

Highly efficient scheduling algorithms for identical parallel machines with sufficient conditions for optimality of the solutions

Sergii TELENYK¹, Grzegorz NOWAKOWSKI¹, Oleksandr PAVLOV², Olena MISURA²,
Oleg MELNIKOV², and Olena KHALUS²

¹ Faculty of Electrical and Computer Engineering, Cracow University of Technology, Warszawska 24, 31-155 Cracow, Poland

² National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Prosp. Peremohy 37, Kyiv, Ukraine

Abstract. This paper aims to develop new highly efficient PSC-algorithms (algorithms that contain a polynomial-time sub-algorithm with sufficient conditions for the optimality of the solutions obtained) for several interrelated problems involving identical parallel machine scheduling. These problems share common basic theoretical positions and common principles of their solving. Two main intractable scheduling problems are considered: ("Minimization of the total tardiness of jobs on parallel machines with machine release times and a common due date" (TTPR) and "Minimising the total tardiness of parallel machines completion times with respect to the common due date with machine release times" (TTCR)) and an auxiliary one ("Minimising the difference between the maximal and the minimal completion times of the machines" (MDMM)). The latter is used to efficiently solve the first two ones. For the TTPR problem and its generalisation in the case when there are machines with release times that extend past the common due date (TTPRE problem), new theoretical properties are given, which were obtained on the basis of the previously published ones. Based on the new theoretical results and computational experiments the PSC-algorithm solving these two problems is modified (sub-algorithms A1, A2). Then the auxiliary problem MDMM is considered and Algorithm A0 is proposed for its solving. Based on the analysis of computational experiments, A0 is included in the PSC-algorithm for solving the problems TTPR, TTPRE as its polynomial component for constructing a schedule with zero tardiness of jobs if such a schedule exists (a new third sufficient condition of optimality). Next, the second intractable combinatorial optimization problem TTCR is considered, deducing its sufficient conditions of optimality, and it is shown that Algorithm A0 is also an efficient polynomial component of the PSC-algorithm solving the TTCR problem. Next, the case of a schedule structure is analysed (partially tardy), in which the functionals of the TTPR and TTCR problems become identical. This facilitates the use of Algorithm A1 for the TTPR problem in this case of the TTCR problem. For Algorithm A1, in addition to the possibility of obtaining a better solution, there exists a theoretically proven estimate of the deviation of the solution from the optimum. Thus, the second PSC-algorithm solving the TTCR problem finds an exact solution or an approximate solution with a strict upper bound for its deviation from the optimum. The practicability of solving the problems under consideration is substantiated.

Keywords: scheduling; intractable problems of discrete optimisation; sufficient conditions for optimality; parallel machines; common due date.

1. INTRODUCTION

Single-stage scheduling problems, which are generally NP-hard, are widely used in the creation of efficient planning and control systems for modern productions with discrete technological processes [1–6]. Many single-stage scheduling problems are solved today with the help of a new theory and practice of creating PSC-algorithms. These are the algorithms that necessarily include sufficient conditions (signs) of optimality (SCOs) for a feasible solution, which can be verified only at the stage in which a feasible solution is built using a polynomial-time algorithm(s). These form the first polynomial component of the PSC-algorithm. The second polynomial component of the PSC-algorithm is an approximation algorithm with polynomial

complexity, which solves the problem in the case where the first polynomial component cannot obtain a feasible solution satisfying a sufficient sign of optimality. A PSC-algorithm for binary or unary NP-hard combinatorial optimisation problems may include an exact solution algorithm for the case where sufficient conditions are found, and satisfying these conditions at the execution stage turns it into a polynomial complexity algorithm [7, 8].

The intractable single-stage scheduling problem is considered, involving the minimisation of the total tardiness of jobs on parallel machines with machine release times and a common due date (the TTPR problem).

TTPR problem statement [9, 10]. We have a set of n jobs $J = \{1, 2, \dots, n\}$ and m identical parallel machines, and we know the processing time p_j for each job $j \in J$. All jobs have a common due date d . A machine i , $i = 1, m$, can start to process any job from the set J after its release time r_i , $0 \leq r_i < d$, where the release times for jobs may be not the same. Machine idle

*e-mail: gnowakowski@pk.edu.pl

Manuscript submitted 2023-05-25, revised 2023-11-26, initially accepted for publication 2023-12-18, published in February 2024.

times are forbidden. The aim is to build a schedule σ for the jobs $j \in J$ on m machines that minimises the functional

$$F(\sigma) = \sum_{j \in J} \max(0, c_j(\sigma) - d),$$

where $c_j(\sigma)$ is the completion time of job j on schedule σ .

This problem has been noted as being NP-hard [11]. It can be solved in a pseudo-polynomial time if $m = 2$ [12].

It was shown in [9] that the TTPR problem can be solved with a PSC-algorithm [7, 10] designed for the problem variant with equal machine release times (the TTP problem, involving the minimisation of the total tardiness of jobs on identical parallel machines). The work in [10] describes PSC-algorithms for the TTP and TTPR problems and explores their efficiency using a statistical approach. Other papers [7, 13] propose a generalised form of the TTPR problem for the case where the machine release times may *exceed* the common due date (the TTPRE problem).

The TTP problem is often considered in the development of production planning systems, project management, construction management, and other areas. Such problems are characterised by severe penalties for due date violations [10]. Criteria based on minimizing job tardiness are among the most important in scheduling, and researchers pay close attention to them [14–17]. At the same time, taking into account the NP-hardness of the considered problems, preference is often given to non-exact heuristic methods [14, 16, 17]. PSC-algorithms serve as a more accurate and efficient alternative.

Considering various release times of machines in the problem formulation allows using this problem in multi-stage planning taking into account partially completed schedules (that can have arbitrary fixed machine release times). Thus, the practical significance of this work results lies in the creation of methods for solving such single-stage scheduling problems which will significantly expand the scope of practical application of existing hierarchical scheduling and operative planning models in discrete manufacturing systems, as well as offer new efficient multi-level planning models for systems with a network representation of technological processes. New mathematics and control software for such systems can be included in universal integrated information technologies to support the efficient functioning of discrete manufacturing systems.

In this article, new, highly efficient PSC-algorithms are developed for several interrelated problems involving identical parallel machine scheduling. These problems share common basic theoretical positions and common principles of their solving. New theoretical properties are obtained for the TTPR and TTPRE problems to find efficient modifications to the known PSC-algorithms for their solutions. These modifications include the following:

- Introducing an additional SCO for a schedule, which is optimal if all jobs in it are not tardy (we call such a schedule *feasible*).
- Embedding a polynomial-time approximation sub-algorithm that efficiently finds a feasible schedule if it exists for the given problem instance.

Based on the results obtained from this approach, a new, efficient PSC-algorithm is developed to solve a previously unconsidered intractable single-stage scheduling problem that involves minimising the total workload of the parallel machines after the common due date (the total tardiness of their completion times) with machine release times (TTCR).

Outline of the paper. Section 2 provides a brief description of the PSC-algorithm for solving the TTPR problem. This consists of sub-algorithms A1 and A2 [7, 10] which check two SCOs for an arbitrary schedule. New theoretical properties for the TTPR and TTPRE problems are presented and proved, and rigorous proof of the SCOs is provided for schedules obtained using previously published PSC-algorithms [7, 10] for these two problems. A rigorous substantiation for the theoretical properties of the modified algorithms A1 and A2 [7] is given for the TTPRE problem. Furthermore, SCO #3 (where the total tardiness of the schedule is zero) is introduced, and the necessity of an efficient polynomial-time approximation sub-algorithm, which will build a feasible schedule if it exists, is demonstrated. Such an algorithm, referred to here as A0, is presented and substantiated in Section 3. Examples are given in Section 4 to show the efficiency of A0 in terms of satisfying SCO #3 for the TTPRE problem, and the results of some computational studies are also presented. Section 5 outlines the structure of the modified PSC-algorithm for the TTPR and TTPRE problems, which follows from the results presented in the previous sections. All the optimal schedules are divided into four classes, and it is shown that:

- If a schedule belongs to the first or third class, then it is always optimal.
- The introduced SCOs are not satisfied for the second class of schedules.
- If the schedule belongs to the fourth class, then it may satisfy the first and second SCOs.

Finally, Section 6 shows that the proposed polynomial-time approximation sub-algorithm for building a feasible schedule is simultaneously an efficient PSC-algorithm for the new TTCR problem.

Remark 1. A problem is referred to as intractable if it is shown to be NP-hard (unary or binary) or there are currently no exact polynomial-time algorithms for its solution [7, 8].

2. THEORY AND METHODOLOGY FOR SOLVING THE TTPR PROBLEM USING PSC-ALGORITHMS

2.1. Known and new theoretical properties of the TTPR problem and the PSC-algorithm for its solution

This section will use content from [9, 10]. Proofs of theorems and statements that are not given here can be found in [7, 9, 10].

Let C_i denote the completion time for machine i , i.e. the point at which it is ready to process jobs. C_i is equal to the completion time of the jobs assigned to machine i (or to the release time r_i before the assignment of jobs).

Algorithm A^{init} for initial TTPR schedule building is given in [9, 10] as follows:

1. Renumber the jobs of the set J in non-decreasing order of processing times.

2. Renumber the machines in non-decreasing order of release times r_i .
3. Initialise the completion times of machines: $C_i = r_i \forall i = \overline{1, m}$.
4. Select an unassigned job j with a minimal processing time p_j . Assign the job j to a machine i that has a minimal completion time C_i .
5. Calculate the new completion time for machine i : $C_i = C_i + p_j$.
6. If all jobs have been assigned, the algorithm terminates. Otherwise, go to Step 4.

The initial schedule obtained in this way is denoted as σ^{init} . The jobs in set J are renumbered in non-decreasing order of p_j and split into (optional non-empty) subsets $J_1, J_2, \dots, J_i, \dots, J_m$, where each pair of subsets does not contain common elements. J_i is the set of jobs processed by machine i , where $i = \overline{1, m}$ [10].

The search for an optimal schedule can be limited [10] to a consideration of schedules in which each machine processes its jobs in the increasing order of their numbers (Theorem 2.3 in Chapter 3 of [18]).

We can split the set of jobs J_i into subsets $P_i(\sigma)$, $S_i(\sigma)$, $Q_i(\sigma)$ as follows [10]:

- $P_i(\sigma)$ is the set of non-tardy jobs on the schedule of machine i .
- $S_i(\sigma)$ is the set that includes the partially tardy job on the schedule of machine i , i.e. job j (if it exists) for which $s_j < d$, $c_j > d$, where s_j is the starting time of job j , $s_j = c_j - p_j$. If such a job j is absent from the schedule of machine i , then $S_i(\sigma) = \emptyset$.
- $Q_i(\sigma)$ is the set of fully tardy jobs on the schedule of machine i , i.e. jobs with $s_j \geq d$, $\forall j \in Q_i(\sigma)$.

We can also define [10]

$$P = \bigcup_{i=\overline{1, m}} P_i; \quad S = \bigcup_{i=\overline{1, m}} S_i; \quad Q = \bigcup_{i=\overline{1, m}} Q_i,$$

where

$$R_i(\sigma) = d - r_i - \sum_{j \in P_i(\sigma)} p_j \text{ is the time reserve of machine } i \text{ for schedule } \sigma.$$

$\Delta_i(\sigma) = \max \left(0, r_i + \sum_{j \in P_i(\sigma) \cup S_i(\sigma)} p_j - d \right)$ is the tardiness of the partially tardy job on machine i regarding the due date (if such a job exists; otherwise, $\Delta_i(\sigma) = 0$).

$|P|$ is the cardinality of set P .

The following theorem was proved in [18] for the case with equal machine release times.

Theorem 1. [10, 18]. There is an optimal schedule that satisfies the conditions:

1. $P \cup S = \{1, 2, \dots, |P \cup S|\}$.
2. If $P \cup S < n$, then $\sum_{j \in P_i(\sigma) \cup S_i(\sigma)} p_j \geq d$, and Q_i contains those and only those elements which differ from $|P \cup S| + i$ by a multiple of m , $i = \overline{1, m}$.

Corollary 1. [18]. A schedule in which each machine $L = \overline{1, m}$ processes jobs in the sequence $L, m + L, 2m + L, \dots$, gives the smallest total completion time for all jobs.

Corollary 2. [18]. Suppose that the processing of jobs on a machine L , $L = \overline{1, m}$, cannot be started earlier than the time $r_L \geq 0$. A schedule in which each next job $k = 1, 2, \dots, n$ is assigned to the machine with the minimal completion time gives the smallest total completion time for all jobs.

Let Ψ_{PS} denote a class of schedules that correspond to the conditions of Theorem 1 [10].

We now demonstrate the truth of Theorem 1 for the case where machines become available at unequal times.

Statement 1. [9]. The schedule σ^{init} for the TTPR problem belongs to the class Ψ_{PS} .

Proof. Consider the schedule σ^{init} . In this case:

- The set of jobs J is divided into m non-overlapping subsets J_i that do not have common elements.
- The jobs on each machine are ordered in non-decreasing order of processing times p_j .
- The sets P , S , and Q satisfy conditions 1 and 2 of Theorem 1 by construction.

Thus, the schedule σ^{init} belongs to the class Ψ_{PS} . $\square$

From the class Ψ_{PS} , we can distinguish a class of schedules $\Psi_P \subseteq \Psi_{PS}$ that satisfy the following additional conditions [10]:

1. $P = \{1, 2, \dots, |P|\}$.
2. $\min_{j \in S(\sigma)} p_j > \max_{i=\overline{1, m}} R_i(\sigma)$.
3. If $p_{j_k} \leq p_{j_l}$, then $s_{j_k} \leq s_{j_l} \forall j_k, j_l \in S(\sigma)$.

Suppose $|P| < n$. Let $P_{\min}$ denote the minimal number of jobs in set P for which $\Psi_P \neq \emptyset$, and let $P_{\max}$ denote the maximal number of jobs in set P [10].

Statement 2. [9]: The schedule σ^{init} for the TTPR problem belongs to the class Ψ_P .

Proof. The schedule σ^{init} satisfies by construction all the above conditions for the class Ψ_P :

- The jobs in set P have numbers $1, 2, \dots, |P|$ (i.e. Condition 1 is true).
 - Condition 2 is met; otherwise, the shortest job from the set S would be included in the set P .
 - Condition 3 is also true by construction of the schedule σ^{init} .
- Thus, the schedule σ^{init} belongs to the class Ψ_P . $\square$

Theorem 2. An optimal schedule for the TTP problem belongs to the class Ψ_P .

Proof. We show that Conditions 1–3 from the definition of class Ψ_P are also true for schedules of the class Ψ_{PS} . Consider the properties of the schedules from class Ψ_{PS} [18]:

1. $P \cup S = \{1, 2, \dots, |P \cup S|\}$. If we consider only the jobs in set P , then this condition is equivalent to $P = \{1, 2, \dots, |P|\}$.
2. The jobs on each machine are in non-decreasing order of p_j (Theorem 2.3 in Chapter 3 of [18]).
3. $d \leq \sum_{k \in P_l(\sigma) \cup S_l(\sigma)} p_k \leq \dots \leq \sum_{k \in P_m(\sigma) \cup S_m(\sigma)} p_k$.

Properties 2 and 3 are only true if each next job from the sequence based on the non-decreasing order of processing times is assigned to the machine with the minimal completion time (i.e. with the maximal reserve). This corresponds to the fulfilment of Conditions 2 and 3 from the definition of the class Ψ_P . Hence,

$\Psi_{PS} \subseteq \Psi_P$, and since $\Psi_P \subseteq \Psi_{PS}$, then we have $\Psi_{PS} = \Psi_P$. According to Theorem 1, an optimal solution belongs to the class Ψ_{PS} , and thus to the class Ψ_P . $\square$

Theorem 3. An optimal schedule for the TTPR problem belongs to the class Ψ_P .

Proof. The structure and all of the properties of the schedules in the TTPR problem are completely analogous to those of the TTP problem and do not depend on the machine release times (this is obvious from Statements 1 and 2). Hence, an optimal schedule for the TTPR problem also belongs to the class Ψ_P , according to Theorem 2. $\square$

It follows clearly from Theorem 3 that the following properties, which were proved in [7, 9, 10] for the TTP problem, are also true for the TTPR problem.

Statement 3. [7, 9, 10]. The following statement is true for all possible schedules $\sigma \in \Psi_P$ built on the set of jobs J :

$$P_{\max} - P_{\min} < m.$$

Statement 4. [7, 9, 10]. Moving jobs as a result of directed permutations is only possible between the sets P and S when building an optimal schedule.

Statement 5. [7, 9, 10]. If a job $j \in P$ is moved from a machine i_k with a greater number of tardy jobs to a machine i_l with a smaller number of tardy jobs, then Δ_{i_k} is decreased by p_j .

Statement 6. [7, 9, 10]. The maximal difference in the number of tardy jobs in schedules $\sigma \in \Psi_P$ does not exceed one.

Definition 1. [7, 9, 10]. A schedule with the same number of tardy jobs on all machines is called an *even schedule*.

Theorem 4. SCO #1 [7, 9, 10]. An even schedule $\sigma \in \Psi_P$ is optimal.

As a notation, let $L_{\max}$ represent the maximal number of tardy jobs on all machines and $L_{\min}$ the minimal number of tardy jobs on all machines. Let the numbers $i = \overline{1, k}$ correspond to the machines with the number of tardy jobs $L_{\max}$. We also define $\Delta_{\Sigma}(\sigma) = \sum_{i=1}^k \Delta_i(\sigma)$ and $R_{\Sigma}(\sigma) = \sum_{i=k+1}^m R_i(\sigma)$; $\Omega_{\Sigma}(\sigma) = \min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma))$ [10].

Theorem 5. [7, 9, 10]. If $L_{\max}$ is the same for schedules $\sigma \in \Psi_P$ and $\sigma' \in \Psi_P$ built on the given set of jobs J , then it follows from $R_{\Sigma}(\sigma) \neq 0$ and $R_{\Sigma}(\sigma') \neq 0$ that $R_{\Sigma}(\sigma) - R_{\Sigma}(\sigma') = \Delta_{\Sigma}(\sigma) - \Delta_{\Sigma}(\sigma')$.

Theorem 6. [7, 9, 10]. For any two schedules $\sigma \in \Psi_P$ and $\sigma' \in \Psi_P$, the relation $F(\sigma) - F(\sigma') = \Omega_{\Sigma}(\sigma) - \Omega_{\Sigma}(\sigma')$ holds true.

Theorem 7. SCO #2 [7, 9, 10]. If $\Omega_{\Sigma}(\sigma) = \min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma)) = 0$ for a schedule $\sigma \in \Psi_P$, then the schedule σ is optimal.

The main characteristic of a schedule $\sigma \in \Psi_P$ is the value of $\Omega_{\Sigma}(\sigma) = \min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma))$, where $\Delta_{\Sigma}(\sigma)$ shows how much the functional value $F(\sigma)$ can be theoretically decreased to give

an optimal schedule. The total reserve $R_{\Sigma}(\sigma)$ represents the reserves that exist for obtaining an optimal schedule [9, 10].

Let σ^* denote an optimal schedule for the set of jobs J .

Theorem 8. [7, 9, 10]. The following expression holds for any schedule $\sigma \in \Psi_P$: $F(\sigma) - F(\sigma^*) \leq \Omega_{\Sigma}(\sigma)$.

Let $I_R(\sigma)$ be the set of machine numbers with a smaller number of tardy jobs for a schedule σ , and let $I_{\Delta}(\sigma)$ be the set of machine numbers with a greater number of tardy jobs for the schedule σ [7, 9, 10].

We now introduce a new class $\Psi(\sigma_P) \subseteq \Psi_{PS}$ that consists of arbitrary schedules σ obtained as results of sequential directed permutations performed in an arbitrary order that decrease $\Delta_{\Sigma}(\sigma)$ and, consequently, $R_{\Sigma}(\sigma)$. These permutations are applied sequentially to the current schedule σ_k , starting with some schedule $\sigma \in \Psi_P$, by moving non-tardy jobs between machines $i_{\Delta}(\sigma_k)$ and $i_R(\sigma_k)$. The order of processing of jobs is not changed on machines other than those mentioned above. A schedule σ_{k+1} is obtained. Each permutation may change the number of tardy jobs only on one machine with a number $i_1 \in I_{\Delta}(\sigma_k)$ and on one machine with a number $i_2 \in I_R(\sigma_k)$. A permutation is forbidden if the number of jobs on the machine i_2 is greater than the number of jobs on the machine i_1 in the schedule σ_{k+1} [7, 9, 10].

Theorem 9. [7, 9, 10]. The following estimate of the deviation of the functional value from the optimum is valid for any schedule $\sigma \in \Psi(\sigma_P)$: $F(\sigma) - F(\sigma^*) \leq \Omega_{\Sigma}(\sigma)$.

Corollary 3. [7, 9, 10]. Theorems 7 and 8 hold for schedules $\sigma \in \Psi(\sigma_P)$.

Hence, the change in the functional value is determined by the values of $\Delta_{\Sigma}(\sigma)$ and $R_{\Sigma}(\sigma)$ for the schedules $\sigma \in \Psi(\sigma_P)$ as well as for the schedules $\sigma \in \Psi_P$ [7, 9, 10].

Theorem 10. [7, 10]. If $\Omega_{\Sigma}(\sigma) = \min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma))$ reaches a minimum for a schedule $\sigma \in \Psi(\sigma_P)$, then the schedule σ is optimal.

Remark 2. Theorem 3 is a rigorous justification of the fact that $\Psi_{PS} = \Psi_P$. This fact was used implicitly in the proofs of Theorems 4–10 (including SCOs #1 and #2) and Statements 3–6 in [7, 10].

The following statements formulate the properties of the jobs $j \in S \cup Q$ in schedules $\sigma \in \Psi(\sigma_P)$.

Statement 7. [7, 9, 10]. Suppose $S'_k(\sigma) \cup Q'_k(\sigma)$ and $S''_l(\sigma) \cup Q''_l(\sigma)$ are the sets of tardy jobs on machines k and l , respectively. We perform a permutation of the sets of tardy jobs between the machines; that is, we move the sets $S''_l(\sigma)$ and $Q''_l(\sigma)$ to machine k and the sets $S'_k(\sigma)$ and $Q'_k(\sigma)$ to machine l . The deviation of the functional value from the optimum in the resulting schedules is determined by $\Omega_{\Sigma}(\sigma) = \min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma))$.

Suppose that the jobs $j \in S \cup Q$ on machines k, l, r in the schedule σ are numbered as follows: $j_k j_{k+m} j_{k+2m} \dots j_l, j_{l+m} j_{l+2m} \dots j_r j_{r+m} j_{r+2m} \dots$ [7, 9, 10].

Definition 2. [7, 9, 10]. The jobs $j_k j_l j_r$, or $j_{k+m} j_{l+m} j_{r+m}, \dots$, or $j_{k+2m} j_{l+2m} j_{r+2m} \dots$ are called the *tardy jobs of the same level*.

Statement 8. [7,9,10]. We can interchange the tardy jobs of the same level on machines with the same number of tardy jobs. The functional value does not change following these permutations.

Statement 9. [7,9,10]. We can always move from a schedule $\sigma \in \Psi(\sigma_P)$ to a schedule $\sigma \in \Psi_{PS}$ with the same functional value permuting the tardy jobs of the same level.

Remark 3. Based on Theorems 3–10 and Statements 3–6, all permutations used in the PSC-algorithm for solving the TTP problem [7, 10] are implemented in the PSC-algorithm for solving the TTPR problem.

2.2. Description of the PSC-algorithm for the TTPR problem solving and obtaining its new theoretical properties

The scheme used for problem solving is as follows [7, 10]:

Stage I. Build an initial schedule $\sigma^{\text{init}} \in \Psi_P$. If $\Omega_\Sigma(\sigma^{\text{init}}) = 0$, then the schedule is optimal, and the process terminates. Otherwise:

- If $R_\Sigma(\sigma^{\text{init}}) \geq \Delta_\Sigma(\sigma^{\text{init}})$, go to Stage II.
- If $R_\Sigma(\sigma^{\text{init}}) < \Delta_\Sigma(\sigma^{\text{init}})$, go to Stage III.

Stage II. Build an even schedule $\sigma \in \Psi(\sigma_P)$ where the number of tardy jobs on each machine is equal to $L(\sigma) = L(\sigma^{\text{init}}) - 1$, where $L(\sigma^{\text{init}})$ is the maximal number of tardy jobs on a machine in the schedule σ^{init} . If such a schedule is found, then it is optimal, and the process terminates; otherwise, go to Stage IV.

Stage III. Build an even schedule $\sigma \in \Psi(\sigma_P)$ where the number of tardy jobs on each machine is equal to $L(\sigma) = L(\sigma^{\text{init}})$.

Stage IV. Build a schedule $\sigma \in \Psi(\sigma_P)$ for which $\Omega_\Sigma(\sigma) < \Omega_\Sigma(\sigma^{\text{init}})$. If $\Omega_\Sigma(\sigma) = 0$, then the schedule is optimal, and the process terminates; otherwise, go to Stage V.

Stage V. Build a schedule $\sigma \in \Psi_{PS}$ for which $\Omega_\Sigma(\sigma) < \Omega_\Sigma(\sigma^{\text{init}})$. The algorithm terminates.

Let $W(\sigma)$ denote an estimate of the deviation from the optimum for schedule σ .

To implement the stages described above within the PSC-algorithm, we use the following types of permutations [10]:

Permutation 1P-0P-Δ. We move a job j from a machine $h \in I_\Delta(\sigma)$ to a machine $r \in I_R(\sigma)$. The job j is such that $p_j \geq \Delta_h(\sigma)$, $p_j \leq R_r(\sigma)$.

Permutation 1P-1P-Δ. We interchange a job j' from machine $h \in I_\Delta(\sigma)$ with a job j'' from machine $r \in I_R(\sigma)$. The jobs j' , j'' are such that $p_{j'} - p_{j''} \geq \Delta_h(\sigma)$, $p_{j'} - p_{j''} \leq R_r(\sigma)$.

We use the permutations given above during the building of an even schedule $\sigma \in \Psi(\sigma_P)$ in Stage II. After each of the permutations, we obtain the schedule σ' , for which $I_\Delta(\sigma') = I_\Delta(\sigma) \setminus \{h\}$; $|P(\sigma')| = |P(\sigma)| + 1$; $\Delta_\Sigma(\sigma') = \Delta_\Sigma(\sigma) - \Delta_h(\sigma)$; $F(\sigma') = F(\sigma) - \Delta_h(\sigma)$.

We apply the following two permutations in Stage III.

Permutation 1P-0P-RΔ. We move a job j from machine $h \in I_\Delta(\sigma)$ to machine $r \in I_R(\sigma)$. The job j is such that $p_j < \Delta_h(\sigma)$, $p_j > R_r(\sigma)$.

Permutation 1P-1P-RΔ. We interchange a job j' from machine $h \in I_\Delta(\sigma)$ with a job j'' from machine $r \in I_R(\sigma)$. The jobs j' , j'' are such that $p_{j'} - p_{j''} < \Delta_h(\sigma)$, $p_{j'} - p_{j''} > R_r(\sigma)$.

After each of these permutations, we obtain the schedule σ' for which $I_\Delta(\sigma') = I_\Delta(\sigma) \cup \{r\}$; $|P(\sigma')| = |P(\sigma)| - 1$; $\Delta_\Sigma(\sigma') = \Delta_\Sigma(\sigma) - R_r(\sigma)$; $F(\sigma') = F(\sigma) - R_r(\sigma)$.

We apply the following two permutations in Stage IV.

Permutation 1P-0P-R. We move a job j from machine $h \in I_\Delta(\sigma)$ to machine $r \in I_R(\sigma)$. The job j is such that $p_j \leq \Delta_h(\sigma)$, $p_j \leq R_r(\sigma)$. After this permutation, we obtain the schedule σ' for which $I_\Delta(\sigma') = I_\Delta(\sigma) \setminus \{h\}$ if $p_j = \Delta_h(\sigma)$ or $I_\Delta(\sigma') = I_\Delta(\sigma)$ if $p_j < \Delta_h(\sigma)$; $R_\Sigma(\sigma') = R_\Sigma(\sigma) - p_j$; $F(\sigma') = F(\sigma) - p_j$.

Permutation 1P-1P-R. We interchange a job j' from machine $h \in I_\Delta(\sigma)$ with a job j'' from machine $r \in I_R(\sigma)$. The jobs j' , j'' are such that $p_{j'} - p_{j''} \leq \Delta_h(\sigma)$, $p_{j'} - p_{j''} \leq R_r(\sigma)$. After this permutation, we obtain a schedule σ' for which $I_\Delta(\sigma') = I_\Delta(\sigma) \setminus \{h\}$ if $p_{j'} - p_{j''} = \Delta_h(\sigma)$ or $I_\Delta(\sigma') = I_\Delta(\sigma)$ if $p_{j'} - p_{j''} < \Delta_h(\sigma)$; $R_\Sigma(\sigma') = R_\Sigma(\sigma) - (p_{j'} - p_{j''})$; $F(\sigma') = F(\sigma) - (p_{j'} - p_{j''})$.

We apply the following three permutations at Stage V to obtain the schedule $\sigma \in \Psi_{PS}$. Jobs in the two sets P and S are considered in these permutations, which are applied to decrease the value of $W(\sigma)$.

Permutation 1P-1S-W1. We interchange a job $j' \in P_r(\sigma)$, $r \in I_R(\sigma)$ with a job $j'' \in S_h(\sigma)$, $h \in I_\Delta(\sigma)$. The jobs j' , j'' are such that $p_{j'} \leq p_{j'} + R_r(\sigma)$ and $p_{j'} < R_h(\sigma)$. After this permutation, we obtain a schedule σ' for which $I_\Delta(\sigma') = I_\Delta(\sigma) \setminus \{h\}$; $F(\sigma') = F(\sigma) - (p_{j''} - R_h(\sigma))$; $W(\sigma') = W(\sigma) - (p_{j''} - R_h(\sigma))$.

Permutation 1P-1S-W2. We interchange a job $j' \in P_r(\sigma)$, $r \in I_R(\sigma)$ with a job $j'' \in S_h(\sigma)$, $h \in I_\Delta(\sigma)$. The jobs j' , j'' are such that $p_{j''} \leq p_{j'} + R_r(\sigma)$ and $p_{j'} > R_h(\sigma)$. After this permutation, we obtain a schedule σ' for which $I_\Delta(\sigma') = I_\Delta(\sigma)$; $F(\sigma') = F(\sigma) - (p_{j''} - p_{j'})$; $W(\sigma') = W(\sigma) - (p_{j''} - p_{j'})$.

Permutation 1P-1S-W3. We interchange a job $j' \in P_r(\sigma)$, $r \in I_R(\sigma)$ with a job $j'' \in S_h(\sigma)$, $h \in I_\Delta(\sigma)$. The jobs j' , j'' are such that $p_{j''} > p_{j'} + R_r(\sigma)$ and $p_{j'} < R_h(\sigma)$. After this permutation, we obtain a schedule σ' for which $I_\Delta(\sigma') = I_\Delta(\sigma)$; $F(\sigma') = F(\sigma) - (p_{j'} + R_r(\sigma) - R_h(\sigma))$; $W(\sigma') = W(\sigma) - (p_{j'} + R_r(\sigma) - R_h(\sigma))$.

The time complexity of the process of checking for the possibility of applying permutations is determined by the formulae $O(n \log n)$ for permutations 1P-0P-Δ, 1P-0P-RΔ, 1P-0P-R and $O(n^2)$ for permutations 1P-1P-Δ, 1P-1P-RΔ, 1P-1P-R, 1P-1S-W1, 1P-1S-W2, 1P-1S-W3.

Each of the PSC-algorithms (A1 and A2) for the solution to the TTP problem includes the first polynomial component and the approximation algorithm and is built solely on directed permutations. The first polynomial component of the algorithm is a deterministic procedure consisting of the sequential execution of directed permutations. The total number of permutations is limited by a polynomial of the number of jobs and the number of machines. When the problem has been solved, we obtain either a strictly optimal solution from the first polynomial component of the algorithm (if any of the SCOs were satisfied during the computation) or an approximate one with an upper bound on the deviation from the optimum [7, 10].

Statement 10. Suppose that a schedule σ for the TTPR problem obtained using Algorithm A1 (A2) has the following properties: it is not feasible, and the completion time of at least one machine

in the schedule is strictly less than the due date. Then, each machine cannot have more than one tardy job, and this job is necessarily partially tardy.

Proof. According to the implementation logic of Algorithm A1 (A2) [7, 10], we assign each next unassigned job to the machine with a minimum completion time. Hence, if any machine has a reserve, then the job will be assigned by Algorithm A1 (A2) to the machine with a reserve and not as the next tardy job to any other machine. $\square$

2.3. Modification to algorithms A1 and A2 for the case where the release times of some machines exceed the due date (the TTPRE problem)

The modification of algorithms A1 and A2 for the TTPRE problem is described below.

Build an initial schedule $\sigma^{\text{init}} \in \Psi_P$ with the algorithm A^{init} described above. Conditionally split σ^{init} into σ^1 and σ^2 , where σ^1 is the schedule of jobs on machines with $r_i < d$ and σ^2 is the schedule of jobs on machines with $r_i \geq d$ [7, 13].

Statement 11. [7, 13]. The maximal difference in the number of tardy jobs on the machines in the schedule σ^1 does not exceed one.

Statement 12. [7, 13]. The number of tardy jobs on each of the machines with $r_i < d$ is greater than or equal to the number of tardy jobs on each of the machines with $r_i \geq d$.

The truth of Statements 11 and 12 is based on the algorithm A^{init} .

Theorem 11. [7, 13]. Any permutation of jobs on the schedule σ^{init} between machines $i \in \sigma^1$ and $i \in \sigma^2$ cannot lead to a decrease in the functional value.

Corollary 4. [7]. It is necessary to perform the optimisation only for the schedule σ^1 . The schedule σ^2 is optimal by construction, as shown in Corollary 2 to Theorem 1.

Corollary 5. [7]. The SCOs are checked for the schedule σ^1 , and the estimate of the deviation of the functional value from the optimum, $\Omega_{\Sigma}(\sigma) = \min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma))$, is determined for the schedule σ^1 , since the schedule σ^2 is optimal by construction.

Theorem 12. The optimal solution for the TTPRE problem belongs to the class of schedules Ψ_P .

Proof. The schedule σ^2 is optimal by construction. The schedule σ^1 corresponds to the formulation of the TTPR problem, so all the properties of the class Ψ_P are true for σ^1 (Theorems 3–10 and Statements 3–6), including SCOs #1 and #2 (Theorems 4 and 7). Permutations between the schedules σ^1 and σ^2 are not performed (Theorem 11). $\square$

Corollary 6. The SCOs #1 and #2 of the TTPR problem (Theorems 4 and 7) remain true for the TTPRE problem.

The following theorem is also true for the TTPRE problem, and its proof is obvious.

Theorem 13. [7, 13]. The functional value for the TTPRE problem is equal to the sum of the functional values for the schedules σ^1 and σ^2 .

Remark 4. Corollary 6 and Theorem 13 are used in the PSC-algorithm for solving the TTPRE problem [7].

Scheme used to solve the TTPRE problem [7, 13]. Build an initial schedule $\sigma^{\text{init}} \in \Psi_P$ using the algorithm A^{init} described above. Conditionally split σ^{init} into σ^1 and σ^2 , where σ^1 is the schedule of jobs on machines with $r_i < d$, and σ^2 is the schedule of jobs on machines with $r_i \geq d$. Execute one of the PSC-algorithms to solve the TTPR problem (Algorithm A1 or A2) for machines with $r_i < d$. Combine the resulting schedule σ^1 with the schedule σ^2 for machines where $r_i \geq d$.

Remark 5. We have described the previously published theoretical provisions and the schemes of the algorithms for the TTPR and TTPRE problems in detail because this allows us to rigorously substantiate (theoretically prove and effectively use in the modified algorithms below) the following new results:

- Rigorous proof of Statements 1 and 2 previously published in [9] (there was no rigorous proof of these statements in previous publications).
- Formulated for the first time Theorems 2, 3, and 12, which contain new theoretical properties of the TTPR and TTPRE problems. From these theorems in particular, a rigorous justification follows for the SCOs #1 and #2 previously published in [7, 9, 10].
- Formulated for the first time Statement 10 (Subsection 2.2) and its corollaries (Statements 15–17 given in Section 5).

2.4. The efficiency of PSC-algorithms A1 and A2 for non-feasible schedules

PSC-algorithms A1 and A2 for solving these problems [7, 10] were built in accordance with the computational scheme given above. *Algorithm A1* is more time-consuming than *Algorithm A2* and consists of the sequential execution of Stages I–V described above with the use of all types of permutations. Its complexity is limited by the polynomial $O(n^2m)$. *Algorithm A2* includes the permutations most frequently performed during the execution of the first polynomial component of *Algorithm A1*, namely *IP-OP-Δ*, *IP-OP-RΔ*, *IP-OP-R*. This second algorithm involves the sequential execution of Stages I–III considering only these three types of permutations (permutations *IP-OP-Δ* and *IP-OP-R* are performed at Stage II, *IP-OP-RΔ* at Stage III, Stages IV–V are not executed). It also contains the given above algorithm A^{init} , which is simpler than that of algorithm A1, for building an initial schedule. The complexity of *Algorithm A2* is therefore significantly lower than that of *Algorithm A1* and is determined by the polynomial $O(mn \log n)$ [7, 10].

A schedule that does not have any tardy jobs is referred to here as *feasible*.

It was statistically shown in [7, 10] that the PSC-algorithms A1 and A2 for the TTP problem are efficient for building an optimal schedule in the case where SCOs #1 and/or #2 are satisfied by the obtained solution.

Statistical studies of algorithms A1 and A2 were conducted in [7, 10] for a due date factor $DF = 0.7$. This article presents statistical studies for $DF = 0.3 \dots 0.99$. Hence, only non-feasible schedules are considered.

To study the efficiency of the algorithms, a modelling system written in C# was used in a Microsoft Visual Studio 2010 environment. Job sets were randomly generated with a uniform distribution of parameters. The processing times for the jobs were chosen from the interval $[1, 200]$ [7, 10]. The due date was calculated as $DF \times L/m$, where L is the total processing time for all jobs. 1000 tests were conducted for each number of jobs n with a fixed number of machines $m = 10$. The study was done on an Intel Core i5-3210M processor with a clock speed of 2.5 GHz and 8 GB of RAM.

The results for the average frequency of obtaining an optimal solution (depending on the dimension and the due date factor) and the average solution times are given in Table 1 for Algorithm A1 and Table 2 for Algorithm A2.

As we can see from an analysis of Table 1 and Table 2, Algorithm A2 is qualitatively superior to Algorithm A1, both in terms of the frequency of obtaining an optimal solution and in the average solution time.

In the case where $\Delta_{\Sigma}(\sigma) > 0$ and $R_{\Sigma}(\sigma) > 0$ for the obtained solution, A1 and A2, as polynomial-time approximation algorithms, give solutions close to the optimum in terms of the functional value, based on an estimate of the deviation of the obtained solution from the optimum. Algorithms A1 and A2 are not statistically efficient for building an optimal schedule in the case where a feasible solution exists for a given problem instance. The reason for this is that to obtain a feasible solution, the algorithm (A1 or A2) must solve the problem instance exactly, and we know that the TTPR problem is NP-hard. We therefore explicitly introduce the following:

- An obvious **SCO #3** for the TTPR and TTPRE problems: a feasible schedule (one that has no tardy jobs) is optimal.
- An efficient polynomial sub-algorithm (A0) for building a feasible schedule (its efficiency in obtaining a feasible schedule follows from the meaning of the functional in the MDM problem given below and, consequently, of Statement 13).

Table 1

Average frequency (%) of obtaining an optimal solution (depending on the number of jobs n and the due date factor DF) and average solution time t_{av} (ms) for Algorithm A1

Number of jobs, n	Due date factor, DF								Average solution time, t_{av} ($DF = 0.7$)
	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99	
15	31.4	17.4	94.0	100.0	29.6	54.9	0.0	0.0	0.019
20	0.0	5.7	56.2	100.0	100.0	80.6	26.0	0.0	0.014
25	33.0	0.0	78.7	100.0	98.9	85.6	51.2	0.0	0.017
30	96.4	33.4	22.4	87.2	100.0	100.0	89.6	0.0	0.014
35	100.0	84.8	37.3	76.7	100.0	100.0	85.3	0.0	0.011
40	41.6	100.0	78.1	35.6	100.0	100.0	77.8	0.0	0.012
45	39.2	89.5	100.0	22.4	87.4	100.0	96.3	0.0	0.015
50	86.2	71.2	100.0	32.1	66.3	100.0	100.0	0.0	0.019
55	97.8	26.2	100.0	90.1	42.2	100.0	100.0	0.0	0.029
60	78.2	72.7	75.1	100.0	22.3	100.0	100.0	0.0	0.027
65	50.1	100.0	40.0	100.0	24.6	100.0	100.0	4.6	0.019
70	67.0	95.6	56.7	97.3	59.1	100.0	100.0	5.7	0.025
75	97.2	74.8	82.8	97.2	90.6	100.0	100.0	10.7	0.025
100	98.0	80.4	58.6	89.4	97.8	43.9	100.0	15.8	0.015
125	90.6	91.0	96.4	88.4	43.7	98.6	100.0	29.9	0.044
150	89.8	97.2	97.4	100.0	100.0	100.0	100.0	48.5	0.030
200	80.8	99.1	63.8	99.2	95.9	67.7	21.4	69.9	0.041
300	89.2	87.4	90.9	92.6	69.0	83.2	100.0	93.1	0.099
400	84.1	88.0	91.2	84.2	100.0	93.7	45.5	96.5	0.075
500	91.4	91.7	90.0	91.4	91.8	98.0	100.0	99.6	0.126
1000	91.7	91.3	90.4	93.7	93.8	98.7	85.5	100.0	0.254
5000	84.0	85.8	83.8	82.8	84.5	82.1	86.6	100.0	9.293
10 000	82.1	83.3	82.6	82.5	81.9	83.2	90.8	20.3	32.365
50 000	83.2	85.9	84.3	84.1	84.3	83.2	84.4	75.7	672.589
t_{av}	333.08	401.83	548.13	598.32	715.18	871.91	951.87	1796.53	

Table 2

Average frequency (%) of obtaining an optimal solution (depending on the number of jobs n and the due date factor DF) and average solution time t_{av} (ms) for Algorithm A2

Number of jobs, n	Due date factor, DF								Average solution time, t_{av} ($DF = 0.7$)
	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99	
15	73.8	67.7	95.1	100.0	100.0	100.0	100.0	100.0	0.007
20	4.2	33.9	79.2	100.0	100.0	100.0	100.0	100.0	0.010
25	30.0	14.1	64.7	97.5	100.0	100.0	100.0	100.0	0.016
30	100.0	20.0	36.1	89.2	100.0	100.0	100.0	100.0	0.016
35	100.0	93.9	3.8	27.0	100.0	100.0	100.0	100.0	0.011
40	51.1	97.4	73.9	4.0	100.0	100.0	100.0	100.0	0.011
45	29.6	86.7	100.0	10.2	95.7	100.0	100.0	100.0	0.015
50	88.7	71.2	100.0	61.1	82.4	100.0	100.0	100.0	0.017
55	100.0	42.9	94.2	97.1	37.6	100.0	100.0	100.0	0.020
60	83.7	81.1	76.9	100.0	15.2	100.0	100.0	100.0	0.026
65	43.7	100.0	46.4	100.0	43.4	100.0	100.0	100.0	0.025
70	87.9	96.6	38.7	100.0	81.4	100.0	100.0	100.0	0.032
75	100.0	76.6	87.1	92.1	93.5	98.4	100.0	100.0	0.023
100	96.1	88.8	58.6	98.8	100.0	59.2	100.0	100.0	0.029
125	95.6	94.9	96.7	73.5	68.3	100.0	100.0	100.0	0.040
150	89.9	96.8	98.2	96.9	100.0	100.0	100.0	100.0	0.040
200	85.9	94.2	90.1	97.2	98.7	84.5	57.4	100.0	0.066
300	90.5	90.1	94.8	96.1	89.6	95.1	100.0	100.0	0.068
400	98.5	92.3	97.1	93.4	100.0	98.4	87.6	100.0	0.080
500	95.4	94.4	99.1	98.5	98.2	98.7	100.0	100.0	0.107
1000	97.8	99.1	98.5	99.3	99.6	99.4	100.0	100.0	0.210
5000	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	0.955
10 000	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	2.101
50 000	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	12.065
t_{av}	16.193	15.677	15.739	15.949	15.990	15.974	15.810	15.627	

3. ALGORITHM A0 FOR BUILDING A SCHEDULE THAT MINIMISES THE DIFFERENCE BETWEEN THE MAXIMAL AND THE MINIMAL COMPLETION TIMES OF THE MACHINES

The statement of the auxiliary problem of minimising the difference between the maximal and the minimal completion times of the machines (MDMM). Given a set of jobs $J = \{1, 2, \dots, n\}$ and a set of identical parallel machines $I = \{1, 2, \dots, m\}$, we know the processing time p_j for each job $j \in J$. A machine i , $i \in I$, can start to process any job after its release time $r_i \geq 0$. Machine release times may be not the same, and machine idling is forbidden. We need to build such a schedule σ of the jobs $j \in J$ on m machines that minimises the functional

$$F(\sigma) = C_{\max}(\sigma) - C_{\min}(\sigma) \rightarrow \min, \quad (1)$$

where $C_i(\sigma)$ is the completion time for all jobs on machine i in the schedule σ ; $C_{\max}(\sigma) = \max_{i \in I} C_i(\sigma)$; and $C_{\min}(\sigma) = \min_{i \in I} C_i(\sigma)$.

Remark 6. SCO #3 must be satisfied in the TTPRE problem, so the MDMM problem is formulated only for machines with $r_i < d$.

After a careful search, we believe that the MDMM problem is formulated and solved by a PSC-algorithm for the first time in this study. The criterion of workload balancing (1) for identical parallel machines has been considered only in the case of equal release times (e.g. [19]). This criterion, if used as a separate problem, is important in the context of delivering finished products to customers, in particular, when the recipient expects their joint delivery (just-in-time delivery planning). The smaller the difference between product delivery times, the more efficiently supply chain planning can be done and warehousing costs, as well as penalties associated with a loss of orders and friendliness of customers, can be minimized.

A justification of Algorithm A0 as an efficient polynomial-time algorithm for building a feasible schedule (SCO #3) if it exists is presented below.

A schedule σ is referred to as *aligned* if $C_i(\sigma) = \text{const } \forall i \in I$. The justification for Algorithm A0 follows from the following statement.

Statement 13. If there is a feasible solution for a given instance of the TTPR problem (an optimal schedule, since the completion times of all jobs in it do not exceed the common due date), then the aligned schedule, if it exists, is feasible and hence optimal for this TTPR problem.

Proof. Consider an arbitrary feasible schedule σ with $C_{\max}(\sigma) \neq C_{\min}(\sigma)$, $C_{\max}(\sigma) \leq d$. A schedule σ' for which $C_i(\sigma') = \text{const } \forall i \in I$ therefore satisfies the following: $C_i(\sigma) < C_{\max}(\sigma) \leq d$. $\square$

Statement 14. Suppose that all the parameters of the MDMM problem are positive digital numbers, and the following statement is true for an arbitrary schedule σ : $C_{\max}(\sigma) - C_{\min}(\sigma)$ is equal to one, or (in the case of equal machine release times) to the greatest common divisor of the total processing time of all jobs (GCDTPT). Then, the schedule σ is optimal based on criterion (1), the value of $C_{\max}(\sigma)$ is the minimum possible, and the resulting schedule is therefore feasible (if a feasible schedule exists).

Proof. Proof of the first part of Statement 14 follows from the fact that an aligned schedule σ' ($C_i(\sigma') = \text{const } \forall i \in I$) does not exist, as otherwise, the inequality $C_{\min}(\sigma) \leq C_i(\sigma') \leq C_{\max}(\sigma)$ would hold, which is impossible because $C_i(\sigma')$ is a positive integer number. The minimal possible value of $C_{\max}(\sigma) - C_{\min}(\sigma)$ is therefore equal to one or the GCDTPT (in the case of equal machine release times).

The second part of Statement 14 is obvious. $\square$

Suppose that no schedule satisfies Statement 13 or 14. The following two facts are true:

- There is a strong relationship between the functional in (1) and $\min_{\sigma} C_{\max}(\sigma)$ concerning the value of $C_{\max}(\sigma)$ (one of the main strategies for obtaining a schedule regarding the functional in (1) is increasing the current minimal completion time of the machines by reducing the current maximal completion time of the machines).
- If there is a feasible schedule for the TTPR problem, then the schedule that minimises $C_{\max}(\sigma)$ is also feasible (which is obvious).

We will therefore use Algorithm A0, which minimises the functional in (1), as a polynomial-time algorithm that checks SCO #3. In this case, an additional effect is achieved: the resulting schedule is aligned as far as possible, which increases the efficiency of further exploitation of the machines.

It follows from Statements 13 and 14 that there are two SCOs for the solution obtained for the MDMM problem.

SCO #1 for the MDMM problem: if $C_i(\sigma) = \text{const } \forall i \in I$ (the schedule σ is aligned), then σ is optimal.

SCO #2 for the MDMM problem (in the case where the parameters of the problem are positive digital numbers): if $C_{\max}(\sigma) - C_{\min}(\sigma)$ is equal to one or the GCDTPT (in the case of equal machine release times), then the schedule σ is optimal based on criterion (1).

This allows us to classify Algorithm A0 as a PSC-algorithm of the second class for solving the MDMM problem.

We also introduce the following notation to describe Algorithm A0:

$$C_{av} = \frac{\sum_{j=1}^n p_j + \sum_{i=1}^m r_i}{m},$$

$\Delta_i(\sigma) = \max\{0, C_i(\sigma) - C_{av}\}$ is the tardiness of a machine $i \in I$, $R_i(\sigma) = \max\{0, C_{av} - C_i(\sigma)\}$ is the reserve of a machine $i \in I$, $I_{\Delta}(\sigma)$ is the set of machine numbers i : $\Delta_i(\sigma) > 0$ in a schedule σ ,

$I_R(\sigma)$ is the set of machine numbers i : $R_i(\sigma) > 0$ in a schedule σ .

Algorithm A0 consists of two stages:

1. An algorithm for building the initial schedule.
2. An approximation algorithm for building a schedule that minimises the specified criterion.

The algorithm for building the initial schedule σ_0 is as follows:

1. Renumber the jobs of the set J in non-increasing order of processing times p_j .
2. Renumber the machines in non-decreasing order of release times r_i .
3. Initialise the completion times of machines: $C_i = r_i \forall i = \overline{1, m}$.
4. Select an unassigned job j with the maximal processing time p_j . Assign the job j to a machine i that has a minimal completion time C_i .
5. Calculate the new completion time of the machine i : $C_i = C_i + p_j$.
6. If all jobs have been assigned, the algorithm terminates; otherwise, go to Step 4.

We denote the resulting schedule as σ_0 .

If any of the SCOs for the MDMM problem is satisfied for the schedule σ_0 , then the obtained schedule is optimal based on the MDMM criterion, and the algorithm terminates. Otherwise, we perform optimisation (the second stage of Algorithm A0).

The second stage consists of the successive execution of a series of permutations that improve the schedule and minimise the functional value.

To improve the schedule, we need to focus on reducing the maximal tardiness $\max_{i \in I_{\Delta}(\sigma)} \Delta_i(\sigma)$. To do this, we use permutations of jobs between machine h with the maximal value of $\max_{i \in I_{\Delta}(\sigma)} \Delta_i(\sigma)$ and the next machine s from the set $I_R(\sigma)$, starting from the machine with the maximal value of $\max_{i \in I_R(\sigma)} R_i(\sigma)$.

A subset of jobs from machine h (denoted as $K_h(\sigma)$, $K_h(\sigma) \subseteq J_h(\sigma)$ by definition) is swapped with a subset of jobs from machine s (this subset is denoted as $L_s(\sigma)$, $L_s(\sigma) \subseteq J_s(\sigma)$) [7, 20]. Let $\theta = \sum_{j \in K_h(\sigma)} p_j - \sum_{j \in L_s(\sigma)} p_j$ represent the difference between the total processing times for jobs taking part in the permutation [7, 20].

We use the following types of permutations: permutations of type A (Table 3) and type B (Table 4).

Table 3
Properties of permutations of type A [7, 20]

Permutation type	Machines and jobs taking part in the permutation		Difference between the total processing times, θ ($\theta > 0$)	Condition applied to perform permutation	Characteristics of the resulting schedule σ^1
	From (h)	To (s)			
1-1A	$h \in I_\Delta(\sigma), j_1 \in J_h$	$s \in I_R(\sigma), j_2 \in J_s$	$p_{j_1} - p_{j_2}$	$\theta \leq \Delta_h(\sigma),$ $\theta \leq R_s(\sigma)$	$\Delta_h(\sigma^1) = \Delta_h(\sigma) - \theta,$ $R_s(\sigma^1) = R_s(\sigma) - \theta$
1-2A	$h \in I_\Delta(\sigma), j_1 \in J_h$	$s \in I_R(\sigma), j_2, j_3 \in J_s$	$p_{j_1} - (p_{j_2} + p_{j_3})$		
2-1A	$h \in I_\Delta(\sigma), j_1, j_2 \in J_h$	$s \in I_R(\sigma), j_3 \in J_s$	$(p_{j_1} + p_{j_2}) - p_{j_3}$		
2-2A	$h \in I_\Delta(\sigma), j_1, j_2 \in J_h$	$s \in I_R(\sigma), j_3, j_4 \in J_s$	$(p_{j_1} + p_{j_2}) - (p_{j_3} + p_{j_4})$		

Table 4
Properties of permutations of type B [7, 20]

Permutation type	Machines and jobs taking part in the permutation		Difference between the total processing times, θ ($\theta > 0$)	Condition applied to perform permutation	Characteristics of the resulting schedule σ^1
	From (h)	To (s)			
1-1B	$h \in I_\Delta(\sigma), j_1 \in J_h$	$s \in I_R(\sigma), j_2 \in J_s$	$p_{j_1} - p_{j_2}$	$\theta \leq \Delta_h(\sigma),$ $\theta > R_s(\sigma)$	$\Delta_h(\sigma^1) = \Delta_h(\sigma) - \theta,$ $R_s(\sigma^1) = \theta - R_s(\sigma)$
1-2B	$h \in I_\Delta(\sigma), j_1 \in J_h$	$s \in I_R(\sigma), j_2, j_3 \in J_s$	$p_{j_1} - (p_{j_2} + p_{j_3})$		
2-1B	$h \in I_\Delta(\sigma), j_1, j_2 \in J_h$	$s \in I_R(\sigma), j_3 \in J_s$	$(p_{j_1} + p_{j_2}) - p_{j_3}$		
2-2B	$h \in I_\Delta(\sigma), j_1, j_2 \in J_h$	$s \in I_R(\sigma), j_3, j_4 \in J_s$	$(p_{j_1} + p_{j_2}) - (p_{j_3} + p_{j_4})$		

The algorithm used in the second stage is a simplification of the PSC-algorithm from [7, 20] in which only permutations of type A or type B are used. It can be summarised as follows:

1. Build the initial schedule σ_0 , $\sigma = \sigma_0$.
2. Determine the sets $I_\Delta(\sigma)$ and $I_R(\sigma)$.
3. Check SCOs #1 and #2 for the MDMM problem. If at least one is satisfied, then go to Step 6 (σ is the optimal schedule); otherwise, go to Step 4.
4. Determine the machine h that corresponds to the maximal value of the tardiness: $\Delta_h(\sigma) = \max_{i \in I_\Delta(\sigma)} \Delta_i(\sigma)$ (if there are several machines with the same maximum value of $\Delta_i(\sigma)$, then consider each of them).
5. For machine h , looking over machines $s \in I_R(\sigma)$, perform a permutation of type A or type B. If there are no such permutations, then go to Step 6; otherwise, go to Step 2.
6. The algorithm terminates: we have obtained schedule σ .

4. EXAMPLES OF THE MDMM PROBLEM AND COMPUTATIONAL STUDIES OF ALGORITHM A0

Example 1. Given: $m = 3$, $n = 4$, the values of r_i and p_j are given in Table 5.

Table 5
Initial data for Example 1

j	1	2	3	4
p_j	9	8	7	3

i	1	2	3
r_i	0	2	4

The initial schedule σ_0 is shown in Fig. 1a. We have $C_{\max}(\sigma_0) = 12$, $I_\Delta(\sigma_0) = \{1\}$, $C_{\min}(\sigma_0) = 10$, $I_R(\sigma_0) = \{2\}$.

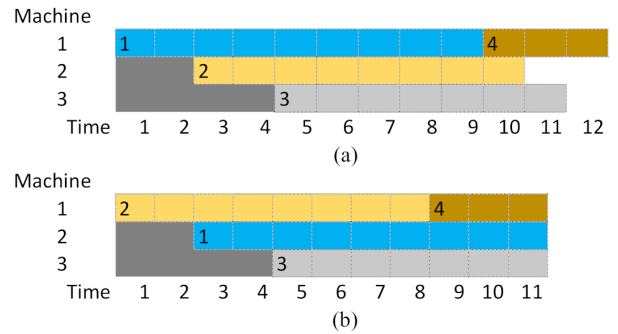


Fig. 1. Illustration of the solution to Example 1

SCOs #1 and #2 for the MDMM problem are not satisfied, and we therefore execute the second stage of Algorithm A0. There is a permutation of type 1-1A in this example, namely a permutation of jobs 1 and 2 ($\theta = \Delta_h(\sigma_0) = R_s(\sigma_0) = 1$). By swapping jobs 1 and 2, we obtain the aligned schedule σ shown in Fig. 1b. SCO #1 for the MDMM problem is satisfied, and the schedule σ is optimal.

Example 2. Given: $m = 3$, $n = 9$, the values of r_i and p_j are given in Table 6.

Table 6
Initial data for Example 2

j	1	2	3	4	5	6	7	8	9
p_j	11	8	8	6	6	5	5	5	3

i	1	2	3
r_i	0	1	2

The initial schedule σ_0 is shown in Fig. 2a. We have $C_{\max}(\sigma_0) = 21$, $C_{\min}(\sigma_0) = 19$, $I_\Delta(\sigma_0) = \{1\}$, and $I_R(\sigma_0) = \{3\}$.

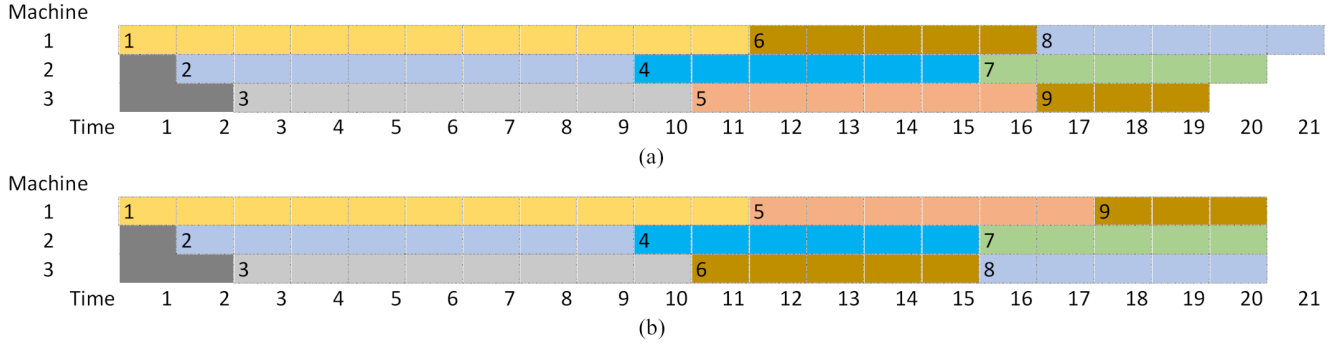


Fig. 2. Illustration of the solution to Example 2

SCOs #1 and #2 for the MDMM problem are not satisfied, and we therefore execute the second stage of Algorithm A0. There is a permutation of type 2-2A in this example, namely a permutation of the group of jobs 5 and 9 from machine 3 and the group of jobs 6 and 8 from machine 1 ($\theta = \Delta_h(\sigma_0) = R_s(\sigma_0) = 1$). After this permutation, we obtain the aligned schedule σ shown in Fig. 2b. SCO #1 for the MDMM problem is satisfied, and the schedule σ is optimal.

Example 3. Given: $m = 4$, $n = 20$, the values of r_i and p_j are given in Table 7.

Table 7
Initial data for Example 3

j	1	2	3	4	5	6	7	8	9	10
p_j	300	270	230	220	200	150	120	90	70	50
j	11	12	13	14	15	16	17	18	19	20
p_j	41	40	35	30	28	24	17	12	7	3

i	1	2	3	4
r_i	0	2	3	5

In the initial schedule σ_0 (Table 8a), we have $C_{\max}(\sigma_0) = 489$, $C_{\min}(\sigma_0) = 485$, $C_{\text{av}}(\sigma_0) = 486.75$, $I_{\Delta}(\sigma_0) = \{3, 4\}$, and $I_R(\sigma_0) = \{1, 2\}$. SCOs #1 and #2 for the MDMM problem are not satisfied, and we execute the second stage of Algorithm A0. There is a permutation of type 2-2B in this example, namely a permutation of the group of jobs 10 and 18 from machine 1 and the group of jobs 12 and 16 from machine 4 ($\theta = 2 < \Delta_h(\sigma_0) = 2.25$, $\theta > R_s(\sigma_0) = 1.75$). After the permutation, we obtain the schedule σ shown in Table 8b. We have $C_{\max}(\sigma) = 487$ and $C_{\min}(\sigma) = 486$. A schedule that is close to aligned is built with a value of the functional equal to one. SCO #2 for the MDMM problem is satisfied, and the schedule σ is optimal.

As described above, we performed computational studies of the efficiency of Algorithm A0 on a series of test instances satisfying SCO #1 or #2 for the MDMM problem. These studies showed that the percentage of schedules obtained by Algorithm A0 that satisfied SCO #1 or #2 for the MDMM problem was in the range of 85–95% of instances for various dimensions and depended on the tightness of the job processing times: the greater the difference between the maximal and the minimal processing times, the lower the percentage of obtaining an optimal schedule using Algorithm A0 according to criterion (1).

Table 8
Solution to Example 3

i	r_i	j	p_j	c_j
1	0	1	300	300
		8	90	390
		10	50	440
		14	30	470
		18	12	482
		20	3	485
2	2	2	270	272
		7	121	393
		11	41	434
		13	35	469
		17	17	486
3	3	3	230	233
		6	150	383
		9	69	452
		15	28	480
		19	7	487
4	5	4	220	225
		5	200	425
		12	40	465
		16	24	489

i	r_i	j	p_j	c_j
1	0	1	300	300
		8	90	390
		12	40	430
		14	30	460
		16	24	484
		20	3	487
2	2	2	270	272
		7	121	393
		11	41	434
		13	35	469
		17	17	486
3	3	3	230	233
		6	150	383
		9	69	452
		15	28	480
		19	7	487
4	5	4	220	225
		5	200	425
		10	50	475
		18	12	487

5. MODIFIED PSC-ALGORITHMS FOR SOLVING THE TTPR AND TTPRE PROBLEMS

The structure of the modified PSC-algorithms for the TTPR and TTPRE problems is based on the results presented in the previous sections and the following new statements.

Statement 15. SCOs #1–3 cannot be satisfied by a schedule obtained with an algorithm (A1 or A2) for the TTPR (or TTPRE) problem if at least one machine in this schedule has $C_i < d$ and at least one machine has $C_i > d$ (this machine has only one tardy job, according to Statement 10).

Proof. In this case, Algorithm A1 (A2) yields a non-feasible schedule (SCO #3 cannot be satisfied). The solution also cannot satisfy SCO #1, since at least one machine in the obtained schedule has $C_i < d$ and no more than one job is tardy on machines with $C_i > d$. This condition contradicts the definition of an even schedule (where we have the same number of tardy jobs on all machines). SCO #2 cannot also be satisfied by this schedule, since $R_\Sigma(\sigma) > 0$ in this case, as by its general definition $R_\Sigma(\sigma)$ is equal to the total reserve of the machines with $C_i < d$, and the analysed schedule σ has at least one machine with a non-zero reserve. In addition, we have $\Delta_\Sigma(\sigma) > 0$ for the analysed schedule, for the following reasons:

- By definition, $\Delta_\Sigma(\sigma)$ is equal to the total tardiness of partially tardy jobs on machines with the maximal number of tardy jobs.
- The analysed schedule has at least one machine with $C_i > d$.
- No machine has a tardy job starting at d because such a job would be moved by Algorithm A1 (A2) to a machine with reserve, meaning that the total tardiness would decrease (Statement 10).

Thus, $\Delta_\Sigma(\sigma)$ for a considered schedule σ is equal to the total tardiness of all partially tardy jobs and is greater than zero. In our case, we determine an estimate of the maximal possible deviation of the solution from the optimum, which is equal to $\min(R_\Sigma(\sigma), \Delta_\Sigma(\sigma))$ (see Theorem 9). $\square$

Statement 16. If $C_i \geq d \forall i = \overline{1, m}$ on a schedule σ obtained by Algorithm A1 (A2), and at least one machine has $C_i = d$, then $R_\Sigma(\sigma) = 0$ on this schedule, and it is optimal.

Proof. Proof of this statement follows from the general definition of $R_\Sigma(\sigma)$ (the value of $R_\Sigma(\sigma)$ is calculated on machines with a smaller number of tardy jobs – in this case, machines without tardy jobs). $\square$

Statement 17. If $C_i > d \forall i = \overline{1, m}$ for a schedule σ obtained by Algorithm A1 (A2), then SCOs #1 and #2 can be satisfied

by such a schedule (which may be even or satisfy the condition $R_\Sigma(\sigma) = 0$ and/or $\Delta_\Sigma(\sigma) = 0$).

Proof. Proof of this statement follows from Theorem 1, or the examples given below. $\square$

Figure 3 shows examples of schedules obtained by Algorithm A1 that correspond to the conditions of Statement 17. The schedule in Fig. 3a is even, as SCO #1 is satisfied (each machine has two tardy jobs). The condition $R_\Sigma(\sigma) = 0$ is satisfied by the schedule in Fig. 3b because $R_i(\sigma) = d - \sum_{j \in P_i(\sigma)} p_j = 0$, $i \in \{2, 3, 4\}$ ($R_i(\sigma)$ is calculated for machines 2, 3, and 4, which have fewer tardy jobs and no reserve). The condition $\Delta_\Sigma(\sigma) = 0$ is satisfied by the schedule in Fig. 3c, as $S_i(\sigma) = \emptyset \forall i = \overline{1, 4}$ ($\Delta_\Sigma(\sigma)$ is calculated for all machines in this case, and no machine has a partially tardy job).

The condition $R_\Sigma(\sigma) = 0$ is also satisfied in the example shown in Fig. 3c. Thus, it follows from this example that both conditions $\Delta_\Sigma(\sigma) = 0$ and $R_\Sigma(\sigma) = 0$ which make up SCO #2 may be satisfied for the same problem instance.

Remark 7. We determine $R_\Sigma(\sigma)$ and $\Delta_\Sigma(\sigma)$ and check SCO #2 only for a non-even schedule since an even schedule is optimal. Accordingly, we should first check using Algorithm A1 (A2) whether SCO #1 holds, and only if it is false do we check SCO #2. The estimate of the maximal possible deviation of the total tardiness from the optimum, which is equal to $\min(R_\Sigma(\sigma), \Delta_\Sigma(\sigma))$ (Theorem 9), also makes sense only for a non-even schedule.

The modified PSC-algorithms are implemented as follows. Suppose that the following inequality holds for the problem instance to be solved:

$$\sum_{j=1}^n p_j \leq \sum_{i=1}^m (d - r_i). \quad (2)$$

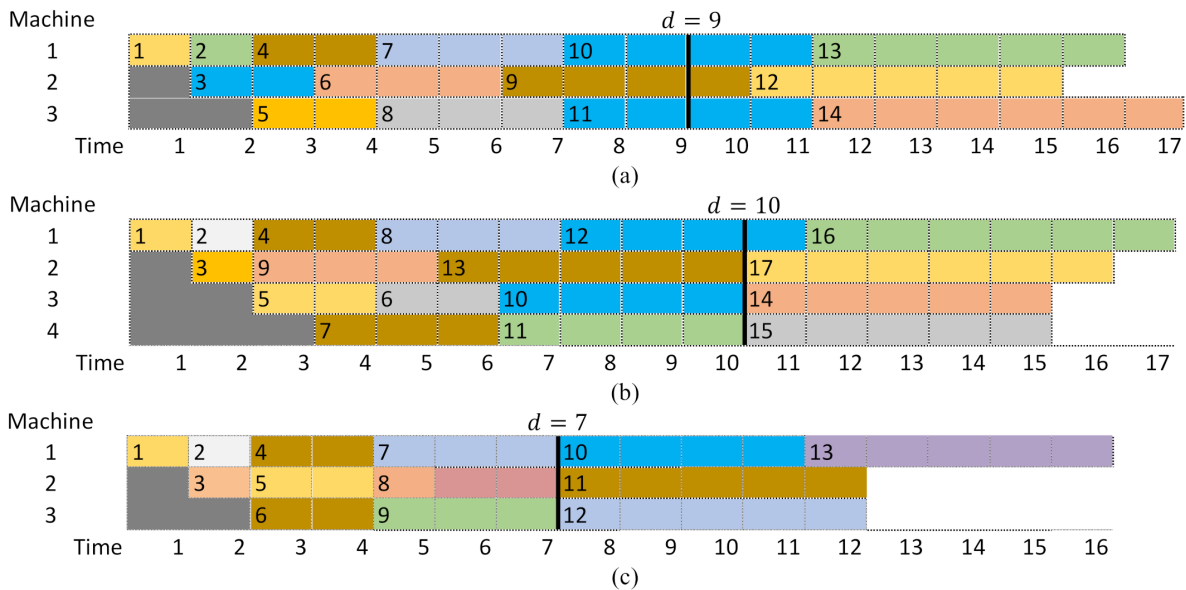


Fig. 3. Illustration of Statement 17

In this case, only SCO #3 can be satisfied. This follows from inequality (2) and Statements 10 and 15. Indeed, a feasible schedule (SCO #3) satisfies inequality (2). If a schedule that satisfies inequality (2) is not feasible, then there must exist a machine with a non-zero reserve, and no more than one job can be tardy on machines with $C_i > d$ (Statement 10). So, from Statement 15 we know that SCOs #1 and #2 cannot be satisfied for the problem instance under consideration.

Statistical studies of Algorithms A1 and A2 showed that if the optimal schedule belongs to the classes of schedules corresponding to Statements 16 and 17, then Algorithm A2 obtains it statistically significantly more often than Algorithm A1. If the schedule obtained by Algorithm A2 is partially tardy with a non-zero reserve (Statement 15), then Algorithm A1 obtains an approximate solution that is statistically significantly better than the schedule obtained by Algorithm A2.

Thus, if Algorithm A0 (for the case $\forall r_i < d$ or only for machines with $r_i < d$) does not obtain a feasible schedule, then we recommend first solving the problem using Algorithm A2. Then, in the case of obtaining a partially tardy schedule with non-zero reserve, solve the problem again by Algorithm A1 (if time buffer and computing capacity allow it) and on the best of the two schedules (obtained by Algorithms A1 and A2) find the estimate of the deviation of the functional from the optimum (Theorem 9).

Suppose that inequality (2) is false. We then solve the problem using Algorithm A1. If the resulting schedule satisfies the conditions of Statement 16, then it is optimal. If the resulting schedule σ has $C_i > d \forall i = \overline{1, m}$, then we check whether SCO #1 or the condition $R_\Sigma(\sigma) = 0$, or the condition $\Delta_\Sigma(\sigma) = 0$ is satisfied. If any of these three conditions are met, the resulting schedule is optimal; otherwise, we have obtained an approximate solution, and by virtue of Theorem 9, the estimate of the maximal possible deviation of the total tardiness from the optimum does not exceed the value of $\min(R_\Sigma(\sigma), \Delta_\Sigma(\sigma))$ calculated for the schedule σ obtained using Algorithm A1.

Remark 8. The modified PSC-algorithms for the TTPR and TTPRE problems can be used instead of previous versions of these algorithms in the Four-Level Model of Planning and Decision Making [21] for discrete manufacturing systems with a network representation of technological processes.

6. THE INTRACTABLE PROBLEM OF MINIMISING THE TOTAL TARDINESS OF PARALLEL MACHINE COMPLETION TIMES REGARDING THE COMMON DUE DATE WITH MACHINE RELEASE TIMES

In this section, we show that Algorithm A0 is an efficient PSC-algorithm for solving the problem of minimising the total tardiness of parallel machine completion times regarding the common due date with machine release times (TTCR).

Formal problem statement. Given a set of jobs $J = \{1, 2, \dots, n\}$, and a set of identical parallel machines $I = \{1, 2, \dots, m\}$, we know the processing time p_j for each job $j \in J$. A machine $i, i \in I$, can start to process any job after its release time $r_i \geq 0$. Machine idle times are forbidden. We need

to build a schedule σ for the jobs $j \in J$ on m machines that minimises the total tardiness of all machine completion times regarding the common due date d :

$$F(\sigma) = \sum_{i=1}^m \max[0, C_i(\sigma) - d] \rightarrow \min,$$

where $C_i(\sigma)$ is the completion time for all jobs on the machine i in schedule σ .

This problem has been considered in the literature only for a single machine or for parallel machines with the same release times, as shown by the latest review [22]. It was stated in [23] that this problem is NP-hard for the case of equal machine release times.

The TTCR problem is of practical importance for minimising the tardiness of jobs or product deliveries. Tardiness minimisation is considered the most important criterion in production planning because it facilitates the reduction of deadline violations, tightening of the production plan, reduction of unnecessary production costs, and an increase in customer satisfaction.

The following two SCOs apply to the problem formulated above:

SCO #1 for the TTCR problem. A feasible schedule is optimal.

Proof of the SCO #1 is obvious (the total tardiness is zero).

SCO #2 for the TTCR problem. An arbitrary schedule in which there are no reserves on all machines (release times are greater than or equal to the common due date) is optimal.

Proof. All machines can have no reserves only if the following inequality is true:

$$\sum_{j=1}^n p_j \geq \sum_{i=1}^m (d - r_i).$$

The minimal functional value in this case is $\sum_{j=1}^n p_j - \sum_{i=1}^m (d - r_i)$,

and this is exactly equal to the value of the functional for any schedule in which there are no reserves for all machines. $\square$

SCO #3 for the TTCR problem. If a schedule obtained by Algorithm A0 satisfies SCO #1 or #2 (when the parameters are positive integers) for the MDM problem, then it is also optimal for the TTCR problem.

Proof. 1. If a schedule obtained using Algorithm A0 satisfies SCO #1 for the MDM problem, then it necessarily satisfies SCO #1 or #2 for the TTCR problem.

2. Suppose that the SCO #2 of the MDM problem is satisfied: for a schedule σ obtained using Algorithm A0, the value of the functional in (1) is equal to one or the GCDTPT (if the machine release times are the same). This schedule may not satisfy SCO #1 or #2 for the TTCR problem but is the optimal solution. The proof clearly follows from the following:

- Consider an arbitrary fixed schedule σ^{opt} that is optimal according to criterion (1). Then the number of machines

with $C_i(\sigma) = C_{\min}(\sigma^{\text{opt}}) + \text{GCDTPT}$ (or one) is the same for any schedule σ that is optimal based on criterion (1).

Remark 9. If the machine release times are not the same, then one is added; if they are not, then GCDTPT is added (see Statement 14).

- $C_{\min}(\sigma^{\text{opt}}) = \text{const}$ for any schedule σ^{opt} is optimal based on criterion (1) and is the maximum possible for an arbitrary schedule σ .
- $C_{\min}(\sigma^{\text{opt}}) + \text{GCDTPT}$ (or 1) $\leq C_{\max}(\sigma)$; see Remark 9.
- An aligned schedule does not exist. $\square$

The following statement is true.

Statement 18. Algorithm A0 is a PSC-algorithm that is both an efficient approximation algorithm for solving the TTPCR problem and an efficient polynomial component for building a schedule that satisfies any of the SCOs #1–3 for the TTPCR problem.

Proof. Proof of this statement obviously follows from the fact that the optimality criterion of the MDMM problem minimises the difference $C_{\max}(\sigma) - C_{\min}(\sigma)$ by virtue of the maximal increase in $C_{\min}(\sigma)$ due to the maximal decrease in $C_{\max}(\sigma)$. It follows from this that an optimal solution based on criterion (1) may be:

- A feasible schedule if it exists.
- A schedule without reserves on all machines if it exists.
- A schedule that satisfies SCO #1 or #2 for the MDMM problem if it exists, because these SCOs are satisfied only when the value of criterion (1) is the minimal possible.

If SCOs #1–3 for the TTPCR problem are not satisfied, then the minimisation of criterion (1) has a strong interrelation with the TTPCR criterion. As shown by the above computational studies, Algorithm A0 efficiently solves the problem based on criterion (1). $\square$

Remark 10. Suppose that the solution of the TTPCR problem from Algorithm A0 turns out to be non-optimal (i.e. the solution is not feasible, at least one machine has a completion time less than the due date, and there cannot be more than one tardy job on each machine (Statement 10)). We then recommend solving this problem again with Algorithm A1, and in the case where a feasible schedule is not obtained, a schedule should be chosen from those obtained by Algorithms A0 and A1 that has lower total tardiness for the machines as an approximate solution. By virtue of Theorem 9, the functional value of the solution obtained by Algorithm A1 differs from the optimum by no more than $\min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma))$. If the functional value on the schedule obtained by Algorithm A0 is less than that of the schedule obtained by Algorithm A1, then the estimate $\min(R_{\Sigma}(\sigma), \Delta_{\Sigma}(\sigma))$ decreases in an obvious way.

The TTPCR problem can be illustrated by the examples given in Section 4 for the MDMM problem if a specific due date is set for each example. If we set the due date $d = 11$ in Example 1, then the functional value for the initial schedule is equal to one, and that of the schedule σ is zero, meaning that the schedule σ is optimal according to SCO #1 for the TTPCR problem. A similar result is obtained for Example 2 with a due date $d = 20$. In Example 3,

for $d = 487$, we have $F(\sigma_0) = 2$, $F(\sigma) = 0$, and the schedule σ is optimal according to SCO #1 for the TTPCR problem. If $d = 485$, we have $F(\sigma_0) = F(\sigma) = 7$, and the schedule σ is optimal according to SCO #2 for the TTPCR problem. If $d = 486$, we have $F(\sigma_0) = 4$, $F(\sigma) = 3$, and the schedule σ is optimal according to SCOs #2 and #3 for the TTPCR problem.

7. CONCLUSIONS

In this article, the solution to three problems was considered. To solve the TTPR problem by finding new theoretical properties and modifying the PSC-algorithm for its solution, two original PSC-algorithms were developed for two new problems, MDMM and TTPCR. All three of these problems are interconnected, with common theoretical properties and some common parts of the algorithms.

All the previously published sources were cited and fragments from them were presented to enable the reader to understand the rest of the article. Some of the previously published materials were improved, and rigorous proofs of some statements, which were previously absent, were given. Some of the previously published materials were not previously available in English.

The main results of this article are as follows:

1. New properties of the TTPR and TTPRE problems were proved (Theorems 2, 3, 12), which made it possible to rigorously substantiate SCOs #1 and #2 for the previously published PSC-algorithms A1 and A2 for solving these problems.
2. The PSC-algorithms for solving the TTPR and TTPRE problems were modified by introducing a new SCO for these problems (a schedule without tardy jobs) and an efficient polynomial-time algorithm A0 that can satisfy this SCO by finding a schedule that minimises the difference between the maximal and the minimal machine completion times (MDMM). Two SCOs for the solutions of the MDMM problem obtained by Algorithm A0 were found, as well.

Examples of the solution to the MDMM problem were given using Algorithm A0 and computational studies of the TTPR problem. The results showed that Algorithm A0 builds a feasible solution (if it exists) in 85–95% of cases. This is more often than is achieved by Algorithms A1 and A2 (depending on the dimension). Thus, it was shown that Algorithm A0 is efficient.

3. All the schedules obtained using Algorithms A0 or A1 (A2) for the TTPR and TTPRE problems were divided into four classes, as follows:

- (1) A feasible schedule (it is optimal).
- (2) A partially tardy schedule in which at least one machine has a completion time less than the due date (we show that none of the introduced SCOs is satisfied for this schedule, but it has a rigorously substantiated efficient upper bound for the deviation from the optimum (Theorem 9)).
- (3) A schedule in which all machines have no reserves and at least one machine has a completion time equal to the due date (we show that this schedule is optimal).
- (4) A schedule in which the completion times of all machines are greater than the due date (we show that it may satisfy both the first and the second SCO).

4. Statistical studies of Algorithms A1 and A2 showed that if the optimal schedule belongs to the class (3) or (4), then Algorithm A2 obtains it statistically significantly more often than Algorithm A1. If the schedule obtained by Algorithm A2 belongs to the class (2), then Algorithm A1 obtains an approximate solution that is statistically significantly better than the schedule obtained by Algorithm A2. Thus, if Algorithm A0 (for the case $\forall r_i < d$ or only for machines with $r_i < d$) does not obtain a feasible schedule, then it is recommended that the problem be first solved using Algorithm A2. Then, in the case of obtaining a partially tardy schedule with non-zero reserve, solve the problem again by Algorithm A1 (if time buffer and computing capacity allow it) and on the best of the two schedules (obtained by Algorithms A1 and A2) find the estimate of the deviation of the functional from the optimum (Theorem 9).

5. It was shown that Algorithm A0 is also an efficient PSC-algorithm for solving the single-stage scheduling problem that involves minimising the total tardiness of parallel machine completion times regarding the common due date with machine release times (TTCR). Three SCOs for Algorithm A0 for this problem were proved.

In the case where a partially tardy schedule with non-zero reserve is obtained using Algorithm A0, in accordance with Remark 10, to get the upper bound on the deviation from the optimal functional value, it is necessary to solve the problem again with Algorithm A1. If a feasible schedule is not obtained, the best of the obtained schedules is chosen as an approximate solution. By virtue of Theorem 9, the functional value for the solution obtained by Algorithm A1 differs from the optimum by no more than $\min(R_\Sigma(\sigma), \Delta_\Sigma(\sigma))$. If the functional value on the schedule obtained by Algorithm A0 is less than that of the schedule obtained by Algorithm A1, then the estimate $\min(R_\Sigma(\sigma), \Delta_\Sigma(\sigma))$ decreases in an obvious way. Thus, a new PSC-algorithm for solving the TTCR problem, based on the use of Algorithms A0 and A1, is proposed.

6. The obtained results allow us to formulate their application areas and future scientific research directions:

- Considered single-stage scheduling problems and proposed PSC-algorithms for their solution extend the application area of the Four-Level Model of Planning and Decision Making [21] and increase its efficiency for operative planning.
- The creation of new multi-stage models of discrete manufacturing systems, for which, based on the solutions obtained in this paper, it is possible to implement efficient scheduling and operative planning algorithms.
- Based on the created hierarchical and multi-stage models of scheduling and operative planning for discrete manufacturing systems, we can create universal software to be used in real manufacturing problem solving.

ACKNOWLEDGEMENTS

Funding: This research was funded by the Faculty of Electrical and Computer Engineering, Cracow University of Technology, and the Ministry of Science and Higher Education, Republic of Poland (grant no. E-1/2024).

The rights to the material cited under references [8] and [10] are owned by a third party – Springer Nature Switzerland AG. Used by permission.

REFERENCES

- [1] J. Leng and P. Jiang, "Dynamic scheduling in RFID-driven discrete manufacturing system by using multi-layer network metrics as heuristic information," *J. Intell. Manuf.*, vol. 30, no. 3, pp. 979–994, Feb. 2017, doi: [10.1007/s10845-017-1301-y](https://doi.org/10.1007/s10845-017-1301-y).
- [2] T. O'Brien, "Discrete Manufacturing Systems: High-Level Requirements." [Online]. Available: <https://www.visualsouth.com/blog/discrete-manufacturing-systems-requirements> [Accessed: 12. Mar. 2023].
- [3] X. Li and L. Gao, "Introduction for Integrated Process Planning and Scheduling," in *Effective Methods for Integrated Process Planning and Scheduling*, 1st ed., X. Li and L. Gao, Eds. Engineering Applications of Computational Methods, vol. 2. Berlin, Heidelberg: Springer, 2020, pp. 1–15, doi: [10.1007/978-3-662-55305-3_1](https://doi.org/10.1007/978-3-662-55305-3_1).
- [4] S. Ceschia, L. Di Gaspero, and A. Schaerf, "Solving discrete lot-sizing and scheduling by simulated annealing and mixed integer programming," *Comput. Ind. Eng.*, vol. 114, pp. 235–243, Dec. 2017, doi: [10.1016/j.cie.2017.10.017](https://doi.org/10.1016/j.cie.2017.10.017).
- [5] X. Chi, S. Liu, and C. Li, "Research on optimization of unrelated parallel machine scheduling based on IG-TS algorithm," *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 70, no. 4, p. e141724, Jun. 2022, doi: [10.24425/bpasts.2022.141724](https://doi.org/10.24425/bpasts.2022.141724).
- [6] A. Burduk, K. Musiał, A. Balashov, A. Batako, and A. Safoynyk, "Solving scheduling problems with integrated online sustainability observation using heuristic optimization," *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 70, no. 6, p. e143830, Dec. 2022, doi: [10.24425/bpasts.2022.143830](https://doi.org/10.24425/bpasts.2022.143830).
- [7] M.Z. Zgurovsky and A.A. Pavlov, *Trudnoreshayemyye zadachi kombinatornoy optimizatsii v planirovanii i prinyatii resheniy (Intractable Combinatorial Optimization Problems in Planning and Decision Making)*. Kyiv: Naukova dumka, 2016. (in Russian).
- [8] M.Z. Zgurovsky and A.A. Pavlov, "Introduction," in *Combinatorial Optimization Problems in Planning and Decision Making: Theory and Applications*, 1st ed., M.Z. Zgurovsky and A.A. Pavlov, Eds. Studies in Systems, Decision and Control, vol. 173. Cham: Springer, 2019, pp. 1–14, doi: [10.1007/978-3-319-98977-8_1](https://doi.org/10.1007/978-3-319-98977-8_1).
- [9] A.A. Pavlov, E.B. Misura, and T.N. Lisetsky, "Sostavleniye raspisaniya vypolneniya nezavisimyykh zadaniy identichnymi parallel'nymi priborami, momenty zapuska kotorykh men'she obshchego direktivnogo sroka (Scheduling independent tasks on the identical parallel machines which starting times are smaller than a common due date)," *Visnyk NTUU "KPI". Informatics, Operation and Computer Science*, no. 58, pp. 24–28, 2013. [Online]. Available: <https://ela.kpi.ua/handle/123456789/15486> [Accessed: 12 Mar 2023]. (in Russian).
- [10] M.Z. Zgurovsky and A.A. Pavlov, "The total tardiness of tasks minimization on identical parallel machines with arbitrary fixed times of their start and a common due date," in *Combinatorial Optimization Problems in Planning and Decision Making: Theory and Applications*, 1st ed., M.Z. Zgurovsky and A.A. Pavlov, Eds. Studies in Systems, Decision and Control, vol. 173. Cham: Springer, 2019, pp. 265–290, doi: [10.1007/978-3-319-98977-8_6](https://doi.org/10.1007/978-3-319-98977-8_6).

- [11] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co., 1990.
- [12] Yu.P. Adler, E.V. Markova, and Yu.V. Granovskiy. *Planirovaniye eksperimenta pri poiske optimal'nykh usloviy (Planning an Experiment in the Search for Optimal Conditions)*, 2nd ed. Moscow: Nauka, 1976. (in Russian).
- [13] A.A. Pavlov and E.B. Misura, "Minimizatsiya summarnogo zapazdyvaniya pri vypolnenii nezavisimyykh zadaniy s obshchim direktivnym srokom identichnymi parallel'nymi priborami, momenty zapuska kotorykh proizvol'ny (The total tardiness minimization when processing independent tasks with a common due date on the identical parallel machines with an arbitrary starting times)," *Visnyk NTUU "KPI". Informatics, Operation and Computer Science*, no. 59, pp. 28–34, 2013. [Online]. Available: <https://ela.kpi.ua/handle/123456789/15526> [Accessed: 12 Mar 2023]. (in Russian).
- [14] Z. Jia, Y. Cui, and K. Li, "An ant colony-based algorithm for integrated scheduling on batch machines with non-identical capacities," *Appl. Intell.*, vol. 52, no. 2, pp. 1752–1769, Jan 2022, doi: [10.1007/s10489-021-02336-z](https://doi.org/10.1007/s10489-021-02336-z).
- [15] M. Haroune, C. Dhib, E. Neron, A. Soukhal, H.M. Babou, and F.M. Nanne, "A Hybrid Heuristic for a Two-Agent Multi-Skill Resource-Constrained Scheduling Problem," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 5, 2022, doi: [10.14569/ijacsa.2022.01305104](https://doi.org/10.14569/ijacsa.2022.01305104).
- [16] J.C.S.N. Pinheiro, J.E.C. Arroyo, and L.B. Fialho, "Scheduling unrelated parallel machines with family setups and resource constraints to minimize total tardiness," in *Proc. 2020 Genetic and Evolutionary Computation Conference Companion (GECCO)*, 2020, pp. 1409–1417, doi: [10.1145/3377929.3398150](https://doi.org/10.1145/3377929.3398150).
- [17] M. Đurasević and D. Jakobović, "Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey," *Artif. Intell. Rev.*, vol. 56, no. 4, pp. 3181–3289, 2022, doi: [10.1007/s10462-022-10247-9](https://doi.org/10.1007/s10462-022-10247-9).
- [18] V.S. Tanaev and V.V. Shkurba, *Vvedeniye v teoriyu raspisaniy (Introduction to scheduling theory)*. Moscow: Nauka, 1975. (in Russian).
- [19] Y. Ouazene, F. Yalaoui, H. Chehade, and A. Yalaoui, "Workload balancing in identical parallel machine scheduling using a mathematical programming method," *Int. J. Comput. Intell. Syst.*, vol. 7, no. Supplement 1, p. 58, 2014, doi: [10.1080/18756891.2013.853932](https://doi.org/10.1080/18756891.2013.853932).
- [20] A.A. Pavlov, O.G. Zhdanova, and M.O. Sperkach, "Zadacha sostavleniya dopustimogo raspisaniya s maksimal'no pozdnim momentom zapuska vypolneniya identichnymi parallel'nymi priborami rabot s obshchim direktivnym srokom (The problem of creating a feasible schedule with maximum late moment of starting the jobs with common due date on identical parallel machines)," *Visnyk NTUU "KPI". Informatics, Operation and Computer Science*, no. 61, pp. 93–102, 2014. [Online]. Available: <https://ela.kpi.ua/handle/123456789/16721> [Accessed: 12 Mar 2023]. (in Russian).
- [21] M.Z. Zgurovsky and A.A. Pavlov, "Algorithms and Software of the Four-Level Model of Planning and Decision Making," in *Combinatorial Optimization Problems in Planning and Decision Making: Theory and Applications*, 1st ed., M.Z. Zgurovsky and A.A. Pavlov, Eds. Studies in Systems, Decision and Control, vol. 173. Cham: Springer, 2019, pp. 407–518, doi: [10.1007/978-3-319-98977-8_9](https://doi.org/10.1007/978-3-319-98977-8_9).
- [22] M. Sterna, "Late and early work scheduling: A survey," *Omega*, vol. 104, p. 102453, 2021, doi: [10.1016/j.omega.2021.102453](https://doi.org/10.1016/j.omega.2021.102453).
- [23] X. Chen, M. Sterna, X. Han, and J. Blazewicz, "Scheduling on parallel identical machines with late work criterion: Offline and online cases," *J. Sched.*, vol. 19, no. 6, pp. 729–736, 2015, doi: [10.1007/s10951-015-0464-7](https://doi.org/10.1007/s10951-015-0464-7).