    else
        $\pi(mx) \leftarrow i$;
        $mx \leftarrow mx - 1$;
```

The computational complexity of Johnson's sequential algorithm is equal to $O(n \ln n)$. In further part of the work we will use this algorithm to determine optimal, due to the criterion $C_{\max}$, order of certain tasks. Since this element of the algorithm solving the problem will be run very often, in the next section we will present a proposal for parallelizing the Johnson's algorithm.

3.1. Parallel Johnson's algorithm.

In this section, a parallel version of the Johnson algorithm will be proposed to thoroughly solve the problem of $F2||C_{\max}$ a used in the further part of the work to optimally set tasks in the block.

Algorithm 2: Optimal parallel algorithm for the problem $F2||C_{\max}$ based on the idea of sequential Johnson's algorithm

```
Input  : $\alpha_i$ – time of execution of $i$-th task on the 1st
         machine;
         $\beta_i$ – time of execution of $i$-th task on the 2nd
         machine;
         $n$ – number of tasks;
Output: permutation $\pi$ – order of tasks execution;
1  $k := 1$; $l := n$; $\gamma_i := \varepsilon_i := 0$; $i = 1,2,\ldots,n$;
2  parfor $i = 1,2,\ldots,n$ do
3      $\delta_i = \min\{\alpha_i, \beta_i\}$;
4      if $\delta_i = \alpha_i$ then
5          $\gamma_i := \delta_i$;
6          $a(i) := i$; $b(i) := 0$;
7      else
8          $\varepsilon_i := \delta_i$;
9          $b(i) := i$; $a(i) := 0$;
10     end
11 end
12 Sort $(\gamma_i, a(i))$ pairs in parallel, in non decreasing order due
     to the first element of a pair element;
13 $x$ := n;
14 parfor $i = 1,2,\ldots,n-1$ do
15     if $a(i) = 0$ and $a(i+1) \neq 0$ then
16         $x := i$; (number of non-negative $\varepsilon_i$)
17     end
18 end
19 Sort $(\varepsilon_i, b(i))$ pairs in parallel, in non decreasing order due
     to the first element of a pair element;
20 $y = n - x$; (number of non-negative $\gamma_i$);
21 parfor $i = 1,2,\ldots,n$ do
22     if $i \le n - x$ then
23         $\pi(i) = a(i+x)$;
24     else
25         $\pi(i) := b(n-(i-y)+1)$;
26     end
27 end
```

Theorem 1. Parallel Johnson's algorithm (Algorithm 2) can be run in time $O(\log n)$ on n-processor EREW PRAM.

Proof. Both substitutions in line 1 and a parallel loop in lines 2–11 (Algorithm 2) can be performed in a fixed time $O(1)$ on n processors. Sorting in lines 12 and 14 of n element sequence can be done with the parallel algorithm of *mergesort* in the time $O(\log n)$ on the n – processor EREW PRAM [11]. Searching for the first non-zero position in the array a can be done in a fixed time (loop in lines 14–18) by checking in parallel by the processors $i = 1,2,\ldots,n-1$, whether in the corresponding cells $a(i)$ there are zero and the next $a(i+1)$ – if they are different from zero. There is only one such position i and it is the equivalent of x denoting the number of zeros in the array a.

The substitution in lines 13 and 20 is performed in a constant time on one processor. The parallel loop in lines 20–26 will be performed in a fixed time using n processors (without the need for simultaneous reading) which completes the proof. □

The parallel implementation of the Johnson's algorithm will be used in the procedure of determining blocks of tasks in permutation.

4. Properties of the problem

To solve the considered *NP*-hard scheduling problem there will be algorithms based on the local search method used. The essential element of this method, having decisive influence not only on the time of calculations but also on the values of determined solutions, is a procedure for generating and searching the neighborhood.

In many metaheuristic algorithms solving the flow shop problem with minimizing the date for completion of performing tasks (i.e. $F||C_{\max}$) there are neighborhoods used generated by *insert* type of moves, in short called *i–moves*. If $\pi \in \Pi$, then *i–move* i_l^k $(1 \le k, l \le n)$ generates from π the new permutation $i_l^k(\pi) = \pi_l^k$ by swapping the element $\pi(k)$ to the position l. The neighborhood generated by these moves has $n(n-1)$ elements. These moves and generated by them neighborhoods are described precisely in the work of Bożejko et al. [5].

In the best metaheuristic algorithms for solving the problem $F||C_{\max}$ there are the so-called 'block elimination properties' used (Nowicki and Smutnicki [26], Grabowski and Wodecki [12]). Due to them, in the neighborhood search procedure, one can omit many of the worse solutions. As demonstrated by computational experiments, due to this procedure, not only the neighborhood search time is shortened, but also better solutions are obtained. In the further part of this section there will be similar properties introduced allowing us for elimination, through indirect review, of many solutions. Weakening or omitting some limitations in the definition of classic blocks (used in algorithms for solving cost problems, Bożejko et al. [5], Wodecki [35,36], there is a new concept of a 'weaker' block introduced, the so-called *semi-block*.

Any sequence of immediately appearing one after another elements in permutation π will be called *subpermutation*. If

$$\eta = (\pi(u), \pi(u+1), \ldots, \pi(v)), \quad 1 \le u \le v \le n,$$

is the subpermutation of the permutation π, then the cost of the execution of tasks from η

$$F_\pi(\eta) = \sum_{i=u}^{v} (w_{\eta(i)} \max\{0, \mathscr{C}_{\eta(i)} - d_{\eta(i)}\}), \tag{6}$$

where $\mathscr{C}_{\eta(i)}$ completion date of the execution of the task $\eta(i)$ in permutation π. By $\mathscr{Y}(\eta)$ we denote the set of elements of subpermutation η, i.e.

$$\mathscr{Y}(\eta) = \{\pi(u), \pi(u+1), \ldots, \pi(v)\}.$$

Let

$$\begin{gathered} \alpha = (1, 2, \ldots, a-1), \quad \beta = (a, a+1, \ldots, b-1, b), \\ \gamma = (b+1, b+2, \ldots, n), \end{gathered} \tag{7}$$

where $1 < a < b \le n$, will be some subpermutations in π, i.e.

$$\pi = (1, 2, \ldots, a-1, a, a+1, \ldots, b-1, b, b+1, \ldots, n). \tag{8}$$

Then, permutation $\pi = (\alpha, \beta, \gamma)$ is a sequence (concatenation) of three subpermutations, and its cost

$$F(\pi) = F_\pi(\alpha) + F_\pi(\beta) + F_\pi(\gamma). \tag{9}$$

4.1. Blocks of early tasks. Let permutation $\pi \in \Pi$ be a sequence of three subpermutations, i.e. $\pi = (\alpha, \beta, \gamma)$ defined in (7).

To the set of tasks from subpermutation β we use Johnson's algorithm (section 3, **Algorithm 1**). In this way there is a new order of tasks designated

$$\beta' = (a', a'+1, \ldots, b'-1, b'). \tag{10}$$

Subpermutation β' will be called *Johnson optimal*, in short *J-opt*. It is the optimal order due to the minimization of the due date completion of all tasks from β.

We are considering permutation $\pi = (\alpha, \beta, \gamma) \in \Pi$ and

$$\pi' = (\alpha, \beta', \gamma), \tag{11}$$

where β' is a subpermutation *J-opt*. It's easy to show that the moment of completion of the last task in β' is not bigger than the moment of completion of the last task in β. This fact will be used in the proof of the following theorem.

Theorem 2. If the permutation $\delta = (\alpha, \beta'', \gamma)$ was generated from π' (11) by swapping the order of tasks in *J-opt* of the subpermutation β', then the completion time of any task from γ (in permutation δ) is not smaller than the moment of completion of this task in permutation π'.

Proof. Let permutation $\pi = (\alpha, \beta', \gamma)$, where subpermutation β' jest *J-opt*. From π we generate permutation $\delta = (\alpha, \beta'', \gamma)$ by swapping the order of tasks in subpermutation β'. Therefore $\mathscr{Y}(\beta') = \mathscr{Y}(\beta'')$. Thus, β' and β'' are permutations of the same subset of the tasks set. By $b(\beta')$ and $b(\beta'')$ we denote the last element respectively in the subpermutation β' and β''. Further, let $l(\gamma)$ be the first element of subpermutation γ. It follows from the definition of the problem that the tasks on the first machine can be carried out directly one by one (i.e., can be pushed to 'left'). Therefore, the completion times for tasks $b(\beta')$ and $b(\beta'')$, on the first machine, are equal to each other, i.e.

$$C_{b(\beta'),1} = C_{b(\beta''),1}. \tag{12}$$

In addition, the completion times for the task $l(\gamma)$ on the first machine, in both permutations π and δ, are the same, and

$$C_{l(\gamma),1} = C_{b(\beta'),1} + p_{l(\gamma),1}. \tag{13}$$

We assumed that β' is subpermutation *J-opt*. This means that it is optimal (the order of tasks from the set $\mathscr{Y}(\beta')$) due to $C_{\max}$ criterion. In this case the value $C_{\max}$, is equal to the completion date of the task $b(\beta')$ on the second machine, i.e. $C_{b(\beta'),2}$. Since β'' is a certain permutation of elements of the same set $\mathscr{Y}(\beta')$ the completion date of the last task on the second machine is $C_{b(\beta''),2} \geq C_{b(\beta'),2}$. Therefore we proved that the completion times of the last task from β' and β'' meet the dependencies:

$$(C_{b(\beta'),1} = C_{b(\beta''),1}) \wedge (C_{b(\beta'),2} \leq C_{b(\beta''),2}). \tag{14}$$

Therefore, the starting date (and therefore the completion moment) of the task $l(\gamma)$ (the first one in the subpermutation γ) in the permutation δ is not less than in the permutation π', similarly on the second machine. Thus, we proved that in generated from π' permutation δ executing the $l(\gamma)$ task will not end sooner ('it will not move to the left'). It is easy to show that it is similar to the other tasks of the subpermutation γ, which ends the proof of the theorem. □

Definition 1. Let permutation $\pi' = (\alpha, \beta', \gamma)$, where β' be the supermutation *J-opt*. If all tasks in β' are early tasks, the β' subpermutation is called *block of early tasks* (in brief *T-block*).

Theorem 3. (Eliminating property of *T-block*)
If the permutation π' was generated from $\pi \in \Pi$ by swapping the order of tasks in some *T-block*, then

$$F(\pi') \geq F(\pi)$$

.

Proof. Let β be some *T-block* in permutation $\pi = (\alpha, \beta, \gamma)$ ($\pi \in \Pi$), and π' permutation generated from π by swapping the order of elements in β. Therefore, permutation $\pi' = (\alpha, \beta', \gamma)$, where β and β' are permutations of the same set of elements i.e. $\mathscr{Y}(\beta) = \mathscr{Y}(\beta')$. Let us assume that subpermuations:

$$\begin{gathered} \alpha = (1, 2, \ldots, a-1), \quad \beta = (a, a+1, \ldots, b), \\ \gamma = (b+1,\, b+2,\, \ldots, n). \end{gathered} \tag{15}$$

It follows from the definition of the cost function (4) and (6) that

$$\begin{aligned} T(\pi) &= T_\pi(\alpha) + T_\pi(\beta) + T_\pi(\gamma) = \\ &= \sum_{i=1}^{a-1} f_{\pi(i)} + \sum_{i=a}^{b} f_{\pi(i)} + \sum_{i=b+1}^{n} f_{\pi(i)}, \end{aligned} \tag{16}$$

and

$$\begin{aligned} T(\pi') &= T_{\pi'}(\alpha) + T_{\pi'}(\beta') + T_{\pi'}(\gamma) = \\ &= \sum_{i=1}^{a-1} f_{\pi'(i)} + \sum_{i=a}^{b} f_{\pi'(i)} + \sum_{i=b+1}^{n} f_{\pi'(i)}, \end{aligned} \tag{17}$$

where f_i is penalty function for tardiness of task i. We are considering successively the sums occurring in expressions (17) and (16). From definition of permutation π and π' the first sums are equal, i.e.

$$\sum_{i=1}^{a-1} f_{\pi(i)} = \sum_{i=1}^{a-1} f_{\pi'(i)}. \tag{18}$$

Since β is *T-block* (all tasks have due dates), therefore

$$\sum_{i=a}^{b} f_{\pi(i)} = 0. \tag{19}$$

Thus,

$$\sum_{i=a}^{b} f_{\pi'(i)} \geq \sum_{i=a}^{b} f_{\pi(i)}. \tag{20}$$

It follows from Theorem 2 that for $i = b+1, b+2, \ldots, n$ tasks completion moments $\pi(i)$ and $\pi'(i)$ satisfy inequality:

$$\mathscr{C}_{\pi'(i)} \geq \mathscr{C}_{\pi(i)}, \quad \text{for } i = 1, 2, \ldots, n.$$

It follows from the above that also inequalities $f_{\pi'(i)} \geq f_{\pi(i)}$, are fulfilled. By adding sides we receive

$$\sum_{i=b+1}^{n} f_{\pi'(i)} \geq \sum_{i=b+1}^{n} f_{\pi(i)}. \tag{21}$$

To sum up, it follows from relation (18)–(21) that $F(\pi') \geq F(\pi)$, which ends the proof of the theorem. □

Proposition 1. Generating some permutations from π one can omit those that were created by swapping the order of tasks in any *T-block*, since they do not improve the value of the criterion function (4).

Let $\pi = (\alpha, \beta, \gamma)$, $\pi \in \Pi$ and $\pi' = (\alpha, \beta', \gamma)$ where β' is supermutation *J-opt*. Let us assume that not all tasks in β' are due date tasks. Therefore, β' is not a *T-block*. We are considering the following conditions:
(A) $(F(\pi') - F(\pi))/F(\pi') < \varepsilon$;
(B) $F(\beta') < \lambda$;
(C) $(\mathscr{C}_\beta - \mathscr{C}_{\beta'})/\mathscr{C}_\beta$,
where $\varepsilon, \lambda, \rho$ are certain parameters (non-negative real numbers).

Definition 2. If *J-opt* subpermutation β' is not a *T-block* in the π permutation, and at the same time meets one of the conditions (A) or (B) or (C), then we call it a *semi T-block* (abbreviated to *sT-block*).

Unfortunately, *sT-block* does not have the *T-block* property contained in Proposition 1. Swapping the order of elements in *sT block* can lead to an improvement in the value of the objective function. However, the size of the improvement can be controlled through appropriate choice of parameters. Due to this, one can possibly reject only slightly better solutions.

Generating the neigborhood in local improvement algorithms, to eliminate certain solutions we will generally apply *T-blocks*. If it turns out that the number of tasks in *T-blocks* is small then we will use *sT-blocks*. We will experimentally determine not only which of the limitations (A), (B) or (C) but also with which parameters they will be used.

4.2. Blocks of tardy tasks. Let permutation of tasks

$$\pi = (1,2,\dots,a,a+1,a+2\dots,b,b+1,\dots,n) = (\alpha,\beta,\gamma),$$

where

$$\alpha = (1,2,\dots,a), \quad \beta = (a+1,a+2,\dots,b),$$
$$\gamma = (b+1,b+2\dots,n).$$

Let us assume that in subpermuation β all tasks are tardy, what is more

$$\forall i \in \beta,\ d_i < \mathscr{C}_{a-1} + p_{i,2}. \tag{22}$$

If $a = 1$, then we assume $\mathscr{C}_{a-1} = 0$.

From the condition (22) it follows that any task from β placed in the first position, in β i.e. on the position a, is tardy.

Let us assume that in the β subpermutation all tasks are tardy, i.e. (22) restrictions are met. From β we generate two new subpermutations:

(a) *J-opt* subpermutation β', i.e. we designate the order of elements using Johnsona's algorithm (**Algorithm 1**),
(b) subpermutation β'' setting tasks according to non increasing values of quotients $w_i/(p_{i,1}+p_{i,2})$.

Proposition 2. In single machine problem with the criterion $\sum w_i T_i$ subpermutation β'' (point (b)) it's optimal due to the sum of the costs of tardiness (Smith [29]).

Definition 3. Subpermutation β'' (point (b)) is called a *semi D-block* (in short *sD-block*), if

$$(\mathscr{C}_{b''} - \mathscr{C}_{b'})/\mathscr{C}_{b''} \le \varphi,$$

where φ is a parameter, whereas $\mathscr{C}_{b''}$ and $\mathscr{C}_{b'}$ are the due dates for completing the last task, respectively in β' and β''.

Defining the neigborhood, we will omit the solutions generated by swapping the order of tasks in any *sD-block*.

Theorem 4. Any permutation $\pi \in \Pi$ can be shown in the form of a sequence

$$\pi = [B^1, B^2, \dots, B^t],$$

where B^i $(i = 1,2,\dots,t)$ is a block or semi-block.

Proof. The proof is similar as in Property 1 in Wodecki's work [36]. □

In the further part of the work we will consider blocks and semi-blocks, which:

(i) have at least 4 elements,
(ii) are maximum due to the inclusion, i.e. an element cannot be added to the beginning or end in such a way that the block or semi block definition is still met.

5. Tabu search algorithm

In order to solve the T2FS problem there was a standard version of the tabu search method (abbreviated to TS) with the neighborhood generated by insert type moves.

The algorithm starts with some feasible (base) solution π^0. In i-th iteration, the neighborhood $\mathscr{N}(\pi^i)$ of the solution π^i is generated. It is a subset of the set of solutions generated from π^i through moves (changes in the position of elements in the permutation). An element from the neighborhood with the minimum value of the objective function is the base solution in the next iteration of the algorithm. During the execution of next iterations, the best found solution π^* is stored. The algorithm terminates when the stop condition is met (e.g. fixed calculation time or number of iterations).

To prevent cyclic repetition of solutions, there is the so-called tabu list (abbreviated as TL) used. Some attributes of recently considered base solutions are remembered on this list. When determining the minimum element from the neighborhood, solutions that have their attributes on the TL list are omitted. There is also the so-called criterion of aspiration used. If the value of the goal function $F(\beta)$, from the *TL* list of some solution β is less than $F(\pi^*)$, then the solution is not treated as tabu. In order to diversify the search process, there was a *backtrack jump*) mechanism introduced, which consists in resuming the search process starting from remembered promising solutions. It is implemented through the so-called long-term memory (abbreviated to *LTM*). It has the form of a fixed length list, where (β, TL) pairs are stored, in which β is the solution, whereas *TL* is the tabu list. If $F(\beta) < F(\pi^*)$ (where β is not the best solution from the searched neighborhood, then the element β and the current list of forbidden moves (tabu list) TL with added solution attributes β are added to LTM list. Backtrack jump (to the last element saved in the LTM list) is executed in the case when in the search process there appears a cycle with a previously fixed length, or when through the process of executing a certain, fixed number of iterations there is no improvement to the best solution that has been found so far.

Simple local search procedures, as Descent Search, rely on the monotonic improvement and stop after obtaining local minimum for which all solutions in the so-called neighborhood are worse or not better than the obtained minimum solution. The main improvement of considered Tabu Search method compared to classic Descent Search is that it can overcome local optima and keep the searching process going. To prevent the trajectory from making cycles, Tabu Search remembers the history of the searching process in the list of forbidden moves. Usually only a few last solutions are kept, however in some theoretical cases it is assumed to remember the *whole* searching history, which makes it possible to prove theoretical convergence of such a TS algorithm (see Hanafi [16]). However there is no theoretical convergence property for the classic Tabu Search method with limited short-time memory remember on tabu list, however the method is extremely fast convergent, much faster than almost all other metaheuristics – but this convergence is observed empirically. What is more, for some kinds of neighborhoods (e.g. N1 neighborhood in TSAB algorithm of Nowicki and Smutnicki [26]) there is no connectivity property, which means that there is no theoretical possibility proven to achieve all the feasible solutions of the space searched from a starting solution. Even without this theoretical property, over 23-years old TSAB algorithm is one all the best metaheuristics for jobs-scheduling problems up to now.

6. Parallel tabu search algorithm

In order to use the parallel computational platform used earlier in the launch of the Johnson parallel algorithm described in Section 3.1, a parallel tabulation algorithm was implemented in MPSS version (Multiple starting Points Single Strategy) according to the Voß [34] classification describing parallel *tabu search* algorithm. The algorithm in the proposed version is adapted to be implemented using the MPI library (see Fig. 1), and its pseudo-code is included in the Algorithm 3. Wider considerations on parallel tabu search algorithms in order to solve scheduling problems can be found in the work of Bożejko et al. [6].

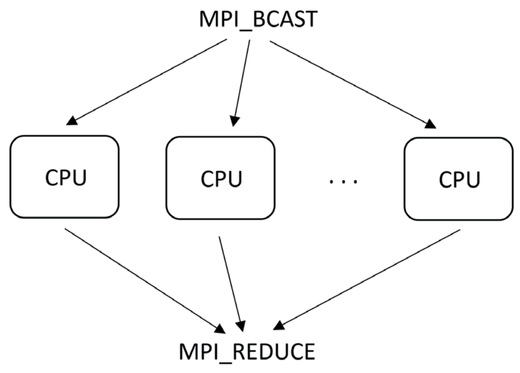

Fig. 1. Schema of the independent MPSS paralell tabu search

The algorithm starts work with a vector of startup solutions π^0 assigned to individual processors (line 1) using the `MPI_Bcast` function. Each processor stores its own local best solution π_p^* (line 4), which is used in the mechanism of back-track jump in the lack of improvement in the quality of the current solution π_p by a certain number of iterations *max_it* (line 15). The loop in lines 5–20 is the core of the algorithm. Successively it is performed concurrently in it:

- generating the neighborhood $N_p(\pi_p)$ of the current solution π_p (line 6),
- removing from this neighborhood solutions forbidden by the local tabu list TL_p (line 7),
- selection of the best solution β_p from the neighborhood (line 8),
- add move parameters to the local tabu list TL_p (line 9) and go to the next solution (line 10).

Next, it is checked if the current solution is better than the best remembered (lines 11–14) and possible correction of this solution along with saving it on the long-term memory list LTM used for back-track jumps. Finally, the best solutions π_p, $p = 1,2,\dots,max\_CPU$, obtained by individual processors (line 21) are collected using the `MPI_Reduce` function.

Algorithm 3: Parallel tabu search algorithm

```
Input  : $\pi^0$ – vector of $max\_CPU$ initial permutations;
         $L$ – vector of $max\_CPU$ lengths of tabu lists;
         $|LTM|$ – lenght of the long time memory list;
Output: solution $\pi^*$ – optimal permutation;
1  Broadcast starting solutions $\pi^0$;
2  parfor $p = 1,2,\dots,max\_CPU$ do
3      $\pi_p \leftarrow \pi_p^0$;   { local for each CPU }
4      $\pi_p^* \leftarrow \pi_p^0$;   $TL_p \leftarrow \emptyset$;
5      for $i = 1,2,\dots,max\_iteration$ do
6          Generate a neighborhood $N_p(\pi_p)$ of the
             solution $\pi_p$;
7          Remove from $N_p(\pi_p)$ solutions forbidden by $TL_p$;
8          $\beta_p = \arg\min_{\sigma_p \in N_p(\pi_p)} F(\sigma_p)$;
9          Add arguments of the move from $\pi_p$ to $\beta_p$ to the
             $TL_p$ removing the oldest one not exceeding the
             length $L_p$ ;
10         $\pi_p \leftarrow \beta_p$;
11         if $F(\pi_p) < F(\pi_p^*)$ then
12             $\pi_p^* \leftarrow \pi_p$;
13             Add $\pi_p^*$ to $LTM$ together with $TL_p = TL$
                 including the next move;
14         end
15         if (there is no improvement of $\pi_p^*$ from the last
             max_it iterations) then
16             Take a new $\pi_p$ from the $LTM$ together with its
                 tabu list $TL$;
17             $TL_p \leftarrow TL$;
18         end
19     end
20 end
21 Reduce all the $\pi_p^*$ to the best $\pi^*$;
22 return $\pi^*$;
```

7. Test examples

Since there are no test examples in the literature, for the problem under consideration, thus for the needs of implementation of computational experiments were there were nine different sets of test instances randomly generated. Adaptation of Vallada, Ruiz and Framinan test instances [33] was not possible by taking only two machines and w_i and d_i, respectively, because the delay values T_i for data of [33] and solutions to our problem would in most cases be equal to zero. In turn, in the cited work [28], no test instances are available. So we decided to propose the new test instances and publish them in [32].

Task execution times on individual machines were generated randomly, according to uniform distribution from the set $\{1,2,\ldots,99\}$, whereas weights of the w_i penalty function from the set of $\{1,2,\ldots,9\}$. The values of the requested due dates for completing tasks were designated basing on two parameters: T – *tardiness factor* and R – *due date range*. These terms (non-negative integers) were, according to the uniform distribution, drawn from the interval $[P(1-T-R/2),(1-T+R/2)]$. Parameter

$$P=\sum_{i=1}^{n}\sum_{j=1}^{2}p_{i,j}\,,$$

is the upper bound value (the sum of execution times of all operation) for the criterion $C_{\max}$ (see Taillard [31]). This method of determining the desired dates of the completion of tasks was described by Potts at work [27]. Test examples were generated for each pair of the parameter values $T=\{0.2,0.4,0.6\}$ and $R=\{0.2,0.6,1.0\}$. In total there are nine such pairs. In some cases, especially for small values of T and for large values of R, the drawn number can be negative. In such cases it was assumed that the requested the due date for completing the task is zero. Examples were generated for the number of tasks $n=10,20,50,100,200,500$ and 1000. For each value of n there were 10 examples generated, in total 70 examples, for each pair of values T and R. Ultimately, in computational experiments there were 630 examples used that have also been placed on the website of Uchroński [32].

8. Computational experiments

The algorithm for solving two-machine flow shop problems with the minimization of the sum of tardiness costs was implemented in the C++ language. Computational experiments were carried out on the Bem cluster in the Wrocław Network – Supercomputer Center(grant no. 96) working under 64-bit control operating system Scientific Linux 6.7 (Carbon) equipped with Intel Xeon processors E5-2670 (2.30 GHz). The results of three algorithms were compared:

NEH – construction algorithm [25], in which tasks were sorted according to non-increasing values $\dfrac{p_{i,1}+p_{i,2}}{w_i}$,

TS_B – tabu search with long-term memory and the use of sT and sD-blocks properties,

TS_{BJ} – tabu search with long-term memory, the use of sT and sD-blocks properties and Johnson's algorithm.

The starting point for metaheuristic algorithms was natural permutation and calculation time, of a single example (algorithms TS_B and TS_{BJ}) was limited to 60 sec.

For each example there was the value of relative error calculated:

$$\delta=\frac{\mathcal{T}_{ref}-\mathcal{T}_{Alg}}{\mathcal{T}_{ref}}\cdot 100\% \tag{23}$$

where $\mathcal{T}_{ref}$ is the value of the objective function for the reference solution, whereas $\mathcal{T}_{Alg}$ is the value of criterion function for the solution determined by the tested algorithm,

$$Alg\in\{TS_B,\,TS_{BJ},\,NEH\}.$$

For the reference solution $\mathcal{T}_{ref}$ the results of the tabu search algorithm without block properties were accepted.

Having performed the calculations, it turned out that the difficulty of the example (relative improvement of the start-up solution) depends on the parameter values R and T determining the size and position on the a numerical interval axis from which tasks completion dates are drawn. Results of parallel tabu search algorithm solutions quality are presented in Table 1. Experiments have been conducted for the number of processors $p=2,4$ and 8. One can observe, that increasing of the number of processors results in improving of the quality of the obtained solutions. Table 2 presents mean relative errors δ_{aprd} of examples generated for the smallest ones values of $R=T=0.2$. According to the solution's expectations the construction algorithm NEH was on average almost 70% worse than TS solutions. The best results were obtained when in construction of the algorithm there were blocks and quasi blocks used. In this case the average relative error (improvement of the solution) is -6.97%. Using only the blocks themselves is much less effective, because the average error (improvement) is almost 3 times smaller and amounts to -2.40%. These examples were thoroughly analyzed and it turned out that the drawn random values of task completion dates are small and not much varied. There are many tasks that are tardy and what is more blocks of due date tasks are practically absent. Hence their low efficiency.

Table 1
Values δ_{aprd} for groups of test examples with different parameters T and R

parameters	n	$p=2$	$p=4$	$p=8$
$T=0.2$, $R=0.2$	10	-1.31	-1.88	-3.00
$T=0.2$, $R=0.6$	20	-7.47	-9.41	-11.18
$T=0.2$, $R=1.0$	50	-9.12	-11.57	-13.58
$T=0.4$, $R=0.2$	100	-0.50	-0.75	-0.95
$T=0.4$, $R=0.6$	200	-4.45	-6.56	-9.79
$T=0.4$, $R=1.0$	500	-5.21	-11.55	-15.06
$T=0.6$, $R=0.2$	100	-0.21	-0.38	-0.55
$T=0.6$, $R=0.6$	200	-1.33	-1.91	-3.01
$T=0.6$, $R=1.0$	500	-2.10	-3.07	-4.48
mean		-3.52	-5.23	-6.84

Table 2
Values δ_{aprd} for groups of test examples with parameters $T = 0.2$ and $R = 0.2$

problem	n	NEH	TS_B	TS_{BJ}
2FPWT01-10	10	75.39	1.99	8.20
2FPWT11-20	20	78.41	0.40	42.88
2FPWT21-30	50	93.70	−2.11	−21.57
2FPWT31-40	100	117.16	−0.92	−51.01
2FPWT41-50	200	117.19	−5.79	52.66
2FPWT51-60	500	6.09	−8.86	4.02
2FPWT61-70	1000	1.37	−1.51	1.37
mean		69.90	−2.40	5.22

In turn, for larger values of the parameters R and T examples are much easier, i.e. it is definitely more effective the use of both blocks and semi-blocks. The results obtained for a group of examples with maximum values of both parameters are presented in Table 3. The surprising fact is, in this case, a considerable improvement of the solutions of the algorithm TS_{BJ} for examples with the smallest size $n = 10, 20, 30$ and 40. It is from 54 to 75 percent. At the same time, the improvement for the biggest examples of $n = 1000$ is only -3.69%. In this case, a single iteration of the algorithm appeared to be time-consuming. With computation time limited to 60 seconds too few iterations are performed to significantly improve the starting solution. In both cases (Table 2 and 3) the error of NEH algorithm is exceptionally big (with complexity $O(n^2)$, which on average is over 82%. In the classic flow shop problem with – criterion $C_{\max}$ (i.e. $F||C_{\max}$ problem) mean relative error of the NEH algorithm in relation to the best known solutions of Taillard's examples [31]) does not exceed several percent. For the algorithm discussed in this paper it is then not useful.

Table 3
Values δ_{aprd} for groups of test examples with parameters $T = 0.6$ and $R = 1.0$

problem	n	NEH	TS_B	TS_{BJ}
2FPWT01-10	10	34.50	0.00	−54.00
2FPWT11-20	20	73.76	1.17	−67.84
2FPWT21-30	50	191.67	3.57	−75.96
2FPWT31-40	100	234.87	1.77	−69.32
2FPWT41-50	200	40.72	−4.83	−19.55
2FPWT51-60	500	1.81	−2.59	−11.75
2FPWT61-70	1000	0.80	−0.09	−3.69
mean		82.59	−0.14	−43.16

Mean relative errors, after computing all 630 examples, for two metaheuristic algorithms described in this paper, are presented in Table 4. The use of blocks in the construction of a tabu search algorithm gave some improvement in results. The average improvement is -6.57%. The introduction of additional quasi-blocks has already given a significant improvement of results. In relation to the output algorithm. this improvement is -43.16%. For examples of smallest sizes ($n = 10$ and 20) using the complete review method, there were optimal solutions determined. It turned out that in 95% solutions determined by the algorithm TS_{BJ} were optimal and the maximum relative error (in relation to optimal solutions) did not exceed 3%.

Table 4
Values δ_{aprd} for test examples and different parameters T and R

parameters	n	TS_B	TS_{BJ}
$T = 0.2$, $R = 0.2$	10	−2.40	−6.97
$T = 0.2$, $R = 0.6$	20	−10.26	−62.58
$T = 0.2$, $R = 1.0$	50	−10.11	−74.45
$T = 0.4$, $R = 0.2$	100	0.15	−5.52
$T = 0.4$, $R = 0.6$	200	−4.18	−33.92
$T = 0.4$, $R = 1.0$	500	6.05	−77.88
$T = 0.6$, $R = 0.2$	100	−0.06	−7.20
$T = 0.6$, $R = 0.6$	200	−1.14	−24.22
$T = 0.6$, $R = 1.0$	500	−6.57	−43.16
mean		−2.4	−37.94

Results of parallel Johnson's algorithm speedup values are presented in Table 5. Experiments have been conducted for different problem size (from 10^3 to $5 \cdot 10^6$) using parallel processors $p = 2, 4, 8$ and 16. One can observe, that for a different values of n the speedup initially increases fairly rapidly and then reaches maximum value. Obtained speedup values depends also on the size of the problem.

Table 5
Values of speedups for different number of processors – Intel Xeon E5-2670

$n/10^3$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
1	1.48	0.29	0.12	0.02
2	1.73	1.71	0.29	0.04
5	1.84	2.37	0.38	0.14
10	1.88	2.70	1.13	0.33
20	1.88	3.07	1.71	0.50
50	1.90	3.27	3.17	0.96
100	1.90	3.34	4.26	1.77
200	1.89	3.35	4.91	3.19
500	1.85	3.31	5.05	5.30
1000	1.81	3.29	5.24	5.27
2000	1.79	3.22	5.17	5.95
5000	1.74	3.18	5.21	6.54

9. Summary

In the work, a two-machine total weighted tardiness flow shop scheduling problem was considered. The description, mathematical model and properties accelerating the neighborhood search were presented. Applying some of these properties (i.e. blocks and quasi-blocks) causes omission of better solutions than current ones. Generally, this is a situation that we would like to avoid because it can lead to the elimination of good solutions. The above presented properties have been used in the construction of an algorithm based on the tabu search method. Since there is no test data in the literature, they were generated randomly. The conducted computational experiments have unambiguously demonstrated that the algorithm 'with properties' sets significantly better solutions (on average the relative improvement of the solution is almost 38%) than the algorithm without these properties. Methods for determining moves being representatives of certain sets of moves were also presented, i.e. generating not worse solutions. Unfortunately, their application brought a small improvement – just at the level of about 0.5%. As it turned out, designation of representatives is a time-consuming procedure. Since the operation time of the tabu search algorithm was limited to 60 seconds, the designation of representatives of moves resulted in reducing the number of iterations of the algorithm and this may be the reason for their low efficiency.

Acknowledgements. The paper was partially supported by the National Science Centre of Poland, grant OPUS no. 2017/25/B/ST7/02181 and statutory grant no. 0401/0023/18 of Faculty of Electronics, Wrocław University of Science and Technology. Computational experiments were performed in the Wrocław Centre for Networking and Supercomputing, grant no. 96.

References

[1] M.-H. Ahmadi-Darani, G. Moslehi, and M. Reisi-Nafchi, "A two-agent scheduling problem in a two-machine flowshop", *International Journal of Industrial Engineering Computations* 9 (3), 289–306 (2018).

[2] M. Al-Salem, L.B. Valencia, and G. Rabadi, "Heuristic and Exact Algorithms for the Two-Machine Just in Time Job Shop Scheduling Problem", *Mathematical Problems in Engineering* 5, 1–11 (2016).

[3] M.A.A. Ardakan, K.S. Beheshti, H. Mirmohammadi, and H.D. Ardakani, "A hybrid meta-heuristic algorithm to minimize the number of tardy jobs in a dynamic two-machine flow shop problem", *Numerical Algebra, Control & Optimization* 7 (4), 465–480 (2017).

[4] M. Bank, S.M. Fatemi, T. Ghomi, F. Jolai, and J. Behnamian, "Two-machine flow shop total tardiness scheduling problem with deteriorating jobs", *Applied Mathematical Modelling* 36 (11), 5418–5426 (2012).

[5] W. Bożejko, J. Grabowski, and M. Wodecki, "Block approach tabu search algorithm for single machine total weighted tardiness problem", *Computers & Industrial Engineering* 50 (1-2), 1–14 (2006).

[6] W. Bożejko, M. Uchroński, and M. Wodecki, "Parallel metaheuristics for the cyclic flow shop scheduling problem", *Computers & Industrial Engineering*, 95, 156–163 (2016).

[7] W. Bożejko, A. Gnatowski, R. Idzikowski, and M. Wodecki, "Cyclic flow shop scheduling problem with two-machine cells", *Archives of Control Sciences* 27 (2), 151–168 (2017).

[8] W. Bożejko, J. Pempera, and M. Wodecki , "Minimal cycle time determination and golf neighborhood generation for the cyclic flexible job shop problem", *Bull. Pol. Ac.: Tech.* 66 (3), 333–344 (2018).

[9] R.L. Bulfin and R. Hallah, "Minimizing the weighted number of tardy jobs on two-machineflow shop", *Computers & Operations Research* 30, 1887–1900 (2003).

[10] S.-R. Cheng, Y. Yunqiang, Ch.-H. Wen, W.-Ch. Lin, Ch.-Ch. Wu, and J. Liu, "A two-machine flowshop scheduling problem with precedence constraint on two jobs", *Soft Computing – A Fusion of Foundations, Methodologies and Applications* 21 (8), 2091–2103 (2017).

[11] R. Cole, "Parallel merge sort", *SIAM J. Comput.* 17 (4), 770–785 (1988).

[12] J. Grabowski and M. Wodecki, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion", *Computers and Operations Research* 31, 1891–1909 (2004).

[13] L.R. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy-Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey", *Annals of Discrete Mathematics* 5, 287–326 (1979).

[14] J.N.D. Gupta and A.M.A. Hariri, "Two-machine flowshop scheduling to minimize the number of tardy jobs", *Journal of the Operational Research Society* 48, 212–220 (1997).

[15] I. Hamdi, A. Oulamara and T. Loukil, "A branch and bound algorithm to minimise the total tardiness in the two-machine permutation flowshop scheduling problem with minimal time lags", *International Journal of Operational Research* 23 (4), 387–405 (2015).

[16] S. Hanafi, "On the Convergence of Tabu Search", *Journal of Heuristics* 7, 47—58 (2000).

[17] S.M. Johnson, "Optimal two- and three-stage production schedules with setup times included", *Naval Res. Logist. Quart.* I, 61–68 (1954).

[18] M. Khalili and R. Tavakkoli-Moghaddam, "A multi-objective electromagnetism algorithm for a bi-objective flowshop scheduling problem", *Journal of Manufacturing Systems*, 31 (2), 232–239 (2012).

[19] M. Kharbeche and M. Haouari, "MIP models for minimizing total tardiness in a two-machine flow shop", *The Journal of the Operational Research Society* 64 (5), 690–707 (2013).

[20] C. Koulamus, "The total tardiness problem: review and extensions", *Operations Research* 42, 1025–1041 (1994).

[21] J.-Y. Lee and Y.-D. Kim, "Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraint on the first machine", *Computers & Industrial Engineering* 114, 22–30 (2017).

[22] J.K. Lenstra, A.G.H. Rinnoy Kan and P. Brucker, "Complexity of Machine Scheduling Problems", *Annals of Discrete Mathematics* 1, 343–362 (1977).

[23] B.M.T. Lin, "Scheduling in the two-machine flowshop with due date constraints", *International Journal Production Economics* 70, 117–123 (2001).

[24] A. Moukrim, D. Rebaine, and M. Serairi, "A branch and bound algorithm for the two-machine flowshop problem with unit-time operations and time delays", *RAIRO-Oper. Res.* 48, 235–254 (2014).

[25] M. Nawaz, E.E. Enscore, and I. Ham, "A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem", *OMEGA* 11 (1), 91–95 (1983).

[26] E. Nowicki and C. Smutnicki, "A Fast tabu serach algorithm for permutation flow shop problem", *European Journal of Operational Research* 91, 160–175 (1996).

[27] C.N. Potts and L.N. Van Wassenhove, "A decomposition algorithm for the single machine total tardiness problem", *Operations Research Letters* 1, 177–181 (1982).

[28] J. Schaller, "Note on minimizing total tardiness in a two-machine flowshop", *Computers & Operations Research* 32 (12), 3273–3281 (2005).

[29] W.E. Smith, "Various Optimizers for Single-Stage Production", *Naval Research Logistics Quarterly* 3 (1-2), 59–66 (1956).

[30] Q.C. Ta, J.-C. Billaut, and J.-L. Bouquard, "Recovering beam search and Matheuristic algorithms for the $F2||\sum T_j$ scheduling problem", In *11th Workshop on Models and Algorithms for Planning and Scheduling Problems* (2013), France.

[31] E. Taillard, "Benchmarks for basic scheduling problems", *European Journal of Operational Research* 64, 278–285 (1993).

[32] M. Uchroński, "Benchmark files for 2FP$||\sum$wiTi" (2018), `https://zasobynauki.pl/zasoby/51102/`

[33] E. Vallada, R. Ruiz, and J. Framinan, "New hard benchmark for flowshop scheduling problems minimising makespan", *European Journal of Operational Research* 240, 666–677 (2015).

[34] S. Voß, "Tabu search: Applications and prospects", in: *Network Optimization Problems* (D.Z. Du and P.M. Pardalos, eds.), World Scientific Publishing Co., Singapore, 333–353 (1993).

[35] M. Wodecki, "A branch-and-bound parallel algorithm for single-machine total weighted tardiness problem", *International Journal of Advanced Manufacturing Technology* 37 (9-10), 996–1004 (2008).

[36] M. Wodecki, "A block approach to earliness-tardiness scheduling problems", *International Journal of Advanced Manufacturing Technology* 40 (7-8), 797–807 (2009).