







The Problem of Tasks Scheduling with Due Dates in a Flexible Multi-machine Production Cell

Wojciech Bożejko¹(✉) , Piotr Nadybski² , Paweł Rajba³ ,
and Mieczysław Wodecki⁴ 

¹ Department of Automatics, Mechatronics and Control Systems, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.edu.pl

² Witelon State University of Applied Science in Legnica, Sejmowa 5A, 59-220 Legnica, Poland
nadybskip@pwsz.legnica.edu.pl

³ Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland
pawel@cs.uni.wroc.pl

⁴ Department of Telecommunications and Teleinformatics, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
mieczyslaw.wodecki@pwr.edu.pl

Abstract. In the paper we consider an NP-hard problem of tasks scheduling with due dates and penalties for the delay in a flexible production cell. Each task should be assigned to one of the cell's machines and the order of their execution on machines should be determined. The sum of penalties for tardiness of tasks execution should be minimized. We propose to use the tabu search algorithm to solve the problem. Neighborhoods are generated by moves based on changing the order of tasks on the machine and changing the machine on which the task will be performed. We prove properties of moves that significantly accelerate the search of the neighborhoods and shorten the time of the algorithm execution and in result significantly improves the efficiency of the algorithm compared to the version that does not use these properties.

1 Introduction

Optimizing the production process relies on designating of such a schedule for the execution of the elements that gives optimal effects measured by the value of a certain criterion (e.g. execution time, sum of penalties for delays, etc.). This usually comes down to the formulation of a combinatorial problem in which an optimal element (sequence or permutation) should be determined from a finite set, usually very large, of feasible solutions.

In Just-in-Time manufacturing systems there is a requirement to complete each element before the expiration of the due date, otherwise a penalty is being

calculated for exceeding it (for the delay or being tardy). In the problem considered here a set of tasks and a set of machines are given. Machines of the same type (parallel machines), i.e. with the same functional properties, form a flexible production cell. Each task must be performed on a machine of the cell. The data includes the times of completing tasks on each of machines, and their due dates. The penalty for being tardy depends on the tardiness value and the penalty factor, different for each task. The problem is to assign tasks to machines and arrange their execution on each machine to minimize the sum of penalties (costs). In short, we will denote this problem by **MTC**.

If the cell consists of only one machine then the **MTC** problem comes down to the single machine scheduling problem. In the literature it is denoted by $1||\sum w_i T_i$ and it belongs to the class of NP-hard problems (see [7]; of course it means, that **MTC** problem is also NP-hard). Its exact description, specific properties and a very effective tabu search algorithm can be found in Bożejko et al. [2]. The parallel exact algorithm described Wodecki [14]. In contrast, a generalization of the **MTC** problem is the flexible task shop problem in which a set of machines is partitioned into cells. According to the relationship of the technological order, each task passes through many cells. In the literature, it is mainly considered with the C_{\max} criterion. Unfortunately, models and problems of scheduling production with other cost criteria, despite significant practical needs, are relatively rarely considered. This is mainly due to the irregularity of the criterion functions and the lack of specific properties of problems leading to a limitation of the set of solutions. The most important, containing main theoretical results, and with good algorithms are papers written by: Karabulut [5], Liu [9], Kayvanfar et al. [6], Ojstersek, Buchmeister [11], Bulfin, Hallah [4], Bożejko et al. [3], Park et al. [10], Tocovicha et al. [13].

In this work, we will present a description of the problem, prove some of its properties and provide constructive and tabu search algorithms. In the design of algorithms, the neighborhood generated by the moves based on changing the order of execution of tasks on the machine and changing the machine on which the task will be performed is used. The properties of the problem have been proven here to enable the elimination of worse solutions from the neighborhood. As a result, the number of elements of the neighborhood is significantly reduced, thus, the time needed for search also decreases. Due to the lack of bigger instances of the considered problem, computational experiments were performed on randomly generated examples.

2 Problem Description

There is a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n tasks given to be performed on the m machines from the set $\mathcal{M} = \{1, 2, \dots, m\}$. For the task $i \in \mathcal{J}$, there are the following notions introduced:

- $p_{i,j}$ - execution time , $j \in \mathcal{M}$,
- d_i - required completion date ,
- w_i - weight of the penalty function (cost of delay).

In the **MTC** problem, the tasks are to be assigned to machines and there must be determined the order of their execution on each machine to minimize the sum of the tardinesses costs. If the assignment of tasks to machines and the order in which they are performed is determined, then for a task $i \in \mathcal{J}$ the following designations are introduced:

- C_i - completion date ,
- $T_i = \max\{0, C_i - d_i\}$ - delay,
- $w_i \cdot T_i$ - penalty (cost) of the delay.

Therefore, the problem consists in designating such an allocation of tasks to machines and the order of their execution to minimize the sum of penalties (costs of tardinesses), i.e. $\sum_{i=1}^n w_i \cdot T_i$, wherein the following constraints must be met:

- (i) the task must be performed on exactly one machine,
- (ii) the task execution cannot be interrupted,
- (iii) at any given time, the machine can perform only one task.

Let a sequence of tasks' sets

$$A = (A_1, A_2, \dots, A_m),$$

such that $A_i \subset \mathcal{J}$, $A_i \cap A_j = \emptyset$, $i \neq j$, $i, j \in \mathcal{M}$ and $\sum_{i=1}^m A_i = \mathcal{J}$, is called *assignment of tasks to machines*. By \mathcal{A} we denote the set of all such assignments.

For assignment $A = (A_1, A_2, \dots, A_m)$, $A \in \mathcal{A}$, let $\pi = (\pi_1, \pi_2, \dots, \pi_m)$ will be a sequence of permutations such that π_i is an n_i -elementary ($n_i = |A_i|$) permutation (order of execution) of tasks from the set A_k , $k \in \mathcal{M}$, i.e. performed by a k -th machine. By $\mathcal{P}(A)$ we denote the set of all such sequences of permutations (in short these sequences will be called *task permutations*).

The set of solutions of the **MTC** problem of performing tasks on machines can be defined as follows:

$$\mathcal{AP} = \{(A, \pi) : A \in \mathcal{A}, \pi \in \mathcal{P}(A)\}. \tag{1}$$

Let the solution $\mathbf{S} = (A, \pi) \in \mathcal{AP}$, where $A = (A_1, A_2, \dots, A_m)$ and $\pi = (\pi_1, \pi_2, \dots, \pi_m)$. If the task $i \in \mathcal{J}$ is executed on a machine $k \in \mathcal{M}$ (i.e. $i \in A_k$) as the l -th in the order ($\pi_k(l) = i$), then the earliest moment of its completion $C_i = \sum_{j=1}^l p_{\pi_k(j)}$, tardiness $T_i = \max\{0, C_i - d_i\}$, and a penalty for the tardiness $f_i(\mathbf{S}) = w_i \cdot T_i$. In turn

$$\mathcal{T}(\mathbf{S}) = \sum_{j=1}^n w_{\pi(i)} T_{\pi(i)} = \sum_{j=1}^n f_j(\mathbf{S}), \tag{2}$$

is the cost of the execution of all tasks (the sum of the penalties for delays).

The problem of task execution on **MTC** machines considered in the work is to determine the optimal solution $\mathbf{S}^* \in \mathcal{AP}$. This problem is NP-hard because for the number of machines $m = 1$, we get NP-hard single-machine problem, as it was written in the Introduction.

In the case where the number of machines $m = 2$ the cardinality of the set of all assignments $|\mathcal{A}| = 2^n$. For each assignment, the number of all possible sequences of tasks is $O(n!)$. In this case all possible solutions to the problem are $O(2^n \cdot n!)$.

3 Solution Method

Scheduling operations in a flexible production cell, solving the **MTC** problem, requires simultaneous decision making on two levels:

1. assigning tasks to machines,
2. determining the order of performing tasks on each machine.

Therefore, an idea of the problem solving method can be presented in the form of the following algorithm:

ATMC algorithm

Let \mathbf{S}, \mathbf{S}^* be solutions, $\mathbf{S} = (A, \pi)$, $\mathbf{S} \in \mathcal{AP}$ and $\mathbf{S}^* := \mathbf{S}$.

repeat

Step 1: generate from \mathbf{S} a new assignment $A' \in \mathcal{A}$;

Step 2: for assignment A' generate permutation

$\pi' \in \mathcal{P}(A')$ - of the task execution order;

$(\mathbf{S}' = (A', \pi')$ is a new solution);

if $\mathcal{T}(\mathbf{S}') < \mathcal{T}(\mathbf{S}^*)$ then $\mathbf{S}^* := \mathbf{S}'$;

$\mathbf{S} := \mathbf{S}'$;

until {*Stop condition*}

\mathbf{S}^* is the solution determined by the algorithm (output).

Step 1 can be accomplished in many ways (e.g. random selection, constructive algorithm, etc.). It should only be emphasized that for two machines the number of possible assignments is 2^n . In turn the implementation of Step 2 requires, for each machine, determining the sequence of performing the assigned tasks. Determining the optimal order (i.e. minimizing the sum $\sum w_i T_i$) is an NP-hard problem. It boils down to solving the NP-hard one-machine task scheduling problem $1||\sum w_i T_i$. Therefore, to solve the **MTC** problem under consideration there will be the approximate algorithm based on the tabu search method used. Its essential elements are neighborhoods, the subsets of the set of feasible solutions generated from the current solution by transformations called moves. When browsing the neighborhood, we select the element with the lowest value of the criterion function, which we adopt as new, current solutions in the next iteration of the algorithm. From a fixed solution, another solution can be generated by executing the move (**Step 1**) consisting of:

1. changing the order of execution of tasks on a certain machine, or
2. *transfer* of the task from one machine to another.

Both of these moves will be described in details further.

For a solution $\mathbf{S} \in \mathcal{AP}$ ($\mathbf{S} = (A, \pi)$) by

$$\mathcal{T}_k(\mathbf{S}) = \sum_{j \in A_k} f_j(\mathbf{S}), \tag{3}$$

we denote the cost of performing tasks by k -th machine. This value will be denoted in short k -th cost. Therefore, the cost of performing all tasks (2) is equal to the sum of the costs of individual machines, i.e. $\mathcal{T}(\mathbf{S}) = \sum_{k=1}^m \mathcal{T}_k(\mathbf{S})$.

3.1 Changing the Order of Tasks on the Machine

To change the order of execution of tasks on a machine in a solution ($\mathbf{S} = (A, \pi)$) there will be an *insert*-type of move used. It consists in shifting a task into a different position. Let π_k be n_k -elementary permutation – an order of tasks on k -th machine. If $1 \leq s, t \leq n_k$, then *insert*-type of move consists in swapping the element from position s (of task $\pi_k(s)$) to position t in permutation π_k . Two cases will be considered.

Case 1. Let us assume that $t \geq s$.

Then this move will be denoted by with $\overrightarrow{\eta_t^s}$, and generated permutation $\overrightarrow{\eta_t^s}(\pi_k) = \pi'_k$. Then:

$$\pi'_k(i) = \begin{cases} \pi_k(i), & \text{if } i < s \vee i > t, \\ \pi_k(i + 1), & \text{if } s \leq i < t, \\ \pi_k(s), & \text{if } i = t. \end{cases} \tag{4}$$

where ($\mathbf{S}' = (A, \pi')$). Having executed the move $\overrightarrow{\eta_t^s}$, in order π'_k

$$\mathcal{T}'_k(\mathbf{S}') = \sum_{i=1}^{s-1} f_{\pi'_k(i)}(\mathbf{S}) + f_{\pi'_k(s)}(\mathbf{S}) + \sum_{i=s+1}^{t-1} f_{\pi'_k(i)}(\mathbf{S}) + f_{\pi'_k(t)}(\mathbf{S}) + \sum_{i=t+1}^{n_k} f_{\pi'_k(i)}(\mathbf{S}).$$

It follows from definition (4) that

$$f_{\pi'_k(t)}(\mathbf{S}) = \max\{0, C_{\pi_k(t)} - d_{\pi_k(t)}\}, \tag{5}$$

$$\sum_{i=1}^{s-1} f_{\pi'_k(i)}(\mathbf{S}) = \sum_{i=1}^{s-1} f_{\pi_k(i)}(\mathbf{S}), \text{ and } \sum_{i=t+1}^{n_k} f_{\pi_k(i)}(\mathbf{S}) = \sum_{i=t+1}^{n_k} f_{\pi'_k(i)}(\mathbf{S}). \tag{6}$$

Since for the task completion times: $\pi'_k(s), \pi'_k(s + 1), \dots, \pi'_k(t)$

$C_{\pi'_k(t)} = C_{\pi_k(t)}$ and $C_{\pi'_k(i)} = C_{\pi_k(i)} - p_{\pi_k(s)} + p_{\pi_k(s+1)}$, for $i = s, s + 1, \dots, t - 1$,

where the cost of the task execution:

$$f_{\pi'_k(i)}(\mathbf{S}) = \max\{0, C_{\pi_k(i)} - p_{\pi_k(s)} + p_{\pi_k(s+1)} - d_{\pi_k(i)}\}. \tag{7}$$

For *insert*-type of move $\overrightarrow{\eta_t^s}$ ($t \geq s$), generating from π_k permutation π'_k , let

$$\delta_k(\overrightarrow{\eta_t^s}) = \sum_{i=s}^{t-1} f_{\pi'_k(i)}(\mathbf{S}) + f_{\pi'_k(t)}(\mathbf{S}) - \sum_{i=s}^t f_{\pi_k(i)}(\mathbf{S}). \tag{8}$$

Theorem 1. *If permutation π'_k was generated from π_k by execution of insert-type of move $\overrightarrow{\eta_t^s}$ ($t \geq s$), then*

$$\mathcal{T}'_k(\mathbf{S}') = \mathcal{T}_k(\mathbf{S}) + \delta_k(\overrightarrow{\eta_t^s}). \tag{9}$$

Proof. To prove of the theorem the following equality should be used (5)–(8).

Case 2. $t < s$. Let us assume that $t < s$. Then the *insert*-type of move generating from π_k a new permutation π''_k by swapping the task from position s to position t will be denoted by $\overleftarrow{\eta_t^s}$. In this case a generated permutation

$$\pi''_k(i) = \begin{cases} \pi_k(i), & \text{if } i < s \vee i > t, \\ \pi_k(i + 1), & \text{if } s \leq i < t, \\ \pi_k(s), & \text{if } i = t. \end{cases} \tag{10}$$

where $(\mathbf{S}'' = (A, \pi''))$. Similarly as in **Case 1** it is possible to determine equality similar to (5)–(7). Next, let

$$\delta_k(\overleftarrow{\eta_t^s}) = \sum_{i=s}^{t-1} f_{\pi''_k(i)}(\mathbf{S}) + f_{\pi''_k(t)}(\mathbf{S}) - \sum_{i=s}^t f_{\pi_k(i)}(\mathbf{S}). \tag{11}$$

Theorem 2. *If permutation π''_k was generated from π_k by insert-type of move $\overleftarrow{\eta_t^s}$ ($t < s$), then*

$$\mathcal{T}''_k(\mathbf{S}'') = \mathcal{T}_k(\mathbf{S}) + \delta_k(\overleftarrow{\eta_t^s}). \tag{12}$$

Proof. The proof of equality (12) is similar to the proof of Theorem 1. It follows from the Theorem 1 that if $\delta_k(\overrightarrow{\eta_t^s}) < 0$, then the execution of the *insert*-type of move generates solution with lower cost of execution of tasks (improvement of the solution). It is the same with the move $\overleftarrow{\eta_t^s}$ (Theorem 2). Each of these moves can therefore be a move improving the current solution.

3.2 Transferring of a Task to Another Machine

Let the solution $\mathbf{S} = (A, \pi)$, where $A = (A_1, A_2, \dots, A_m)$, $\pi = (\pi_1, \pi_2, \dots, \pi_m)$. We are considering two machines k and l ($k \neq l$, $k, l \in \mathcal{M}$). From a solution \mathbf{S} we generate $\mathbf{S}' = (A', \pi')$ by transferring a single task from a machine k to l . Let s be a position in permutation π_k , and t position in permutation π_l . The *transfer*-type of moves (τ -move) transfers the task from the position s on machine k

to position t on machine l generating in this way a new solution \mathbf{S}' (in short we will designate it by $\mathbf{S}' = \tau_l^k(s, t)(\mathbf{S})$). In the generated solution $\mathbf{S}' = (A', \pi')$

$$A' = (A'_1, A'_2, \dots, A'_m), \text{ and } \pi' = (\pi'_1, \pi'_2, \dots, \pi'_m), \text{ where}$$

$$n'_k = n_k - 1, n'_l = n_l + 1, A'_i = A_i \text{ and } \pi'_i = \pi_i, \text{ for } i \neq k, l (k, l \in \mathcal{M}), \quad (13)$$

$$A'_k = A_k \setminus \{\pi_k(s)\}, A'_l = A_l \cup \{\pi_k(s)\},$$

$$\pi'_k = (\pi_k(1), \pi_k(2), \dots, \pi_k(s-1), \pi_k(s+1), \dots, \pi_k(n'_k)),$$

$$\pi'_l = (\pi_l(1) \dots, \pi_l(t-1), \pi_k(s), \pi_l(t), \pi_l(t+1) \dots, \pi_l(n'_l)).$$

It is easy to check that the determined in this way solution $\mathbf{S}' = (A', \pi')$ is feasible for the **MTC** problem. Then we determine the cost of performing tasks for both machines.

After executing the move $\tau_l^k(s, t)(\mathbf{S})$ (transferring the task from machine k to machine l) The cost of executing tasks on machine k

$$\mathcal{T}_k(\mathbf{S}') = \sum_{i=1}^{n'_k} f_{\pi'_k(i)}(\mathbf{S}') = \sum_{i=1}^{s-1} f_{\pi'_k(i)}(\mathbf{S}') + \sum_{i=s}^{n'_k} f_{\pi'_k(i)}(\mathbf{S}'). \quad (14)$$

From definition of permutation π'_k the first sum $\sum_{i=1}^{s-1} f_{\pi'_k(i)}(\mathbf{S}') = \sum_{i=1}^{s-1} f_{\pi_k(i)}(\mathbf{S})$. Since permutation π'_k is created from π_k by removing the task $\pi_k(s)$, then the execution by the k -th machine the tasks $\pi'_k(s), \pi'_k(s+1), \dots, \pi'_k(n'_k)$ in solution \mathbf{S}' is exceeded by the $p_{\pi_k(s)}$ (this is $\pi_k(s)$). Therefore, the moment of completion of tasks execution $C_{\pi'_k(i)} = C_{\pi_k(i+1)} - p_{\pi_k(s)}$, and the cost

$$f_{\pi'_k(i)}(\mathbf{S}') = w_{\pi_k(i+1)} \cdot \max\{0, C_{\pi_k(i+1)} - p_{\pi_k(s)} - d_{\pi_k(i+1)}\}, \quad (15)$$

for $i = s, s+1, \dots, n'_k$. Ultimately, using (15) the cost of execution of tasks by k -th machine in the order π'_k

$$\mathcal{T}_k(\mathbf{S}') = \sum_{i=1}^{s-1} f_{\pi_k(i)}(\mathbf{S}) + \sum_{i=s+1}^{n_k-1} w_{\pi_k(i+1)} \max\{0, C_{\pi_k(i+1)} - p_{\pi_k(s)} - d_{\pi_k(i+1)}\}. \quad (16)$$

Similarly, for the l -th machine, the cost of execution of tasks on machine l

$$\mathcal{T}_l(\mathbf{S}') = \sum_{i=1}^{n'_l} f_{\pi'_l(i)}(\mathbf{S}') = \sum_{i=1}^{t-1} f_{\pi'_l(i)}(\mathbf{S}') + f_{\pi'_l(t)}(\mathbf{S}') + \sum_{i=t+1}^{n'_l} f_{\pi'_l(i)}(\mathbf{S}'). \quad (17)$$

It follows from definition of permutation π'_l that $\sum_{i=1}^{t-1} f_{\pi'_l(i)}(\mathbf{S}') = \sum_{i=1}^{t-1} f_{\pi_l(i)}(\mathbf{S})$, and $f_{\pi'_l(t)}(\mathbf{S}') = w_{\pi_k(s)} \cdot \max\{0, C_{\pi_l(t-1)} + p_{\pi_k(s)} - d_{\pi_k(s)}\}$. The

moment of completing each task $\pi'_l(t+1), \pi'_l(t+2), \dots, \pi'_l(n'_l)$ is transferred (relative to its execution time in π_k) by the time of the duration of the task $\pi_k(s)$, namely by $p_{\pi_k(s)}$. Therefore, similarly as above

$$\sum_{i=t+1}^{n'_l} f_{\pi'_l(i)}(\mathbf{S}') = \sum_{i=t}^{n_l} w_{\pi_k(i)} \cdot \max\{0, C_{\pi_l(i)} + p_{\pi_k(s)} - d_{\pi_l(i)}\}.$$

Ultimately

$$\begin{aligned} \mathcal{T}_l(\mathbf{S}') &= \sum_{i=1}^{t-1} f_{\pi_l(i)}(\mathbf{S}) + w_{\pi_k(s)} \cdot \max\{0, C_{\pi_l(t-1)} + p_{\pi_k(s)} - d_{\pi_k(s)}\} \\ &\quad + \sum_{i=t}^{n_l} w_{\pi_k(i)} \cdot \max\{0, C_{\pi_l(i)} + p_{\pi_k(s)} - d_{\pi_l(i)}\}. \end{aligned} \tag{18}$$

For the solution $\mathbf{S} \in \mathcal{QP}$, let

$$\begin{aligned} \delta_{\mathbf{S}}^- &= \sum_{i=s+1}^{n_k-1} w_{\pi_k(i)} \cdot \max\{0, C_{\pi_k(i+1)} - p_{\pi_k(s)} \\ &\quad - d_{\pi_k(i+1)}\} - f_{\pi_k(s)}(\mathbf{S}) + \sum_{i=s+1}^{n_k} f_{\pi_k(i)}(\mathbf{S}). \end{aligned} \tag{19}$$

Lemma 1. *If the solution $\mathbf{S}' = \tau_l^k(s, t)(\mathbf{S})$, then the cost (14) of tasks execution on k -th machine*

$$\mathcal{T}_k(\mathbf{S}') = \mathcal{F}_k(\mathbf{S}) + \delta_{\mathbf{S}}^-,$$

Proof. From definition the cost of tasks execution

$$\begin{aligned} \mathcal{T}_k(\mathbf{S}) &= \sum_{i=1}^{s-1} f_{\pi_k(i)}(\mathbf{S}) + f_{\pi_k(s)} + \sum_{i=s+1}^{n_k} f_{\pi_k(i)}(\mathbf{S}). \\ \mathcal{T}_k(\mathbf{S}') &= \mathcal{T}_k(\mathbf{S}) + \delta_{\mathbf{S}}^- = \sum_{i=1}^{s-1} f_{\pi_k(i)}(\mathbf{S}) + f_{\pi_k(s)} + \sum_{i=s+1}^{n_k} f_{\pi_k(i)}(\mathbf{S}) + \\ &+ \sum_{i=s+1}^{n_k-1} w_{\pi_k(i)} \cdot \max\{0, C_{\pi_k(i+1)} - p_{\pi_k(s)} - d_{\pi_k(i+1)}\} - (f_{\pi_k(s)}(\mathbf{S}) + \sum_{i=s+1}^{n_k} f_{\pi_k(i)}(\mathbf{S})) \\ &= \sum_{i=1}^{s-1} f_{\pi_k(i)}(\mathbf{S}) + \sum_{i=s+1}^{n_k-1} w_{\pi_k(i)} \cdot \max\{0, C_{\pi_k(i+1)} - p_{\pi_k(s)} - d_{\pi_k(i+1)}\}. \end{aligned}$$

The last equality follows from the formula (16). A similar lemma will be proven for l -th machine. Let

$$\begin{aligned} \delta_{\mathbf{S}}^+ &= w_{\pi_l(s)} \cdot \max\{0, C_{\pi_l(t-1)} + p_{\pi_k(s)} - d_{\pi_k(s)}\} \\ &+ \sum_{i=t}^{n_l} w_{\pi_k(i)} \cdot \max\{0, C_{\pi_l(i)} + p_{\pi_k(s)} - d_{\pi_l(i)}\} - \sum_{i=t}^{n_l} f_{\pi_l(i)}(\mathbf{S}). \end{aligned} \tag{20}$$

Lemma 2. *If the solution $\mathbf{S}' = \tau_l^k(s, t)(\mathbf{S})$, then the cost (17) of tasks execution by l -th machine*

$$\mathcal{T}_l(\mathbf{S}') = \mathcal{T}_l(\mathbf{S}) + \delta_{\mathbf{S}}^+$$

Proof. Similarly as in the case of the proof of the Lemma 1

$$\mathcal{T}_l(\mathbf{S}) = \sum_{i=1}^{t-1} f_{\pi_k(i)}(\mathbf{S}) + \sum_{i=t}^{n_k} f_{\pi_k(i)}(\mathbf{S}).$$

Then, using equality (18) and definition $\delta_{\mathbf{S}}^+$ we obtain

$$\begin{aligned} \mathcal{T}_l(\mathbf{S}') &= \mathcal{T}_l(\mathbf{S}) + \delta_{\mathbf{S}}^+ = \sum_{i=1}^{t-1} f_{\pi_l(i)}(\mathbf{S}) + \sum_{i=t}^{n_l} f_{\pi_t(i)}(\mathbf{S}) + \\ &+ w_{\pi_{\pi_l}(s)} \cdot \max\{0, C_{\pi_l(t-1)} + p_{\pi_k(s)} - d_{\pi_k(s)}\} + \\ &+ \sum_{i=t}^{n_l} w_{\pi_k(i)} \cdot \max\{0, C_{\pi_l(i)} + p_{\pi_k(s)} - d_{\pi_l(i)}\} - \sum_{i=t}^{n_l} f_{\pi_l(i)}(\mathbf{S}). \end{aligned}$$

Let

$$\Delta_l^k(s, t)(\mathbf{S}) = \delta_{\mathbf{S}}^- + \delta_{\mathbf{S}}^+. \tag{21}$$

The value of this expression can be determined in time $O(n)$. It will be used to estimate the effectiveness (change of the criterion value) of the *transfer*-type move. It will allow us to determine the best solution from the neighborhood.

Theorem 3. *If \mathbf{s} is a solution to the MTC problem and \mathbf{S}' was generated from \mathbf{S} by the transfer-type move $\tau_l^k(s, t)$, then the cost of tasks execution*

$$\mathcal{T}(\mathbf{S}') = \mathcal{T}(\mathbf{S}) + \Delta_l^k(s, t)(\mathbf{S}). \tag{22}$$

Proof. The proof results directly from the Lemma 1, 2 and the definition (22).

The value of the expression (22) can be computed in time $O(n)$. It will be used to quickly compute the value of the solution criteria generated by the *transfer*-type of move.

4 Elimination of Solutions

Here we will present some properties of the MTC problem that allow one to eliminate 'the worse' solutions from the neighborhood. This will reduce the number of elements in the neighborhood, and thus the time it takes to search.

Let \mathbf{S} be some solution, and l some machine. For task v ($v \notin A_l$) transferred to machine l

$$\lambda_l(v) = \max\{j : C_{\pi_i(j-1)} + p_v > d_v, 1 \leq j \leq n_i\} - 1 \tag{23}$$

is the maximum number of positions in the permutation π_l on which the task v was transferred if not delayed (its penalty equals to 0). In definition (23) there is an assumption that $C_{\pi_i(0)} = 0$. For a fixed machine l and task $v \notin A_l$ parameter (23) can be determined in time $O(n)$. We will prove the theorem enabling the elimination of certain solutions from the neighborhood generated by transfer-type of moves.

Theorem 4. *Let $\mathbf{S}(A, \pi)$ be some solution to MTC problem. For any pair of moves $\tau_l^k(r, s)$ and $\tau_l^k(r, t)$, where $k \neq l$, $k, l \in \mathcal{M}$, $1 \leq s < t \leq \lambda_k(\pi_k(r))$ there is*

$$\mathcal{T}(\tau_l^k(r, s)(\mathbf{S})) \geq \mathcal{T}(\tau_l^k(r, t)(\mathbf{S})). \tag{24}$$

Proof. In order to simplify the notation, in the proof of the theorem we assume that the order of execution of tasks on the l -th machine is h element identity permutation $\pi = (1, 2, \dots, h)$, and $\mathcal{T}(\pi)$ is the cost of tasks execution on l -th machine in the order π (this is equivalent to the definition (3)). Further, by π^i ($1 \leq i \leq h$) we denote the permutation resulting from π by inserting into position i tasks $v = \pi_k(r)$, this is

$$\pi^i = (1, 2, \dots, i = 1, v, i, i + 1, \dots, h, \eta), \text{ where } \eta = h + 1.$$

We are considering two positions s and t ($1 \leq s < t \leq \lambda_l(v)$) in permutation π . By inserting the task v to permutation π respectively to position s and t we generate two permutations (order of tasks execution by machine l) π^s and π^t .

We are now calculating the cost of tasks execution by the machine l respectively for the order π^s and π^t .

$$\mathcal{T}(\pi^s) = \sum_{i=1}^{s-1} f_i(C_{\pi^s(i)}) + f_s(C_{\pi^s(s)}) + \sum_{i=s+1}^t f_i(C_{\pi^s(i)}) + \sum_{i=t+1}^{\eta} f_i(C_{\pi^s(i)}). \tag{25}$$

Similarly, for permutation π^t

$$\mathcal{T}(\pi^t) = \sum_{i=1}^{s-1} f_i(C_{\pi^t(i)}) + \sum_{i=s+1}^{t-1} f_i(C_{\pi^t(i)}) + f_t(C_{\pi^t(t)}) + \sum_{i=t+1}^{\eta} f_i(C_{\pi^t(i)}). \tag{26}$$

From definition of both permutations

$$\pi^s(i) = \pi^t(i) \text{ and } C_s(i) = C_t(i), \text{ for } i = 1, 2, \dots, s - 1, t + 1, t + 2, \dots, \eta.$$

Therefore, in expressions (25) and (26) we have the equality of components:

$$\sum_{i=1}^{s-1} f_i(C_{\pi^s(i)}) = \sum_{i=1}^{s-1} f_i(C_{\pi^t(i)}) \text{ and } \sum_{i=t+1}^{\eta} f_i(C_{\pi^s(i)}) = \sum_{i=t+1}^{\eta} f_i(C_{\pi^t(i)}).$$

Since $\pi^s(s) = v$ and the task v is not delayed (by assumption of the theorem $s \leq \lambda_l(v)$), thus $f_s(C_{\pi^s(s)}) = 0$. Similarly $f_t(C_{\pi^t(t)}) = 0$. The proof of the

theorem thesis comes down to showing that $\sum_{i=s+1}^t f_i(C_{\pi^s(i)}) \geq \sum_{i=s}^{t-1} f_i(C_{\pi^t(i)})$. The tasks $s, s + 1, \dots, t - 2, t - 1$ are placed in permutation π^t to positions $s, s + 1, \dots, t - 2, t - 1$. In turn, in permutation π^s they are on positions $s + 1, s + 2, \dots, t - 1, t$. This shift to the right by one position is due to the fact that in π^s is preceded by the task v .

Since $C_{\pi^s(i)} = C_{\pi^t(i-1)} + p_v$, it's easy to show that

$$f_i(C_{\pi^s(i)}) \geq f_i(C_{\pi^t(i-1)}) \text{ for } i = s + 1, s + 2, \dots, t.$$

Summing up these inequalities in sides, we finally get $\sum_{i=s+1}^t f_i(C_{\pi^s(i)}) \geq \sum_{i=s}^{t-1} f_i(C_{\pi^t(i)})$, which ends the proof of the theorem.

Using the above theorem, it is easy to prove the properties that allow one to eliminate certain transfer-type moves generating worse solutions from the neighborhood.

Property 1. (Elimination of solutions). If \mathbf{S} is solution to **MTC** problem, k and l are machine numbers, and $\pi_k(s)$ a task transferred from the machine k to l by transfer-type move, then

$$\mathcal{T}(\tau_l^k(s, 1)(\mathbf{S})) \geq \mathcal{T}(\tau_l^k(s, 2)(\mathbf{S})) \geq \dots \geq \mathcal{T}(\tau_l^k(s, \lambda_l(s))(\mathbf{S})).$$

Therefore, by designating the elements of the neighborhood for each task $\pi_k(s) \notin A_k$ it is possible to omit the moves from the set

$$\mathcal{R}_l^k(s) = \{\tau_l^k(s, 1)(\mathbf{S}), \tau_l^k(s, 2)(\mathbf{S}), \dots, \tau_l^k(s, \lambda_l(s) - 1)(\mathbf{S})\}. \tag{27}$$

They generate solutions no better (of not lower value than the penalty function) than the move $\tau_l^k(s, \pi_l(\lambda_l(s)))(\mathbf{S})$. This move will be called a *representative* of the moves from the set (27). Then

$$\mathcal{R}(\mathbf{S}) = \bigcup_{i=1}^m \bigcup_{\substack{j=1, \\ j \neq i}}^m \bigcup_{s=1}^{n_j} \mathcal{R}_j^i(\mathbf{S}). \tag{28}$$

Is the set of all the *transfer-type* of moves, which can be omitted.

5 Solution Method

In order to solve the problem considered at work there was the tabu search (**TS**) algorithm used for solving the single-machine problem $1|| \sum w_i T_i$, adopted from the work of Bożejko et al. [2]. The essential component of the algorithm based on this method is a neighborhood – subset of the set of all solutions. In each iteration of the algorithm, the element with the minimum value of the criterion function is determined from the neighborhood. Generation procedure and searching the neighborhood have a decisive influence on the values of the determined solutions, the pace of convergence and time calculations. Below we

describe the method of generating and searching the neighborhood in algorithm solving the **MTC** problem.

Let $\mathcal{I}(\mathbf{S})$ and $\Theta(\mathbf{S})$ be respectively, the set of all insert and transfer-type of moves for a certain solution $\mathbf{S} \in \mathcal{AP}$. Neighborhood \mathbf{S} is the set of solutions

$$\mathcal{N}(\mathbf{S}) = \{\lambda(\mathbf{S}) : \lambda \in \mathcal{I}(\mathbf{S}) \cup \Theta(\mathbf{S})\}. \quad (29)$$

If by generating the neighborhood $\mathcal{N}(\mathbf{S})$ 'the worse' moves will be removed, i.e. moves from the set $\mathcal{R}(\mathbf{S})$ then we obtain a subneighborhood $\mathcal{N}_{sub}(\mathbf{S})$. When determine an element from the neighborhood, we also omit moves whose attributes are on the so-called tabu list.

For the $\mathbf{S} \in \mathcal{AP}$ solution, generating and searching the neighborhood is performed in two steps:

Step 1. Designate the best solution from the set generated by *insert*-type of moves, i.e. from the set of moves $\mathcal{I}(\mathbf{S})$.

Step 2. Designate the best solution from the set generated by *transfer*-type of moves, i.e. from the set of moves $\Theta(\mathbf{S})$.

If the solution in Step 1 is better than the \mathbf{S} solution, then Step 2 is omitted.

In the description of the algorithm \mathbf{S} is a starting solution, and \mathbf{S}^* is the best designated solution.

Tabu Search (TS) algorithm

```

 $\mathbf{S}^* := \mathbf{S}$ ;
repeat
  Determine solution  $\mathbf{S}'$  such that  $\mathcal{T}(\mathbf{S}') = \min\{\mathcal{T}(\lambda(\mathbf{S})) : \lambda \in \mathcal{I}(\mathbf{S})\}$ 
  omitting moves whose attributes are on the tabu list;
  if (  $\mathcal{T}(\mathbf{S}') < \mathcal{T}(\mathbf{S}^*)$  ) then  $\mathbf{S}^* := \mathbf{S}'$ ;  $\mathbf{S} := \mathbf{S}'$ ; else
  begin
    Determine solution  $\mathbf{S}''$  such that  $\mathcal{T}(\mathbf{S}'') = \min\{\mathcal{T}(\lambda(\mathbf{S})) : \lambda \in \Theta(\mathbf{S})\}$ 
    omitting moves whose attributes are on the tabu list;
    if (  $\mathcal{T}(\mathbf{S}'') < \mathcal{T}(\mathbf{S}^*)$  ) then  $\mathbf{S}^* := \mathbf{S}''$ ;  $\mathbf{S} := \mathbf{S}''$ ;
  end{else}
  Update move's attributes on a tabu list;
until (stop_condition).

```

The above algorithm with subneighborhood $\mathcal{N}_{sub}(\mathbf{S})$ will be denoted by \mathbf{TS}_{sub} .

6 Computational Experiments

Variations of the problem are considered in the literature, e.g. [1,8], but the sizes of test data used do not correspond to the contemporary requirements of industrial practice ($n = 18$, $m = 4$ in [8] (B&B), $n = 500$, $m = 10$ in [1] (heuristics only). Therefore, we decided to propose new test examples of large

sizes ($n \leq 1000, m \leq 20$). Therefore the test examples were generated similarly to those described in the work by Potts and Van Wassenhove [12]: parameters of each the tasks $i \in \mathcal{J}$ are the implementation of random variables of uniform distribution. The execution time p_i - on the interval $[1, 100]$, the weight of the penalty function w_i - on the interval $[1, 10]$, and the latest completion date d_i - on the interval $[P(1 - TF - RDD/2), P(1 - TF + RDD/2)]$, $P = \lceil (\sum_{i=1}^n p_i) / m + 1 \rceil$, $RDD, TF \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$.

For each of 25 pairs of RDD and TF parameter values there were 5 examples generated. So, for each pair n ($n = 50, 100, 500, 1000$) and m ($m = 2, 5, 10, 20$), there were 125 examples of varying difficulty determined - a total of 2,000 examples was reached.

The percentage relative deviation (PRD) was used to evaluate the algorithms: $\delta = \frac{T_{ref} - T_{alg}}{T_{ref}} \cdot 100\%$, where: T_{ref} - value of the reference solution, T_{TS} - the value of the solution designated by tested algorithm. There are no reference data in the literature. Therefore, there will be a comparison of the solutions determined by the **TS** algorithm and the results of a constructive algorithm.

Constructive GRC algorithm

```

 $n_k = 0; \pi_k := (), k = 1, 2, \dots, m;$ 
Number tasks in such a way that  $p_1/w_1 \geq p_2/w_2 \geq \dots \geq p_n/w_n;$ 
for  $l := 1$  to  $n$  do begin
    Determine machine  $k$  such that  $\mathcal{T}(\pi_k) = \min\{\mathcal{T}(\pi_i) : i = 1, 2, \dots, m\};$ 
    Insert the task  $l$  on position in permutation  $\pi_k$  so that the penalty
        for the tasks execution tardiness on machine  $k$  was minimal;
end

```

The computational complexity of the algorithm is $O(mn^2)$.

Algorithms **GRC** and **TS** were programmed in C++ language and run on a personal computer with an Intel processor Core i7 3.5 GHz. First, the effectiveness of the **GRC** algorithm was tested, the solutions of which will be the reference when evaluating the results of the **TS** algorithm. In the work of Bożejko et al. [2] presents the tabu search algorithm for solving the $1||\sum w_i T_i$ problem and the results of computational experiments. On the set of these examples, the mean relative error of the **GRC** algorithm is 13.7%.

On the basis of the preliminary calculations of the **TS** algorithm, the following parameter values were determined: the length of the tabu list $L_{LT} = 11$, the maximum number of iterations $L_{iter} = 2n$. The main calculations were made on the examples described at the beginning of this section. The mean relative deviation for each group of examples is presented in Table 1.

This error (the average relative improvement of the solutions determined by the **GRC** algorithm) for all computed examples is 10.4%. The lowest value 7.2% is for $n = 20$ and $m = 2$ and it grows with increasing number of tasks and machines. In general, the average errors do not vary much. This is due to the fact that the P parameter, on which the sizes of the intervals from which

Table 1. The mean relative deviation of the **TS** algorithm in reference to **GRC**.

n	$m = 2$	$m = 5$	$m = 10$	$m = 20$
20	7.2	7.8	9.4	12.7
50	8.0	7.9	8.6	12.6
100	7.9	8.2	9.8	11.3
500	9.3	9.1	1.5	14.1
1000	11.4	12.0	11.6	16.5
Average	8.7	9.0	10.3	13.4

the critical lines d_i are drawn, depends on the number of machines. The total computation time for all 2000 examples was 35.5 min.

The mean relative error of the **TS_{sub}** algorithm is 10.3%. Hence, it is almost identical to the **TS** algorithm. However, the total computation time decreased significantly by 21.5%. Thus, the use of elimination properties in the construction of the algorithm resulted in a significant improvement.

7 Comments and Conclusions

In the paper there was an NP-hard task scheduling problem considered with the earliest requested completion dates. The tasks should be assigned to machines and the order of their execution should be determined so that the sum of penalties for delays is minimal. Quick methods of calculating the value of the criterion function and properties have been introduced to enable the elimination of ‘worse’ solutions. The conducted computational experiments have shown that the use of elimination properties in the construction of the tabu search algorithm significantly reduces the computation time.

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

References

1. Alidaee, B., Rosa, D.: Scheduling parallel machines to minimize total weighted and unweighted tardiness. *Comput. Oper. Res.* **24**(8), 775–788 (1997)
2. Bożejko, W., Grabowski, J., Wodecki, M.: Block approach tabu search algorithm for single machine total weighted tardiness problem. *Comput. Ind. Eng.* **50**(1–2), 1–14 (2006)
3. Bożejko, W., Uchroński, M., Wodecki, M.: Blocks for two-machines total weighted tardiness flow shop scheduling problem. *Bull. Pol. Acad. Sci. Tech. Sci.* **68**(1), 31–41 (2020)
4. Bulfin, R.L., Hallah, R.: Minimizing the weighted number of tardy tasks on two-machine flow shop. *Comput. Oper. Res.* **30**, 1887–1900 (2003)

5. Karabulut, K.: A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Comput. Ind. Eng.* **98**, 300–307 (2016)
6. Kayvanfar, V., Komaki, G.H.M., Aalaei, A., Zandieh, M.: Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times. *Comput. Oper. Res.* **41**, 31–43 (2014)
7. Lenstra, J.K., Rinnooy Kan, A.G.H., Brucker, P.: Complexity of machine scheduling problems. *Ann. Discrete Math.* **1**, 343–362 (1977)
8. Liaw, C.-F., Lin, Y.-K., Cheng, C.-Y., Chen, M.: Scheduling unrelated parallel machines to minimize total weighted tardiness. *Comput. Oper. Res.* **30**(12), 1777–1789 (2003)
9. Liu, C.: A hybrid genetic algorithm to minimize total tardiness for unrelated parallel machine scheduling with precedence constraints. *Math. Probl. Eng.* (2013). ID 537127
10. Park, J.M., Choi, B.C., Min, Y., Kim, K.M.: Two-machine ordered flow shop scheduling with generalized due dates. *Asia-Pac. J. Oper. Res.* **37**(01), 1950032 (2020)
11. Ojstersek, R., Tang, M., Buchmeister, B.: Due date optimization in multi-objective scheduling of flexible job shop production. *Adv. Prod. Eng. Manag.* **15**(4), 481–492 (2020)
12. Potts, C.N., Van Wassenhove, L.N.: A decomposition algorithm for the single machine total tardiness problem. *Oper. Res. Lett.* **1**, 177–181 (1982)
13. Toncovicha, A., Rossit, D., Frutos, M., Rossit, D.G., Daniel, A.: Solving a multi-objective manufacturing cell scheduling problem with the consideration of warehouses using a simulated annealing based procedure. *Int. J. Ind. Eng. Comput.* **10**, 1–16 (2019)
14. Wodecki, M.: A block branch-and-bound parallel algorithm for single-machine total weighted tardiness problem. *Adv. Manuf. Technol.* **37**, 996–1004 (2008)